UNIVERSIDAD POLITÉCNICA DE MADRID
FACULTAD DE INFORMÁTICA

Departamento de Lenguajes y Sistemas Informáticos e Ingeniería del Software

MIMO: Propuesta de un Metamodelo de Objetos y su Aplicación al Diseño de Bases de Datos

TESIS DOCTORAL

Autora: Esperanza Marcos Martínez
Directora: Adoración de Miguel Castaño
Coodirector: Mario G. Piattini Velthuis

1997

ÍNDICE

Resumen

Actualmente existen numerosos modelos de objetos procedentes de metodologías de desarrollo, de implementaciones de productos, de estándares, etc. Sin embargo existen diversos motivos que nos han llevado a la definición de un nuevo modelo de objetos. Por una parte, los modelos existentes, siempre mejorables, no resuelven todos los problemas de modelado de sistemas de información; estos problemas serán más fácilmente resolubles con modelos que proporcionen mayor capacidad semántica. Por otra parte, si bien es cierto que la orientación al objeto difumina las fronteras existentes entre las distintas fases del desarrollo, no elimina la necesidad de transformación de un esquema conceptual a un esquema de implementación. No cabe duda que la utilización de un único modelo en los diferentes niveles de abstracción del desarrollo facilitaría la transición entre las distintas etapas, así como la automatización de dicho proceso de transición. Sin embargo, ninguno de los modelos estudiados soporta el modelado a distintos niveles de abstracción.

A fin de paliar los problemas anteriormente expuestos, en la presente Tesis Doctoral se define MIMO (Metamodelo para la Integración de Modelos de Objetos), un nuevo modelo de objetos que soporta el modelado a nivel conceptual y de implementación, suavizando la transición entre las distintas etapas del desarrollo, y mejorando la capacidad expresiva con respecto a los modelos existentes en ambas fases del desarrollo. Con el fin de no aumentar más la heterogeneidad de modelos existente en la actualidad, MIMO se plantea como un aglutinador de los principales modelos (Método Unificado, SQL3 y ODMG-93). Se pretende con ello, utilizar MIMO como marco de referencia para establecer la equivalencia semántica entre los dos modelos de implementación estándar (SQL3 y ODMG-93) y como un primer paso hacia la automatización de la traducción de esquemas entre uno y otro modelo.

En la presente Tesis se define MIMO y se realiza una validación parcial del metamodelo. Dicha validación se lleva a cabo mediante la implementación de una herramienta de modelado conceptual en MIMO, la implementación parcial de MIMO como modelo de una metabase y el estudio comparativo de MIMO con los tres modelos que integra. Todo ello se realiza siguiendo un método de trabajo, definido dentro de la misma Tesis, que se basa en el método experimental de investigación científica.

Abstract

There are currently numerous object models coming from development methodologies, product implementations, standards, etc. Nonetheless, there are several reasons to define a new object model. On the one hand, existing models, always subject to improvement, do not solve all information system modelling problems; this problems will be more easily solvable by models providing a greater semantic capacity. On the other hand, although object orientation blurs the frontiers between development phases, it does not prevent the need to transform a conceptual schema into an implementation schema. The use of a single model in the different abstraction levels would undoubtedly ease the transition between phases, and the automation of the aforementioned transition process. Nonetheless, none of the models known to the author supports modelling at different abstraction levels.

With the aim to alleviate the aforementioned problems, MIMO (a Metamodel for the Integration of object MOdels) is defined in the present Thesis. It is a new object model supporting modelling at conceptual and implementation levels, smoothing the transition between different development phases, and improving the expressiveness with respect to existing models in both development phases. With the aim not to enhance further the current model heterogeneity, MIMO agglutinates the most relevant models (Unified Method, SQL3, ODMG-93). As a result, it might be possible to use MIMO as a framework of reference to establish the semantic equivalence between the standard implementation models (SQL3 and ODMG-93), and as a first step towards the automation of the schemata translation between both models.

Within this Thesis MIMO is defined, and a partial validation of the metamodel is provided. This validation is carried out by means of the implementation of a MIMO conceptual modelling tool, the partial implementation of MIMO as a metabase model, and the comparative study of MIMO with respect to the three models it integrates. This is accomplished observing a work method, defined within this very Thesis, based on the scientific research experimental method.

1 Introducción

"Para trazar un límite en el pensamiento tendríamos que ser capaces de pensar ambos lados de este límite, y tendríamos por consiguiente que ser capaces de pensar lo que no se puede pensar".

Wittgenstein, *El Tractatus*

En la presente Tesis Doctoral se propone un metamodelo de objetos, MIMO (Metamodelo para la integración de Modelos de Objetos), que soporta el modelado en los distintos niveles de abstracción de una base de datos, integrando los principales modelos de objetos existentes en la actualidad.

En este primer capítulo se realiza una introducción, en la que se analizan, epígrafe 1.1, los motivos que nos han llevado a la necesidad de un nuevo modelo de objetos, así como las aportaciones que MIMO supone respecto a los modelos existentes. En el apartado 1.2 se presentan la hipótesis y objetivos del trabajo. El epígrafe 1.3 es un resumen del marco de trabajo en el que se enmarca la Tesis.

1.1. Planteamiento y justificación del trabajo

El paradigma de la orientación al objeto, como afirma Soloviev (1992), ha involucrado a la mayor parte de desarrolladores e investigadores en el entorno del software de bases de datos. Esta concepción de los datos se ha presentado, no sólo como una posibilidad de solucionar gran parte de las limitaciones de las bases de datos relacionales, sino también como una mejora sustancial de las técnicas de análisis, aportando una capacidad semántica superior a la proporcionada por el modelo E/R. Pero, a diferencia de lo sucedido con el modelo relacional, las bases de datos orientadas al objeto se implantan en el mercado sin la existencia previa de un modelo teórico que las sustente. Por este motivo, a pesar de mantener unos principios comunes, los modelos en los que se apoyan los distintos productos comerciales son diferentes. En Bancilhon y Ferran (1994) se señala el riesgo de divergencia entre los diferentes productos y se expresa tanto la necesidad de convergencia entre ellos como la de definición de estándares. En Vossen (1995) se afirma que mientras los sistemas de gestión de bases de datos orientados al objeto han llegado ya al mercado, sus fundamentos teóricos están todavía en fase de desarrollo, motivo por el cual se hace necesaria la existencia de un modelo formal. Aunque no de un modo estrictamente formal, son varios los modelos propuestos, lo que ha provocado la aparición de un nuevo problema: la heterogeneidad.

A la heterogeneidad producida por el número de modelos existentes (algunos definidos como modelos teóricos y otros procedentes de productos comerciales) se unen los problemas de heterogeneidad derivados de la clásica separación entre los sistemas orientados al objeto "puros", Atkinson et al. (1989), y los procedentes de extensiones a los sistemas relacionales, Stonebraker et al. (1990). De los modelos de estándares para sistemas comerciales, los más relevantes son: ODMG-93 (estándar para sistemas de gestión de bases de objetos) en Cattel (1994a) y Cattell (1995) y el modelo de objetos del SQL3 (futuro estándar para bases de datos relacionales, extendidas con capacidades de orientación al objeto) en DBL: MAD-004 (1996) y DBL: MAD-010 (1996).

Cada uno de estos estándares, como se afirma en Manola y Mitchell (1994), ha tenido una gran importancia en sí mismo, pero se hace necesario que los sistemas que

los soporten sean capaces de interoperar reduciendo las diferencias existentes entre ellos. Además de estos autores, otros muchos, Kim (1994), Cattell (1994b), Melton (1994), propugnan la necesidad de acercamiento entre el SQL3 y el ODMG-93, cuyo objetivo principal es permitir la interoperabilidad entre bases de objetos. En la actualidad se ha constituido un grupo de convergencia formado por distintos integrantes del ODMG y del comité de estandarización del SQL3. Dicho grupo se ha reunido ya en diversas ocasiones y los resultados del trabajo realizado se recogen en DBL: YOW-031 (1995), DBL: LHR-079 (1995) y DBL: MCI-079 (1996).

Aunque en la actualidad han surgido varios modelos de objetos (algunos de ellos como modelos de implementación) en nuestra opinión, es difícil que uno de estos modelos se imponga sobre los demás. Por ello, pensamos que la idea de llegar en un futuro próximo a un único modelo es utópica y que es mejor trabajar para conseguir una integración de los ya existentes. Para ello, se propone el desarrollo de un Modelo que Integre los principales Modelos de Objetos de la actualidad (MIMO) como un marco a partir del cual pueden derivarse otros modelos de objetos, de un modo similar a la propuesta de Chen (1976) donde el modelo E/R, con mayor capacidad expresiva que los modelos existentes, se planteaba para ser utilizado como base y a fines de unificación de diferentes vistas de los datos (red, relacional y conjunto de entidades -entity set-).

La integración, no sólo de modelos, sino también de metodologías, es una de las principales líneas de trabajo en la actualidad. Algunos ejemplos[1] de esta tendencia son: SUMM(1992), STEP/EXPRESS en Spiby (1994), el Método Unificado en Booch y Rumbaugh (1995), COMMA/OPEN en Henderson-Sellers (1996).

MIMO integra los modelos del SQL3 y ODMG-93 y se propone como un marco a partir del cual derivar esquemas SQL3 y ODMG-93, de igual modo que el modelo E/R se propuso como un marco a partir del cual derivar esquemas relacionales, en red o entity-set. Los dos estándares citados presentan diferencias esenciales entre sí debido a que provienen de distintos paradigmas. El SQL3, procedente del entorno de las bases de datos relacionales, se obtiene a partir de extensiones del modelo relacional, mientras que

[1] Todos ellos se estudiarán con más detalle en el capítulo 3.

el ODMG-93 procede de los SGBDOO (Sistemas de Gestión de Bases de Datos Orientadas al Objeto) y surge con el fin de integrar un modelo de objetos con un lenguaje de programación existente. Ambos, al nacer con la finalidad de estandarización de productos comerciales, se centran en aspectos relativos a la implementación por lo que su capacidad semántica es muy restringida. Uno de los objetivos fundamentales de MIMO es aumentar la capacidad de expresión respecto a los modelos actuales, incluso a nivel de implementación.

MIMO se concibe como un modelo que además de integrar los principales modelos de objetos existentes, soporta el modelado en todas las fases de desarrollo de una base de datos. Para ello es necesario considerar otros modelos que cubran la fase de diseño conceptual. MIMO se basa, principalmente, en el Método Unificado (MU), Booch y Rumbaugh (1995). Sin embargo, el método unificado está muy influenciado por los lenguajes de programación (principalmente el C++), por lo que se centra principalmente en aspectos relativos a las aplicaciones, dejando de lado algunos aspectos importantes en el modelado de bases de datos. Así, por ejemplo, aunque soporta restricciones, no lo hace de un modo riguroso ni sistemático; apenas se soportan restricciones específicas (es decir, restricciones explícitamente definidas dentro del método), sino que en general se permite la definición de cualquier restricción como si se tratase de un literal. El modelado de restricciones en bases de datos es importante y, pensamos, que es necesario hacerlo de un modo sistemático, distinguiendo entre restricciones de clase o de atributo, entre aserciones y disparadores, etc. Sin embargo, el MU si trata de un modo especial conceptos directamente importados del C++, como el de “clase amiga”, y cuya importancia en el modelado de una base de datos es, cuando menos, discutible.

La mayoría de las metodologías que se pueden encontrar en la literatura, Shaller y Mellor (1990), Jacobson (1993), Booch (1994), Coleman et al. (1994), Graham (1994), Robinson y Berrisford (1994), Yourdon et al. (1995), Martin y Odell (1995), Henderson-Sellers y Edwars (1995), Henderson-Sellers et al. (1995), Henderson-Sellers (1996), etc., tienen las mismas carencias en cuanto a bases de datos se refiere. Por ello, MIMO

se basa también en el modelo definido en MEDEA, Piattini (1994), una metodología orientada al objeto definida en un trabajo de tesis doctoral dentro de este mismo grupo de investigación, que se centra en aspectos de análisis y diseño de bases de datos.

La orientación al objeto difumina las fronteras existentes entre las etapas de análisis, diseño e implementación, pero dichas fronteras siguen existiendo debido, en gran parte, a la necesidad de cambiar de modelo de una a otra etapa. En KADS-II (1993) se pone de manifiesto la necesidad de dos tipos de modelos en la ingeniería del conocimiento y sistemas expertos: ***modelos conceptuales*** (cuyo objetivo es la representación del conocimiento independientemente de la implementación) y ***modelos de diseño*** (son modelos orientados al producto y que se obtienen de una transformación del correspondiente modelo conceptual). Los modelos de diseño deben poder llevarse a código ejecutable sin cambios substanciales. Igualmente, se pone de manifiesto la necesidad de desarrollar métodos de transformación entre un modelo conceptual y un modelo de diseño. Dicha transformación podrá ser manual o, preferiblemente, automática. En nuestra opinión, todo lo dicho para sistemas de conocimiento es igualmente aplicable a los sistemas de información. En este ámbito, como ya se ha explicado, existen modelos conceptuales y modelos de diseño e implementación, pero hay que señalar la importancia de la necesidad de métodos que permitan automatizar la transformación de unos a otros. En Blum (1996) se pone también de manifiesto la necesidad de este tipo de automatización y se afirma que, una meta razonable es la de trabajar con un modelo conceptual y automatizar la creación del producto sin una especificación intermedia. Con MIMO se pretende solucionar, en parte, este problema. Aunque no es un objetivo del presente trabajo llegar a la definición y automatización de la traducción de un esquema de análisis en MIMO a una implementación (ya que ello llevaría consigo la elaboración de una metodología completa de desarrollo), es evidente que un modelo que soporte de modo uniforme la definición de esquemas en cada etapa de desarrollo facilitará de un modo sustancial esta tarea. La definición de una metodología de desarrollo apoyada en MIMO se dejará para futuros trabajos.

Además, MIMO tratará de aumentar la capacidad semántica respecto a los modelos existentes en la actualidad. Es cierto que la orientación al objeto ha contribuido

substancialmente a aumentar el poder de expresión respecto a los anteriores paradigmas; sin embargo, aún es posible obtener modelos con mayor capacidad semántica. Un modelo de tales características proporciona mecanismos de representación del conocimiento con un gran nivel de abstracción, lo que facilita en gran medida la resolución de problemas complejos ya que, como se afirma en Jesse et al. (1996), *"el nivel de abstracción con el que seamos capaces de hablar afecta directamente al tamaño del problema que podremos resolver"*. MIMO, debido a que soporta los distintos niveles en el desarrollo de una base de datos, aumenta la capacidad de expresión no sólo a nivel de análisis, sino también a nivel de diseño e implementación. Los modelos de implementación en orientación al objeto, en general, dejan una parte importante de la semántica (por ejemplo, la de las restricciones) para ser recogida mediante operaciones, con los inconvenientes que de ello se derivan. En Calvanese y Lenzerini (1994) se plantea la necesidad de aumentar la expresividad de los modelos de implementación orientados al objeto. Nosotros pensamos que este aumento de expresividad debe llevarse a cabo en todas las etapas de desarrollo.

En resumen diremos que el trabajo de investigación que aquí se propone se plantea como solución a los siguientes problemas:

1. Inexistencia de un modelo que, de un modo uniforme, soporte los distintos niveles de abstracción de todas las etapas del desarrollo de una base de datos.

1. Carencia de un modelo que se centre en el desarrollo de bases de datos.

1. Insuficiente capacidad semántica de los modelos de objetos existentes, especialmente en la fase de implementación.

1. Inexistencia de un modelo, que junto con una metodología, permita automatizar la transformación de un modelo conceptual a un modelo de implementación.

1. Heterogeneidad de modelos de objetos existente en la actualidad, facilitando la interoperabilidad entre bases da datos definidas según diferentes estándares.

Para ello se propone un nuevo modelo (MIMO) que permite la definición de esquemas en dos niveles de abstracción diferentes: nivel de análisis y nivel de construcción[2]. MIMO, a fin de no aumentar la heterogeneidad de modelos ya existentes sino con el objetivo de contribuir a eliminarla, se concibe como una integración[3] de los principales modelos de objetos actuales. MIMO integra el MU/UML (3.2.4) en análisis y construcción, y SQL3 (3.2.3) y ODMG-93 (3.2.2) en construcción. Además incorpora aspectos importantes de otros modelos entre los que cabe destacar OOM (3.2.5), MEDEA (3.2.7) y STEP/EXPRESS (3.2.6). La figura 1.1 muestra cómo MIMO soporta distintos niveles de abstracción, aglutinando los modelos del MU, SQL3 y ODMG-93, Marcos et al. (1997a).

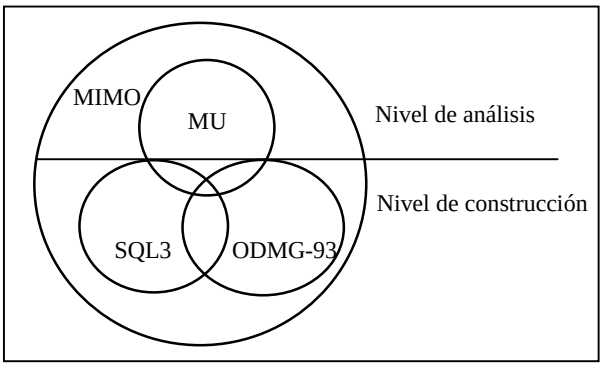


Figura 1.1: Integración de modelos en MIMO

MIMO aumenta la capacidad semántica respecto a los modelos existentes en sus distintos niveles debido, por una parte a que integra varios modelos (por lo que su semántica será mayor que la de cada uno de ellos), y por otra a la incorporación de aspectos semánticos no incluidos en los modelos anteriores.

En SQL/MM: MAD-024 (1996) se describe la estrategia del grupo ISO/TC211 para la descripción de esquemas conceptuales de información espacial (cabe destacar, dentro de éstos, los Sistemas de Información Geográficos, SIG[4]). En dicho documento

[2] Este nivel incluye los niveles de diseño e implementación. Hablaremos de **niveles,** de análisis, construcción y explotación, en lugar de **fases** debido a que MIMO es un modelo y no un método. MIMO soporta todos los constructores que un modelo necesita para cada fase del desarrollo de una base de datos.
[3] El término **integración** se utiliza también, en el entorno de las bases de datos, con otros significados, como por ejemplo, para hacer referencia a la integración de esquemas en base de datos federadas y heterogéneas. Queremos subrayar que aquí la integración se refiere a **integración de modelos de datos** y no a integración de esquemas.
[4] GIS (Geographic Information System)

se plantean los requisitos para un lenguaje para la descripción de información geográfica y se analizan diferentes lenguajes existentes, para llegar a la conclusión de que ninguno de ellos, independientemente, cumple las necesidades deseadas. En concreto se estudian: EXPRESS (3.2.6), IDEF1X (1993), INTERLIS[5], OMG/IDL (3.2.1), ODMG-93/ODL (3.2.2), OMT en Rumgaugh et al. (1992), Syntropy en Cook y Daniels (1994), MU/UML (3.2.4), COMMA (3.2.8.3), OPEN (3.2.8.4), OOram-ODP (3.2.8.2) y SQL3/MM[6]. Se propone como solución la definición de un lenguaje que soporte un metamodelo en el que se integren, ver figura[7] 1.2, los modelos de algunos de los lenguajes estudiados: IDL/ODL, EXPRESS, INTERLIS e IDF1X. Esta es precisamente la solución que nosotros hemos adoptado[8].

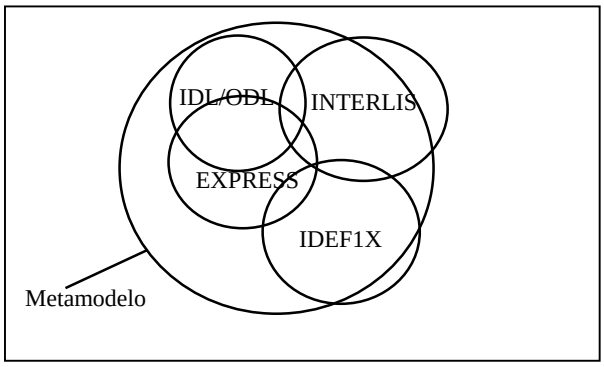


Figura 1.2: Integración de lenguajes propuesta en SQL/MM: MAD-024 (1996)

1.2. Hipótesis y objetivos

Después de este breve planteamiento de nuestra investigación en el que únicamente se ha pretendido proporcionar una visión general y una justificación del trabajo realizado, pasamos a exponer la hipótesis y los objetivos que se han fijado al comienzo de la investigación llevada a cabo en esta Tesis Doctoral.

La **hipótesis** que nos hemos planteado, después de un detenido estudio de la situación actual de los modelos de objetos, es que *es factible la definición de un modelo de objetos que integre los conceptos de los principales modelos existentes en las*

[5] En documento sin fecha
[6] Extensiones para tratamientos de datos multimedia del SQL3 (ver epígrafe 3.2.3)
[7] La figura está tomada de SQL/MM: MAD-024.

diferentes etapas del desarrollo de una base de datos, permitiendo así el modelado en cada una de éstas etapas, y que ofrezca una mayor capacidad semántica, facilitando la interoperabilidad de bases de datos definidas siguiendo distintos estándares (SQL3 y ODMG-93).

El **objetivo principal** de la investigación, derivado directamente de la hipótesis, es la definición de un modelo de objetos que soporte los distintos niveles del desarrollo de una base de datos, integrando los principales modelos de objetos de la actualidad y proporcionando mayor capacidad semántica que éstos sin aumentar excesivamente su complejidad. Es importante señalar que, debido a la dimensión y complejidad de un modelo de tales características, nos centraremos en la definición de la estática, dejando la dinámica para futuras ampliaciones del modelo.

Para la consecución de este objetivo se han planteado los siguientes **objetivos parciales**:

1. Analizar los distintos modelos de objetos existentes en la actualidad (en especial los modelos promulgados como estándares) determinando sus aportaciones así como sus limitaciones con el fin de delimitar cuales de ellos se integrarán en el modelo propuesto.

2. Definir un modelo de objetos (MIMO) que cubra todo el ciclo de desarrollo de una base de datos integrando los modelos de objetos previamente seleccionados y aumentando la capacidad semántica proporcionada por ellos.

3. Definir una notación gráfica que soporte todos los constructores del modelo

4. Verificación y validación[9] del modelo, para lo cual se fijan los siguientes objetivos parciales:

[8] Es importante destacar que este documento apareció cuando la presente tesis doctoral ya estaba en sus últimas fases.
[9] Ver capítulo 2 y capítulo 3, epígrafe 3.2, donde se discute la validación y verificación

4.1. Definir una primera aproximación a la correspondencia entre las estructuras de la fase de implementación de MIMO y las estructuras de los modelos de implementación que MIMO integre.

4.2. Implementar un prototipo de MIMO, como modelo de una herramienta de modelado conceptual OO.

4.3. Descripción del modelo en términos de sí mismo.

4.4. Implementar un prototipo de MIMO, como modelo de la metabase de una herramienta de diseño de bases de datos avanzadas (ver epígrafe 1.3).

1.3. Marco de la Tesis: el proyecto ENEAS/BD

La Tesis se enmarca dentro de un proyecto de investigación, denominado ENEAS/BD, que está siendo llevado a cabo por el grupo de Bases de Datos Avanzadas del Departamento de Informática de la Universidad Carlos III de Madrid. Este proyecto es continuación de una labor de investigación que viene realizándose desde hace varios años, bajo la coordinación de la directora del presente trabajo, y que ha dado lugar a varias tesis doctorales y proyectos fin de carrera.

Durante los años 1988 y 1989 se elaboró un proyecto sobre una metodología de diseño para bases de datos relacionales, Piattini (1989), De Miguel y Piattini (1992a), al mismo tiempo que se estudiaban los modelos de referencia de ANSI (1986) y ANSI (1990), con especial énfasis en la descripción recursiva de los datos, De Miguel y Piattini (1988) y De Miguel y Piattini (1991a). También se analizaban las extensiones semánticas del modelo E/R básico, Hidalgo (1991).

En 1990 se inició el proyecto DELFOS[10] para el desarrollo de un Diccionario de Recursos de Información, lo que nos llevó a estudiar bibliografía sobre el paradigma de la

[10] El proyecto DELFOS, parcialmente subvencionado por el Ministerio de Industria, Comercio y Turismo (dentro de las actuaciones de estímulo a la investigación e innovación tecnológicas de la Dirección General de Electrónica y Nuevas Tecnologías), por el IMADE (Instituto Madrileño de Desarrollo) y por el CDTI

orientación al objeto -con el fin de analizar su posible aplicación al proyecto- al tiempo que profundizábamos sobre los estándares relativos a los SDRI. En el proyecto, que finalizó en abril de 1993, se implementó un SDRI basado en las normas ISO 10027 y 10728, con una arquitectura relacional.

El estudio sobre la orientación al objeto y los SDRI continuó y en 1991 se precisaron dos temas concretos de investigación que iban a constituir el núcleo de dos tesis doctorales: el primero, sobre una metodología de desarrollo orientado al objeto MEDEA, Piattini (1994), y un segundo trabajo sobre el diseño de un SDRI orientado al objeto[11]. Además, se comenzaron una serie de proyectos de fin de carrera relacionados con los SGBDOO. Entre estos proyectos cabe destacar Marcos (1992), en el que se realizaba un análisis de la orientación al objeto en el borrador del estándar SQL3 y que constituyó uno de los comienzos de la investigación actual.

En 1994 se comienza con el proyecto ENEAS/BD, De Miguel et al. (1996a), en el que se pretende la integración de varios de los trabajos realizados durante los años anteriores. ENEAS/BD, ENtorno para la Enseñanza Avanzada de Sistemas de Bases de Datos, es un sistema informático concebido con dos objetivos principales: ayudar al usuario en el diseño de bases de datos y proporcionar un entorno de aprendizaje en este área. Para ello se integran distintas técnicas de diseño de bases de datos en un único sistema, permitiendo al usuario componerse su propia metodología de desarrollo. En la figura 1.3 se muestra la arquitectura de ENEAS/BD la cual se compone de distintos módulos que se integran por medio de una metabase.

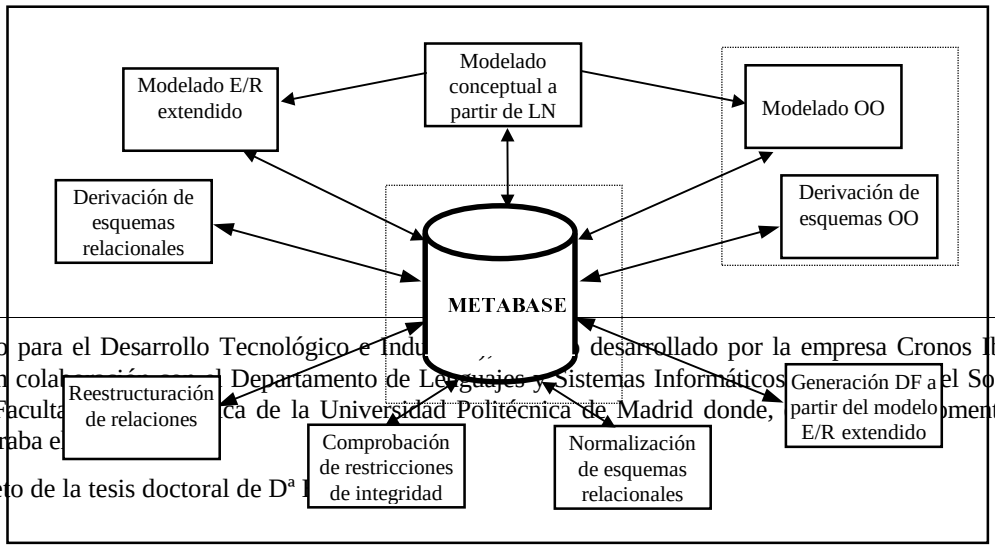


Figura 1.3: Integración de módulos de ENEAS/BD por medio de la

(Centro para el Desarrollo Tecnológico e Indu[illegible] desarrollado por la empresa Cronos Ibérica, S.A. en colab[illegible] el Departamento de Lenguajes y Sistemas Informáticos de la Facult[illegible]ca de la Universidad Politécnica de Madrid donde, [illegible]omentos, se encontraba e[illegible]

[11] Objeto de la tesis doctoral de Dª I[illegible]

El núcleo del proyecto ENEAS/BD reside en la metabase, ya que es ella la que se encarga de realizar la integración y conexión entre todos los módulos, los cuales pueden operar también de forma aislada. Cada uno de ellos se puede comunicar con los otros a través de la metabase. MIMO será el modelo que soporte la metabase ya que su capacidad expresiva permitirá recoger, con una representación fiel y sencilla, los conceptos de los módulos de ENEAS/BD (que corresponderán a distintas etapas del desarrollo).

En la figura 1.3 se señala, en trazo discontinuo, los módulos del proyecto en los que se enmarca este trabajo de Tesis Doctoral. El módulo de ***modelado OO*** obtiene un análisis (en MIMO), bien a partir de especificaciones de usuario en lenguaje natural[12], De Miguel et al. (1996b), bien introducidas directamente a través de un interfaz gráfico que dicho módulo soporta. El módulo de ***derivación de esquemas OO*** convierte un análisis MIMO a una implementación MIMO, a partir de la cual se generará el correspondiente código ODMG-93 o SQL3[13], Marcos et al. (1997a). El objetivo final es acercarse, en la medida de lo posible, a la automatización del proceso de traducción de un análisis MIMO a una implementación SQL3 u ODMG-93.

Se necesitaba pues un modelo que, a fin de proporcionar las funcionalidades que ENEAS/BD requiere, soportase los conceptos de todas las etapas del desarrollo de una base de datos (para poder ser el modelo de la metabase) integrando los modelos del SQL3 y del ODMG-93 (para poder generar esquemas de ambos estándares). Todo ello,

[12] En este módulo se enmarca otra Tesis Doctoral, a punto de finalizar, que se está llevando a cabo dentro del mismo grupo de Bases de Datos Avanzadas y cuyo objetivo principal es obtener un modelo conceptual (E/R y OO) a partir de especificaciones de usuario en lenguaje natural.

[13] MIMO debe integrar, como modelos de diseño e implementación, los de los principales estándares de bases de datos orientadas al objeto: SQL3 y ODMG-93.

proporcionando la mayor capacidad expresiva posible (para facilitar el modelado conceptual) y manteniendo esta semántica en diseño e implementación. Al no existir un modelo de tales características se inició el proceso de definición de un nuevo modelo, MIMO, que se ajustara a tales necesidades y el cual ha dado lugar a la Tesis Doctoral que se presenta.

2 Método de trabajo

"Por regla general, empiezo mis clases sobre el Método Científico diciendo a mis alumnos que el método científico no existe".

Popper, *Realismo y el Objetivo de la Ciencia*

La distinta la naturaleza del saber de las ingenierías, con respecto al saber de las ciencias empíricas y formales, hace que los métodos de investigación de éstas últimas ciencias no sean directamente aplicables a la investigación en el ámbito de la ingeniería del software.

En este capítulo se discute la necesidad de un nuevo método de investigación que se adapte al saber de las ingenierías, y más concretamente al saber en la ingeniería del software, y se propone un método de investigación que se ajusta al problema concreto que nos ocupa.

2.1. Necesidad de un método de investigación para la ingeniería del software

Para la realización del presente trabajo se han estudiado diversos métodos[1] de investigación científica, en el supuesto, casi universal, de que toda tesis doctoral responde a un trabajo de investigación científica. Sin embargo, ninguno de los métodos tradicionales, ver a título de ejemplo Chalmers (1984), es válido para abordar el problema que aquí se plantea. Un método científico es aplicable a la investigación en disciplinas que son en sí mismas científicas. Sin embargo, cuando la naturaleza del conocimiento tratado no es científica[2], como es el caso de las ingenierías, tampoco son aplicables los mismos métodos de investigación. El Real Decreto 185/1985, de 23 de enero, sobre Obtención y Expedición del Título de Doctor y otros Estudios de Postgrado (B.O.E. de 16 de febrero de 1985) especifica: *"La tesis doctoral consistirá en un trabajo original de investigación sobre una materia relacionada con el campo científico, técnico o artístico..."*. La materia de investigación de la presente Tesis no es, en nuestra opinión, científica en el sentido estricto de la palabra, sino que se encuadra en el campo técnico. Por ello, ninguno de los métodos de investigación propuestos en la literatura, que son distintas variantes de métodos de investigación científica, es válido para el presente trabajo. A continuación pasamos a exponer los motivos que nos han llevado a dicha conclusión:

[1] Aunque en ingeniería del software es más habitual el uso del término *"metodología"*, nosotros hemos preferido hablar de *"método"* ya que metodología, aunque también correcto, hace más referencia al estudio del método que al método en sí mismo. **Método**: *"Modo de decir o hacer con orden una cosa. Procedimiento que se sigue en las ciencias para hallar la verdad y enseñarla"*, RAE (1992). **Metodología**: *"Ciencia del método. Conjunto de métodos que se siguen en una investigación científica o en una exposición doctrinal"*, RAE (1992).

[2] Entendida ciencia, no como saber, sino en un sentido más estricto: *"Conocimiento cierto de las cosas por sus principios y causas"*, según el diccionario de la Real Academia Española, RAE (1992), o *"conocimiento sistematizado de las cosas, basado en el estudio y experimentación"* según el diccionario de la lengua de Anaya. Existen, efectivamente, otras acepciones del término ciencia que discutiremos posteriormente. Es importante señalar que no tratamos de plantear una discusión sobre lo que es, o no es, ciencia y si las ingenierías pueden o no considerarse dentro de esta rama del saber. Nuestro objetivo con la presente discusión es justificar el motivo por el que los tradicionales métodos de investigación científica no nos han servido para la presente investigación: la naturaleza del saber de la ingeniería es diferente de la naturaleza del saber en los campos del saber que tradicionalmente se han considerado científicos.

Las ciencias, tradicionalmente[3], se dividen en ciencias formales (lógica y matemáticas) y ciencias empíricas (entre las que se encuentran la biología, la química, etc., y según algunos autores, las ciencias humanas y sociales como la antropología o la historia). En ninguno de estos grupos se encuentran ni pueden encontrarse las ingenierías.

Las ***ciencias empíricas***, a fin de encontrar respuesta a numerosos interrogantes sin resolver, se centran en el estudio de objetos existentes para obtener, a través de la creación de hipótesis, la observación y la experimentación, respuestas a dichos interrogantes. Dependiendo de la naturaleza del interrogante la solución propuesta responde a un modelo explicativo diferente que, según Nagel (1974) puede ser: deductivo, probabilístico, teleológico y genético. Estos modelos de explicación dan lugar a los diversos métodos de investigación científica. Sin embargo, y a diferencia de lo que ocurre en las ciencias empíricas, una ingeniería *construye* nuevos objetos, los cuales podrán ser a su vez objetos de investigación científica. Según Gallego (1987), el objeto de la ciencia empírica *"reside fuera de nosotros y tiene una existencia en el mundo exterior"*. Por tanto, su conocimiento es de naturaleza fundamentalmente experimental y además, tal experimentación debe basarse en la realidad. Tal y como afirma Fetzer (1993): *"La ciencia necesita más que 'adecuación empírica' para tener éxito"*, necesita el conocimiento de la realidad. Éste no es el caso de las ingenierías, cuyo conocimiento tiene un importante componente de creatividad[4] que dificulta la elaboración de un método para la resolución de problemas dentro del ámbito tecnológico.

Es evidente que tampoco podemos clasificar las ingenierías como ***ciencias formales*** a pesar de que éstas constituyen una base imprescindible para aquellas. Las matemáticas son consideradas por muchos autores, ver Bunge (1985), como una disciplina de suma utilidad para cualquier otra ciencia, ya que permite la formulación de las teorías científicas en términos matemáticos. La matemática es como un esqueleto

[3] El diccionario de la RAE (1992) divide las ciencias en: exactas (matemáticas), humanas y naturales; en ninguna de estas clases pueden encuadrarse las ingenierías.

que puede ser usado por cualquier otra ciencia; el esqueleto que puede cubrirse con cualquier tipo de materia que sea compatible con aquella estructura formal. La lógica es habitualmente utilizada como lenguaje formal (ver 3.1.2), si bien es verdad que a partir de Russell (1903), con su idea de que la matemática es lógica, ambas áreas de conocimiento se han ido acercando.

De las distintas clasificaciones de ciencia que hemos podido encontrar, quizá, en la que mejor se encuadra la ingeniería del software (así como las ingenierías en general) es, paradójicamente, en la realizada por Aristóteles, ver Grene (1985). Aristóteles clasifica las ciencias[5] en diversas dimensiones, dos de la cuales son válidas para el caso que nos ocupa:

a) Según sus resultados, divide las ciencias en *teoréticas* si su resultado es saber (matemáticas, ciencias naturales...), *prácticas* cuando su resultado es una acción (ética, política...) y *productivas* cuando el resultado es un objeto (aquí se engloban las técnicas, como alfarería, escultura...). En estas últimas podríamos clasificar las ingenierías.

b) Por la substancialidad del objeto de estudio pueden dividirse en las que estudian *sustancias* (como la teología o las ciencias naturales), las que estudian *objetos que no son sustancias* (como las matemáticas, la ética o la política) y aquellas cuyo objeto de estudio son *entidades accidentales* (las técnicas). Las ingenierías las englobaríamos, de nuevo, entre las ciencias técnicas. En nuestro caso veremos (capítulo 4), que la cualidad accidental de los objetos que son susceptibles de estudio en nuestra investigación, nos va a permitir llegar a algunas conclusiones importantes.

En Bunge (1976) se presenta otra clasificación en la que se divide a las ciencias en *puras* y *aplicadas*, englobando dentro de éstas últimas a las tecnologías. Así, por

[4] Obsérvese la diferencia entre la *creación* en ciencia y en ingeniería: la ciencia crea hipótesis sobre objetos existentes; la ingeniería crea hipótesis sobre objetos que aún no existen y además crea el propio objeto que ha sido susceptible de estudio.

[5] Entendida ahora ciencia, no en su sentido estricto, sino como saber. Este es el motivo por el que la técnica en Aristóteles es una ciencia. La diferencia estriba en si consideramos ciencia en un sentido estricto o si cualquier saber es considerado ciencia.

ejemplo, Bunge considera la ingeniería eléctrica como una tecnología física, o la medicina como una tecnología biológica. No tenemos muy claro donde podría encuadrarse la ingeniería del software, así como otras ingenierías, dentro de esta clasificación, no sólo porque no es aplicación directa de una única *ciencia pura,* sino porque consideramos que la ingeniería del software, así como el resto de las ingenierías, no son mera aplicación de otras ciencias. Esta idea es defendida en Aracil (1986a), donde se hace una dura crítica a las corrientes que no consideran las ingenierías dentro del campo del saber de las ciencias. Sin embargo, Aracil sí establece distinción entre ciencia e ingeniería. Según este autor, la diferencia fundamental entre ambas es que mientras la primera se ocupa del estudio del cómo son las cosas, la segunda se ocupa del cómo deberían ser a fin de llegar a construir nuevos objetos y afirma que "las ciencias se ocupan de lo natural, mientras que el dominio específico de la ingeniería es lo artificial".

En nuestra opinión, lo que Aracil plantea es más bien una disquisición terminológica. El saber técnico es, en nuestra opinión, diferente que el saber de las tradicionales "ciencias", pero, en todo caso, "saber" al igual que ellas. La naturaleza de su conocimiento es diferente, en tanto que también su objeto de estudio es diferente, pero no de menor categoría. Podría ampliarse el concepto de ciencia e incluir en ella, además de las ciencias empíricas y las formales, las llamadas, en términos de Aracil, ciencias de la ingeniería. En el caso que nos ocupa es irrelevante, ya que los métodos de investigación aplicables a las ciencias tradicionales tampoco serían aplicables a las ciencias de la ingeniería. Lo importante es que ciencias e ingenierías son saberes y que la naturaleza de su conocimiento difiere en el objeto de su estudio y por tanto en sus métodos. Que el saber de las ingenierías, incluido o no en el saber científico, posee, cuando menos, la misma categoría que éstas.

Al igual que nosotros, Blum (1996) distingue entre ciencia y tecnología; detalla la relación existente entre ciencia y tecnología comparando la labor del ingeniero con la del científico, así como el conocimiento en cada una de éstas áreas. Blum afirma: *"rechazo la estrecha definición de ingeniería del software procedente de las ciencias de la computación; en efecto, yo propongo diseñar una nueva ciencia de la computación para la ingeniería del software..."* y continúa definiendo la ciencia de la tecnología de la computación como *"el estudio de la transformación de ideas en operaciones"*.

En conclusión, podemos decir, que si la naturaleza del conocimiento de nuestra investigación no se ajusta a ninguno de los tipos de conocimiento incluidos en las clasificaciones tradicionales de ciencia, tampoco podremos aplicar ningún método[6] tradicional de investigación científica[7]. Por todo ello, para la presente Tesis Doctoral hemos elaborado nuestro propio método de trabajo.

2.2. Definición de un método para la realización de nuestro trabajo

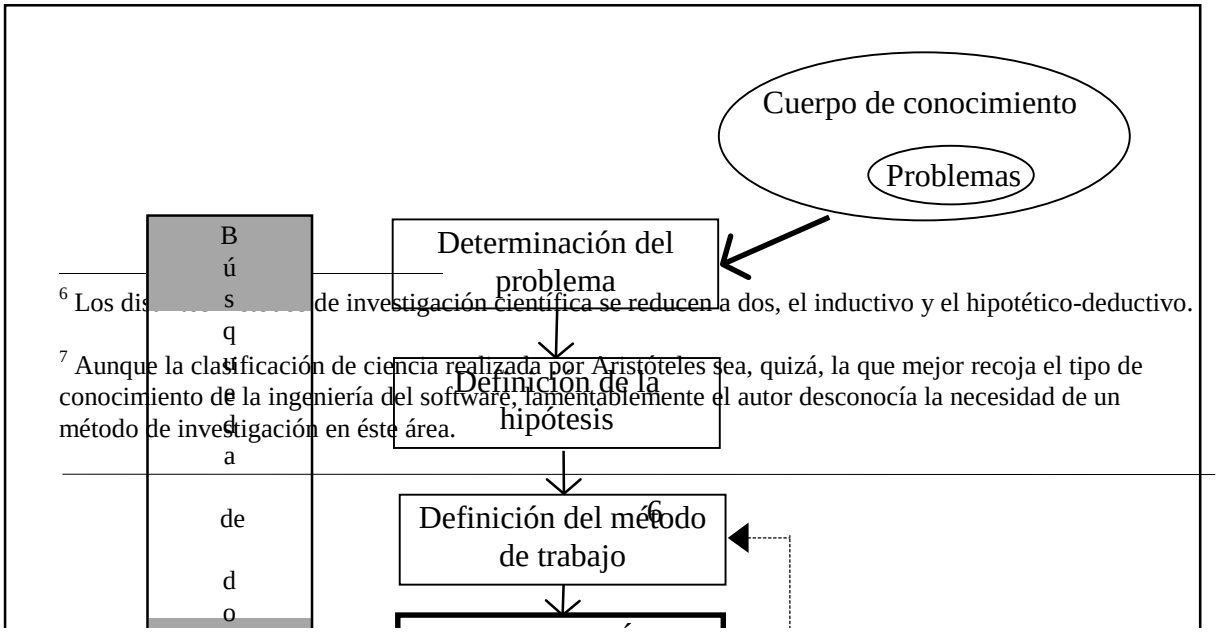
Realizaremos la exposición del método de trabajo en dos etapas:

a) se presenta el método de trabajo general

b) se profundiza en el método de resolución del problema concreto que nos ocupa

2.2.1. Método general de trabajo

En una primera aproximación se elabora un **método de trabajo general**, figura 2.1, para el que se ha tomado como base los pasos a seguir, según Bunge (1976), en toda investigación científica y que están basados en el método hipotético-deductivo. Estos pasos, por su generalidad son aplicables, con ciertas modificaciones, a cualquier tipo de investigación.



[6] Los dis… de investigación científica se reducen a dos, el inductivo y el hipotético-deductivo.

[7] Aunque la clasificación de ciencia realizada por Aristóteles sea, quizá, la que mejor recoja el tipo de conocimiento de la ingeniería del software, lamentablemente el autor desconocía la necesidad de un método de investigación en éste área.

a.1) Determinación del problema

El primer paso consiste en determinar, entre los problemas sin resolver dentro de nuestro campo de conocimiento, cuál se va a abordar. Para ello se seleccionan, en un primer momento, aquellos problemas cuya resolución puede dar lugar a un trabajo de investigación (según Bunge, a un trabajo de investigación científica). De entre ellos se elige uno que, en nuestra opinión, deberá cumplir las siguientes características:

- Debe ser resoluble en un período de tiempo razonable con los medios de que disponemos.
- Su resolución ha de ser de interés para la comunidad de estudio en la que nos movemos.

- Del mismo modo es importante que suscite un interés especial en los investigadores encargados de llevar a cabo el trabajo.

a.2) Definición de la hipótesis

Una vez delimitado el problema a abordar se pasa a la siguiente etapa que consiste en la definición de la hipótesis de trabajo. En los métodos tradicionales de investigación científica la hipótesis debe formularse en términos causales (si ocurre A entonces ocurre B). Estas hipótesis son conjeturas de hechos que el método científico deberá validar[8] y contrastar a fin de determinar su veracidad. Un ejemplo de hipótesis así formulada podría ser[9]: "si el ruido afecta al grado de atención de un individuo, un nivel de ruido superior a cierto umbral puede hacer que el individuo baje su rendimiento en la realización de un trabajo". El método científico deberá realizar "experimentos", midiendo niveles de ruido y atención para sacar conclusiones a cerca de la hipótesis planteada.

Sin embargo, es fácil comprobar que la hipótesis propuesta en el capítulo 1 (epígrafe 1.2) no responde a un planteamiento de causa-efecto. El motivo es, de nuevo, que el carácter de la investigación en el ámbito de las ingenierías no se corresponde con el de las ciencias. Si formuláramos la hipótesis en términos causales cambiaríamos también el objeto y el carácter de nuestra investigación. Veamos a que nos referimos:

La hipótesis del trabajo de investigación que nos ocupa, formulada en términos causales, podría ser[10]: "Si la heterogeneidad de modelos existente en las distintas fases de desarrollo de una base de datos y el enfoque funcional de éstos afecta cualitativamente a dicho proceso además de impedir la interoperabilidad entre distintas bases de datos, entonces la definición de un único modelo que soporte los diferentes

[8] En el método científico se habla de verificación de hipótesis. Sin embargo, nosotros hablaremos de validación de hipótesis por homogeneidad con la terminología habitual en la ingeniería del software, utilizando el término verificar para referirnos a la comprobación de la consistencia (de una teoría o, en nuestro caso, de un modelo). En nuestra opinión los términos validar y verificar en ingeniería del software se usan de un modo impreciso. Según la RAE (1992) **verificar** significa "Probar que una cosa que se dudaba es verdadera. Comprobar o examinar la verdad de una cosa", y **validar** significa "Dar fuerza o firmeza a una cosa; hacerla válida". Sin embargo, mantendremos en todo el discurso la terminología de la ingeniería del software a fin de evitar ambigüedades terminológicas.

[9] El ejemplo no pretende ser un ejemplo riguroso de hipótesis científica sino más bien un ejemplo ilustrativo.

niveles de abstracción en el desarrollo de una base de datos, integrando los principales modelos ya existentes y priorizando los aspectos relativos a la estática, mejorará substancialmente la calidad en el proceso de desarrollo de bases de datos facilitando al mismo tiempo la interoperabilidad".

Para poder contrastar la hipótesis así planteada, deberíamos: definir un modelo que cumpliera las características descritas; validar que efectivamente el nuevo modelo se ajusta a los requisitos planteados; implementar completamente dicho modelo; y someterlo a pruebas de contrastación con respecto a otros modelos a fin de validar la hipótesis planteada. Un modelo que cumpla los requisitos exigidos (integración de varios modelos y cubrimiento de todas las fases de desarrollo) es un modelo lo suficientemente grande para que sólo su definición e implementación completa, pueda suponer un trabajo de años a un grupo de varias personas. Además, todavía faltaría la etapa de contrastación la cual constituye, por sí sola un trabajo de investigación (que en este caso, como veremos a continuación, sí será científica).

El trabajo así planteado podría constituir dos tesis: una de investigación en el ámbito de las ingenierías, cuyo objetivo es la creación de un nuevo objeto (en nuestro caso un modelo); la otra constituiría el estudio del funcionamiento de este nuevo objeto. Una vez que la ingeniería ha dado lugar a un nuevo objeto, éste podrá ser susceptible de estudio en sí mismo y este estudio constituirá una labor de investigación científica.

Por tanto, la elaboración de la hipótesis de trabajo no se realizará en términos causales tal y como se hace en investigaciones de carácter científico.

a.3) Definición del método de trabajo

Aunque en ninguno de los métodos estudiados se plantea como una tarea a realizar la definición del método a seguir para la resolución y validación del problema, ver figura[11] 1.2, nosotros consideramos que es necesario ya que, en contra de la opinión de

[10] Tampoco aquí pretendemos plantear un ejemplo riguroso de hipótesis, sino un ejemplo ilustrativo.

Bunge (1976), pensamos que no existe un método universal de resolución de problemas, sino que cada problema requiere su propio método. Existen varios autores cuya opinión difiere de la de Bunge. Así, por ejemplo, Popper (1985) dice: *"Por regla general, empiezo mis clases sobre el Método Científico diciendo a mis alumnos que el método científico no existe"* y en Chalmers (1992) se defiende también la tesis de que no existe un método universal. De hecho, en el caso que nos ocupa, no es posible detallar el método para la fase de definición del nuevo modelo, lo que constituye, precisamente, el núcleo de la resolución del problema; el método consistirá, fundamentalmente, en estudiar los modelos existentes, reflexionar acerca de ellos determinando sus ventajas y limitaciones, y plantear un nuevo modelo que, manteniendo las ventajas de los modelos estudiados, supere, en la medida de lo posible, las limitaciones de los mismos. El llegar a una mejor propuesta final dependerá, en gran medida, de la creatividad y sentido común aplicados a la construcción del nuevo modelo. En Kosso (1993) se habla de la similitud del razonamiento científico y el razonamiento general y se afirma: *"Entre ciencia y sentido común hay sólo algunas sombras de diferencias"*.

Además, pensamos que existe una realimentación entre la fase de resolución y validación, y la de definición del método de trabajo (señalada con trazo discontinuo en la figura 2.1), ya que éste, en nuestra opinión, se va refinando a medida que se avanza en la resolución del problema. Así, podemos decir que la definición del método de trabajo no concluye hasta que se finaliza la fase de resolución y validación.

a.4) Resolución, verificación y validación

Esta fase se detalla en una segunda aproximación del método que se está exponiendo (ver 2.2.2). Aunque, en general, los métodos de investigación científica y, en particular el propuesto por Bunge (1976), sólo señalan la etapa de validación[12], en

[11] La figura completa comprende la resolución y validación del problema; la parte por debajo del trazo discontinuo corresponde a la validación.

[12] Aunque bajo el nombre de *verificación.*

nuestro caso es necesario también realizar una verificación. Podemos construir un modelo que valide la hipótesis planteada pero que no sea consistente, por lo que la verificación del mismo es una tarea necesaria en nuestra investigación.

a.5) Análisis de resultados y elaboración de conclusiones

Se trata de contrastar la hipótesis planteada al comienzo de la investigación con los resultados obtenidos de ésta. Se debe comprobar hasta qué punto se han cumplido los objetivos y en qué medida se ha resuelto el problema. En esta fase es muy importante delimitar los aspectos que no se han podido resolver y otros nuevos problemas que hayan surgido como consecuencia de la investigación y que pasarán a ser puntos de partida de nuevas investigaciones.

a.6) Redacción del informe final

Consiste en la redacción del informe en el que se expone, paso a paso, la investigación realizada. En él se detalla: la hipótesis de trabajo, la justificación del mismo, el método de investigación empleado, el proceso de investigación, las conclusiones obtenidas, la bibliografía consultada y cualquier otro dato que se considere de relevancia para la comprensión y evaluación del trabajo realizado así como para futuros investigadores que deseen continuar el trabajo emprendido[13].

Es conveniente resaltar la importancia de la "ética de la investigación" en esta fase, en la que se deberá reconocer con total sinceridad cuales han sido las aportaciones de nuestra investigación, así como los aspectos que han quedado sin resolver.

Al finalizar la tesis, el cuerpo de conocimiento inicial se ha modificado (en general, se habrá ampliado) y los problemas de los que partimos ya no son los mismos. Algunos se habrán eliminado, otros se habrán modificado (reduciéndose o no) y aparecerán, además, nuevos problemas fruto de la investigación realizada. Este nuevo cuerpo de conocimiento con sus nuevos problemas dará lugar otros trabajos de investigación.

[13] Como sabemos, un trabajo de investigación no se acaba sino que, llegado a un punto, los investigadores deciden finalizarlo. Toda investigación es susceptible de mejoras y ampliaciones.

a.7) Búsqueda de documentación

La búsqueda de documentación aparece, generalmente, como la primera etapa a realizar en toda investigación. Nosotros no estamos totalmente de acuerdo con este planteamiento, ya que pensamos que se debe buscar y analizar documentación durante todo el proceso de investigación. Existen, no obstante, dos fases en las que esta labor debe ser especialmente intensa:

- Durante la determinación del problema.- En este momento deberá hacerse una búsqueda de documentación exhaustiva y dicha bibliografía habrá de ser, principalmente, de carácter general (aunque siempre relacionada con el cuerpo de conocimiento del que se parte).

- Durante la etapa de resolución y validación.- En este etapa se deberá consultar la bibliografía específica del problema concreto que se ha seleccionado. Además, el procedimiento de búsqueda en esta etapa difiere del que se emplea en un primer momento ya que en ella se dispone de referencias de partida y se busca una bibliografía más específica y concreta.

La labor de búsqueda de documentación deberá prolongarse hasta que se finalice el proceso de investigación, a fin de mantenerse al día sobre otros trabajos relacionados que pudieran estarse realizando.

2.2.2. Método de resolución

En una segunda aproximación se detalla el **método a seguir para la resolución validación y verificación** del problema concreto que nos ocupa, para lo cual hemos elaborado nuestro propio método, figura 2.2. Se puede observar que éste se acerca más a cualquiera de los métodos tradicionalmente utilizados en la ingeniería del software (refinamientos sucesivos) que a los métodos de investigación científica (la fase de resolución y validación equivaldría a las fases de creación y realización del experimento en el método experimental). El motivo es, como ya se ha expuesto (apartado a.2 del epígrafe 2.2.1), que el carácter del problema que se desea resolver es también más

cercano a los problemas que se plantean en el ámbito de la ingeniería del software que a los problemas que la investigación científica trata de resolver.

Se presentan juntas las etapas de resolución y validación ya que existe una realimentación entre ellas, de tal modo que el proceso completo de resolución incluye al de validación. La resolución comienza por plantear una primera versión del modelo, la cual se irá refinando según los resultados obtenidos en el proceso de validación. Esta nueva versión refinada volverá a validarse y así sucesivamente. En realidad siempre podrán obtenerse mejoras por lo que el proceso de resolución, como ya se ha comentado anteriormente, no finaliza nunca sino que se decide su finalización. Es tarea del experto (en el caso de una tesis doctoral el experto será su director) decidir cuando puede darse por finalizada la etapa de resolución y para ello se basará, en el caso que nos ocupa, en pautas como el grado de compleción, consistencia y madurez del modelo, en definitiva, en qué momento se puede considerar que se han alcanzado los objetivos propuestos.

Es importante señalar que la investigación en el área de la ingeniería del software requiere, además de validar que la hipótesis se cumple, verificar que el producto construido, en nuestro caso un metamodelo, es consistente. Por tanto, hay que tener en cuenta que la fase de validación incluye, en realidad, verificación y validación.

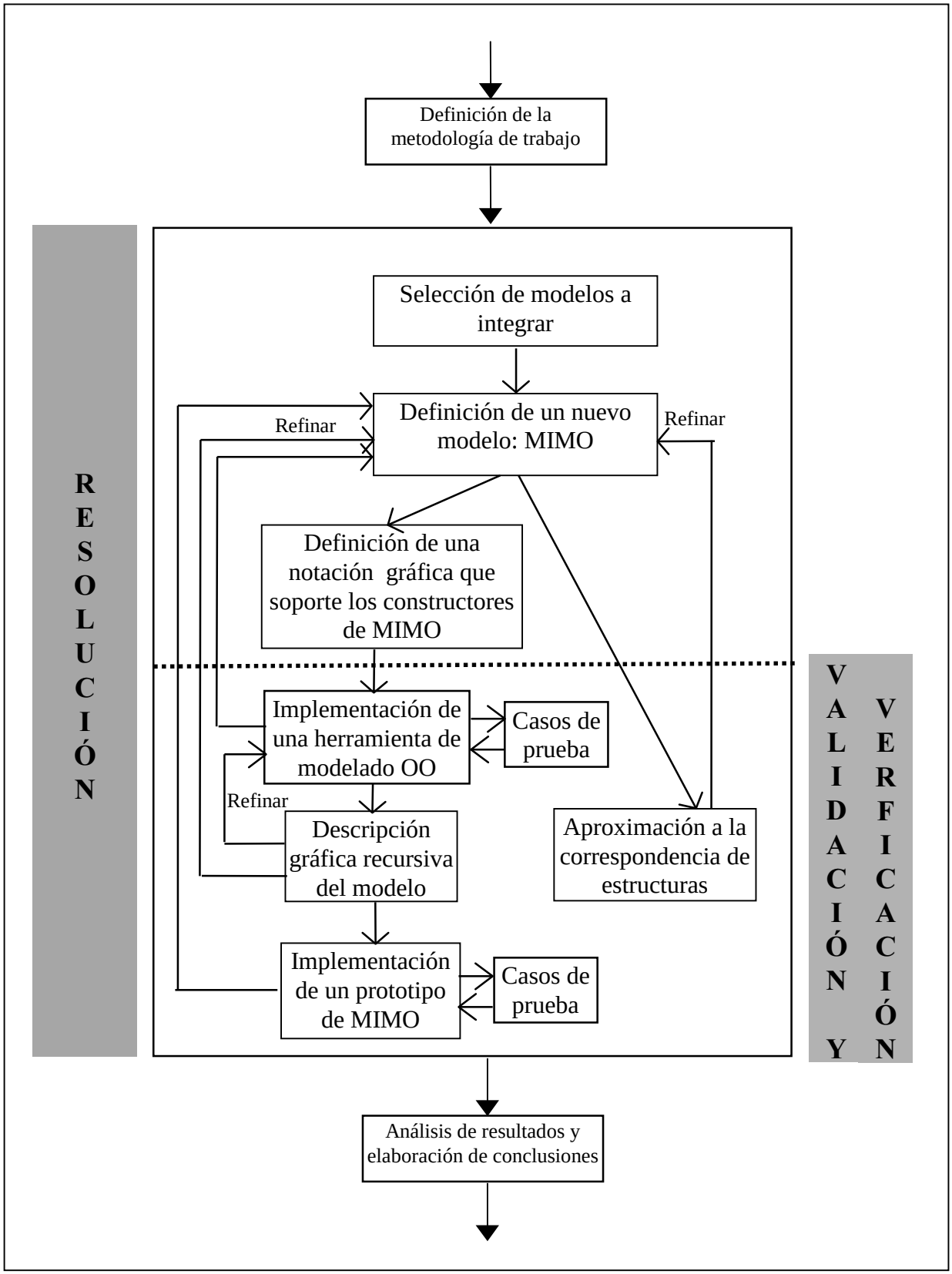


Figura 2.2.- Método de trabajo para la resolución del problema concreto

b.1) Selección de modelos a integrar

En primer lugar deberán estudiarse los modelos más representativos en el ámbito de la orientación al objeto, así como otros que puedan considerarse de interés. Entre ellos se seleccionarán los modelos que habrán de integrarse en MIMO, teniendo en cuenta que se deberán seleccionar modelos de todas las fases de desarrollo.

b.2) Definición de un nuevo modelo: MIMO

Esta es la etapa central de la Tesis. Consiste en la definición de un nuevo modelo que habrá de integrar los modelos seleccionados en la etapa b.1. Se incluirán también aquellos aspectos de otros modelos estudiados, los cuales, a pesar de que no van a ser integrados en su totalidad, se consideran de interés, así como otros aspectos que, aunque no estén recogidos en ninguno de los modelos estudiados, se considere que puedan suponer mejoras en el nuevo modelo.

b.3) Definición de una notación gráfica que soporte los constructores de MIMO

Esta etapa tiene como objetivo la definición de una notación gráfica que permita modelar en MIMO.

b.4) Implementación de un prototipo de una herramienta que soporte el modelado conceptual en MIMO

Dicha herramienta es el módulo de modelado OO de ENEAS/BD (ver 1.3). Esta etapa permitirá validar y verificar parcialmente la consistencia de MIMO como modelo de análisis. Se definirán los casos de prueba más significativos a fin de comprobar su funcionamiento. Uno de los casos de prueba será el propio MIMO. Esta etapa llevará consigo ciertas revisiones de MIMO.

b.5) Descripción recursiva del modelo

Basándonos en la descripción recursiva de los datos propuesta por ANSI (1986), se describirá MIMO en términos de MIMO. Esta etapa permite validar la hipótesis de que MIMO soporta el nivel de análisis y verificar parcialmente su consistencia; esta etapa llevará consigo ciertas revisiones de MIMO.

b.6) Aproximación a la correspondencia de estructuras

Se trata de definir una primera aproximación a la correspondencia entre las estructuras de la fase de análisis-diseño (y diseño-implementación) de MIMO y las de los modelos de análisis-diseño (y diseño-implementación) que MIMO integre. De este modo se valida la hipótesis de que MIMO soporta los niveles de análisis, diseño e implementación, así como la hipótesis de la integración de modelos en MIMO.

Esta etapa se realiza simultáneamente a las fases b.4 y b.5 y también puede dar lugar a modificaciones en el modelo.

b.7) Implementación de un prototipo de MIMO

Se implementará un prototipo de MIMO como modelo de la metabase de ENEAS/BD (ver 1.3). La implementación del modelo completo no es factible en un período de tiempo razonable, ya que, debido a las características de MIMO, éste será de grandes dimensiones. Por ello, se seleccionará para la implementación la parte más original del modelo. Se probará la implementación con los mismos casos de prueba utilizados para la herramienta de modelado conceptual.

La implementación de MIMO permitirá verificar parcialmente su consistencia y dará lugar a cambios que deberán introducirse en las especificaciones de MIMO.

Los refinamientos realizados en el modelo, procedentes de las distintas etapas de validación y verificación, implicarán la necesidad de refinar nuevamente todas las tareas realizadas después de la etapa de definición del modelo (definición de la notación, definición de la correspondencia de estructuras, etc.) y por tanto, de rehacer las tareas de verificación y validación. Se obtiene así un modo de trabajo cíclico en el que el modelo

se irá mejorando en versiones sucesivas. Por ello, la finalización de la Tesis no implicará en modo alguno la finalización del modelo que siempre podrá, y deberá, ser refinado a fin de ampliarlo y mejorarlo.

3 Situación actual del tema

"Se trata no tanto de ver lo que aún nadie ha visto, como de pensar lo que todavía nadie ha pensado sobre aquello que todos ven".

E. Schrödinger

En este capítulo se exponen las características más relevantes de los modelos que, o bien han sido integrados en su totalidad en MIMO, o bien han influido en éste, en algunos casos, de modo un modo muy significativo; se citan también otros modelos estudiados por su importancia y relación con el tema y que han contribuido de algún modo en la realización del presente trabajo (3.2).

Asimismo, y antes de entrar en el estudio de los diferentes modelos, se plantea una discusión acerca del concepto ontológico de modelo de datos (3.1.1). En dicha discusión se reflexiona sobre el concepto de modelo en la ingeniería del software en relación con el concepto de modelo en las ciencias empíricas y formales. Se reflexiona también sobre las ventajas y desventajas de la formalización de modelos de datos, justificando la decisión tomada para nuestro caso concreto (3.1.2). La discusión a cerca de la formalización se plantea, no sólo desde la perspectiva de la ingeniería del software, sino también en comparación con el actual debate sobre la formalización de teorías en filosofía de la ciencia.

3.1. Modelo de datos: concepto y formalización

En el ámbito en que nos movemos es muy común la utilización del término *modelo de datos,* aunque no siempre con la misma acepción. Por ello, queremos hacer algunas reflexiones, que consideramos importantes, acerca de su significado y que, quizá precisamente por lo familiar del término, en muchas ocasiones pasamos por alto. Pensamos que es importante hacer un esfuerzo por clarificar el concepto de modelo en el ámbito de las bases de datos, no sólo por razones puramente formales, sino también porque esta clarificación puede contribuir a esclarecer otras discusiones importantes relativas a los modelos de datos como, por ejemplo, la discusión a cerca de la necesidad de formalización.

3.1.1. Una aproximación ontológica al concepto de modelo de datos

Tsichritzis y Lochovsky (1982) define el concepto **modelo de datos**, distinguiéndolo del de esquema, como *"un mecanismo de abstracción que nos permite ver el bosque (contenido de información de los datos) como oposición a los árboles (valores individuales de datos)"*; un ***esquema*** estaría constituido por *"los nombres de las categorías, junto con sus propiedades, resultantes de la aplicación del modelo de datos a un caso particular"*. Desde entonces se han dado muchas definiciones de modelo de datos, De Marco (1982), Brodie et al. (1984), etc. Sin embargo, es Dittrich quien se acerca más directamente a la discusión que aquí se plantea.

Dittrich (1994) define **modelo de datos** como *"un conjunto de herramientas conceptuales para describir la información en términos de datos. Un modelo de datos comprende aspectos relacionados con:*

- tipos y estructuras de datos

- operaciones, y

- restricciones (consistencia)".

Un modelo de datos permite representar una parcela de información del mundo real de especial interés, lo que habitualmente se denomina *universo del discurso* o, en términos, de Dittrich *mini-mundo*. La representación del universo del discurso se concibe a dos niveles: el de la información en sí misma y el de las estructuras que hacen posible la representación de tal información. Estos dos niveles dan lugar, en el ámbito de las bases de datos, a la distinción entre *esquema* y *base de datos,* conceptos que Dittrich define como sigue: *"La descripción específica de un mini-mundo determinado, en términos de un modelo de datos, recibe el nombre de* ***esquema*** *(esquema de datos o esquema de base de datos) de dicho mini-mundo. La colección de datos que en sí misma representa la información del mini-mundo da lugar a la* ***base de datos****"*. Como el esquema en sí mismo habrá de ser descrito en términos de datos, hablaremos de metadatos. Si es descrito, recursivamente, en términos del mismo modelo de datos que él describe, hablaremos de metaesquema.

Sin embargo, tal y como señala Dittrich, el término *modelo de datos* no está bien escogido para hacer referencia al concepto que representa en el ámbito de las bases de datos ya que, en sus propias palabras, *"... es menos un modelo en sí mismo, que un marco para concebir modelos (del mundo real)"*. Por ello, algunos autores hablan de *formalismos,* en lugar de modelos, como marcos en los que definir esquemas (este es, por ejemplo, el caso de la metodología MERISE, Rochfeld (1992)).

Nosotros coincidimos, en términos generales, con Dittrich, pero pensamos que su observación da lugar a una nueva discusión que planteamos a continuación. De ella podremos concluir que el término *modelo de datos* no está tan mal elegido dentro del ámbito de las bases de datos, siempre que se defina un nuevo concepto de modelo que se ajuste a la ingeniería del software en lugar de importarlo directamente, y sin una mayor reflexión, de las ciencias empíricas y formales. En este sentido planteamos el resto de la discusión.

Un modelo puede definirse, según García Pelayo (1975), como la *"construcción mental a partir de la realidad en la que se reproducen los principales componentes y relaciones del segmento de la realidad analizada"*. Dependiendo de cómo se realice la construcción a partir de la realidad, Aracil (1986b) distingue entre modelos mentales y

modelos formales. La definición de García Pelayo coincide, efectivamente, con el significado de modelo en las ciencias empíricas en las que, a fin de estudiar el comportamiento de una determinada parcela de la realidad, se crea un modelo de ésta. Dicho modelo habrá de ser isomórfico respecto a la realidad que representa y más simple que ésta, destacando sus principales características. Un modelo, según De Marco (1982), representa, a través de abstracciones del detalle, características seleccionadas del sistema empírico en el mundo real que soporta. Utilizando esta acepción de *modelo*, diremos que el *esquema* de una base de datos es un *modelo* del universo del discurso al que representa.

Sin embargo, ésta no es la única acepción del término *modelo*. En Estany (1993) se propone la siguiente distinción:

- Por una parte, el modelo entendido como una **reproducción simplificada de la realidad** (a lo que llamaremos ***modelo-como-copia***). Este es el caso, ya expuesto, de las ciencias empíricas donde se definen modelos de comportamiento simplificados de la parcela del mundo real que es objeto de estudio. Los esquemas, en bases de datos, se comportan de un modo similar: un esquema es un modelo simplificado de la parcela del mundo real que se desea llevar a la base de datos y que habitualmente se denomina *universo del discurso.*

- Por otra parte, el modelo entendido como la **realidad propiamente dicha** (a lo que llamaremos ***modelo-como-original***). Piénsese, por ejemplo, en un pintor, quien reproduce en lienzo a sus *modelos*. En este segundo caso, el modelo no es la representación del mundo real, sino que constituye el mundo real en sí mismo. Es un modelo a copiar o a simular.

El Diccionario de la Lengua Española, RAE (1992), también recoge estas dos acepciones de modelo, proporcionando las siguientes definiciones: *"Arquetipo o punto de referencia a imitar o seguir; en las obras de ingeniero o en acciones morales, ejemplar que por su perfección se debe seguir e imitar"*, para la 2ª de las acepciones; y "*Esquema teórico, generalmente en forma matemática, de un sistema o de una realidad*

compleja (por ejemplo, la evolución económica de un país), que se elabora para facilitar su comprensión y el estudio de su comportamiento", para la 1ª acepción[1].

Manzano (1989) afirma que la acepción de modelo-como-original es la que corresponde a la lógica matemática, donde la representación recibe el nombre de teoría y lo representado el de modelo. Cabría plantearse si éste no es también el caso de los modelos de datos.

Un *modelo de datos* tiene dos finalidades. Por un lado, como indica Dittrich, debe servir como marco para concebir esquemas (que son modelos, entendidos como reproducciones del mundo real). Pero ese mismo marco ha de ser el patrón de funcionamiento de un sistema informático (por ejemplo de un Sistema de Gestión de Bases de Datos -SGBD-); en este caso, el modelo de datos es, en un primer momento, la realidad (ya que el sistema aún no existe) a partir del cuál se implementará el sistema.

Quizá resulte más intuitivo el ejemplo de un arquitecto, para quien una maqueta[2] es un modelo para la construcción de un nuevo edificio. La maqueta sería, en un primer momento, la realidad, puesto que el edificio aún no existe. El arquitecto obtiene un nuevo edificio tomando dicha maqueta como "modelo", lo que le permite obtener una nueva realidad. Del mismo modo, el modelo de datos de un SGBD debe ser definido previamente a su implementación. El SGBD se construirá tomando como "modelo" el modelo de datos previamente definido (análogamente al procedimiento empleado por un arquitecto). Es en este sentido en el que pensamos, a diferencia de Dittrich, que el término modelo de datos sí está bien empleado.

Entendido de este modo, podemos decir que un ***modelo de datos*** es un modelo (***modelo-como-original***) para la construcción de un sistema informático, y a la vez un marco para la definición de nuevos modelos (***modelo-como-copia***), en cuanto que

[1] Puede observarse como el orden de las dos acepciones, en nuestra exposición, no coincide con el de la Real Academia Española; esto es debido a que en nuestra exposición hemos mantenido en primer lugar la acepción más habitual de modelo en las ciencias empíricas, a pesar de coincidir con la primera acepción proporcionada por la RAE.

[2] Es curioso, e ilustrativo para nuestra discusión, que el término español "maqueta", para refrirse al modelo de un arquitecto, no existe en inglés. En inglés una maqueta es un "model" o, en tal caso, un "scale model".

permite la construcción de nuevos modelos (lo que en nuestro contexto se denomina esquema) como representación de parcelas del mundo real.

Podríamos ir todavía más lejos, y ver como un esquema responde a las dos acepciones del término, ya que, al servir de vínculo entre la realidad y el mundo informático, se comporta simultáneamente como una reproducción del primero y un original para el segundo. Del mismo modo, un modelo de datos es un original para el sistema informático y una copia de una realidad: "los datos" (entendiendo los datos como algo abstracto y no como un conjunto de datos concreto).

Coincidimos con Dittrich en que, independientemente de que el término *modelo de datos* sea o no el más apropiado, es demasiado tarde para cambiarlo. Por ello, nosotros mantendremos la terminología habitual entendiendo por **modelo de datos** el formalismo que permite describir un determinado universo del discurso y por **esquema** la representación, en un determinado modelo de datos, de un universo del discurso concreto.

Sin embargo, pensamos que la reflexión planteada es especialmente importante; debemos ser conscientes del doble significado del término *modelo*, ya que en muchas ocasiones se habla de modelos para referirse indistintamente, tanto al modelo del universo del discurso (esquema) como al formalismo en el que éste es definido (modelo de datos). Esta ambigüedad se puede apreciar en expresiones[3] como: "el modelo E/R de una biblioteca" (para referirse al esquema de una biblioteca definido en el modelo E/R) o "el modelo E/R" (para hacer referencia al modelo de datos E/R). Si no evitamos el doble significado del término, o si, al menos, no somos conscientes de ello, esta confusión terminológica puede dar lugar a otro tipo de ambigüedades conceptuales mucho más graves.

[3] Se propone al lector que, en lo sucesivo, cuando lea algún texto relativo a los modelos de datos, modelos de conocimiento, etc., lo haga siendo consciente de esta distinción. El lector podrá comprobar como, en la mayor parte de los casos, aparece la ambigüedad que aquí presentamos oscureciendo y dificultando el entendimiento del texto.

3.1.2. Acerca de la formalización en los modelos de datos

A diferencia de lo ocurrido con el modelo relacional, cuyos principios teóricos fueron anteriores a su implementación, las implementaciones de modelos de objetos, al igual que pasó en los SGBD de la primera generación con los modelos jerárquico y Codasyl, han surgido sin ninguna base teórica en la que apoyarse. Es quizá este hecho el que ha dado lugar a la importancia que actualmente se atribuye a la formalización de modelos en el ámbito de las bases de datos. En Vossen (1995) se presenta una discusión detallada de la necesidad de formalización de modelos de orientación al objeto.

Sin embargo, las últimas tendencias en filosofía de la ciencia parecen ser contrarias a la formalización de teorías, y aunque éste no es exactamente el caso que nos ocupa, sus motivaciones pueden ser igualmente aplicables. En O'neill (1991), se plantea una discusión acerca de la necesidad de aplicación de los lenguajes formales y se presenta el caso de la teoría de los números complejos, como ejemplo de teoría formalizada e inconsistente. Para este autor los beneficios de los lenguajes estrictamente formales[4] no compensan el aumento de complejidad que su aplicación supone. Por ello, en filosofía de la ciencia, existen actualmente dos tendencias: los partidarios de eliminar toda formalización, como por ejemplo los historicistas, y los partidarios de relajarla, entre los que se encuentran los semanticistas y estructuralistas.

Se plantea una otra discusión importante en torno a los modelos que afecta a la realización del presente trabajo, y que se refiere a la necesidad o no de su formalización. Es evidente que el caso que nos ocupa no es equiparable con la elaboración de teorías, ya que nuestro objetivo final es siempre el de la automatización. La automatización de un modelo requiere necesariamente su descripción en términos de un lenguaje (artificial) que ha de ser completo, consistente y sin posibilidad de ambigüedades, lo que no implica, en nuestra opinión, que deba ser necesariamente un lenguaje formal en sentido estricto. Sin embargo, en el ámbito en que nos movemos, parece que el término "formal" siempre hace referencia a la lógica o a las matemáticas. En este mismo sentido, en Luqi y Gogen (1997) se afirma: *"ser formal no requiere necesariamente el uso de*

[4] Entendiendo por lenguaje formal en sentido estricto el lenguaje lógico o matemático.

lógica formal, o incluso de matemáticas. Pero en ciencias de la computación, la frase 'métodos formales' ha adquirido un estrecho significado...".

Es cierto que la formalización puede aportar ciertas ventajas, pero también es cierto que los inconvenientes que de ella se derivan pueden no compensar sus beneficios. En el caso concreto que nos ocupa, y en la misma línea de los semanticistas y estructuralistas en su idea de un cierto relajamiento en la formalización, pensamos que la automatización[5] es una formalización suficiente y en este sentido plantearemos el resto de la discusión.

3.1.2.1. Concepto de modelo/lenguaje formal

El primer inconveniente que encontramos al analizar la utilidad de la formalización de MIMO, es que no existe un criterio comúnmente aceptado sobre lo que es o no "formal".

En Bowen y Hinchey (1995) se define un método formal como *"una técnica basada en las matemáticas para describir sistemas"*. En este mismo artículo se afirma que, *"sin embargo, las definiciones básicas de métodos formales y términos relacionados son confusas".*

En un primer momento, parece que se consideraría un lenguaje formal todo aquel que estuviera basado en las matemáticas (por ejemplo, lenguajes algebraicos) o en la lógica (por ejemplo, lógica de predicados). Sin embargo, en la literatura pueden encontrarse clasificaciones de lenguajes, en formales o no formales, que no atienden exclusivamente a este criterio. A modo de ejemplo puede verse la clasificación propuesta en SQL/MM: MAD-024 (1996) donde se consideran formales lenguajes como EXPRESS (ver 3.2.6), IDL (ver 3.2.1), ODL (ver 3.2.2); se considera un lenguaje no formal, por ejemplo, el propuesto para OMT, Rumbaugh et al. (1991); y se consideran lenguajes con un cierto grado de formalismo, por ejemplo, UML (ver 3.2.4), Open (ver 3.2.8.4), OOram (ver 3.2.8.2).

[5] Automatización significa interpretable por el ordenador de un único modo.

Pero, ¿cuáles son los criterios que llevan a considerar un lenguaje como más o menos formal?. Más concretamente, respecto a la clasificación anteriormente propuesta podemos preguntamos ¿porqué ODL puede ser considerado un lenguaje formal y UML, sin embargo, tiene un grado de formalismo relativo? ¿Cuál es la diferencia entre ambos lenguajes?

En SQL/MM: DBL-024 (1996) se especifica: *"formal significa semántica bien definida, y notación gráfica y sintáctica procesable por el ordenador"*.

Esta definición de lenguaje formal no es tampoco muy precisa. Parece claro que la semántica tiene que estar bien definida, lo que no es tan obvio es determinar como se podría medir el grado de "buena definición" de la semántica. Pensamos también que un lenguaje formal debe tener la capacidad de ser entendido por un ordenador y que si el lenguaje posee dos notaciones, ambas, deben cumplir este requisito. En este sentido podríamos encontrar una diferencia entre el ODL y el UML y es que mientras que el primero se genera mediante una gramática de contexto libre el segundo[6] no.

Sin embargo, un lenguaje descrito mediante una gramática de contexto libre garantiza, tal y como se exige en la definición de modelo formal de SQL/MM: DBL-024 (1996), que la "sintaxis del lenguaje será procesable por el ordenador", pero no garantiza en modo alguno una semántica "bien definida"; para ello, sería necesario formalizar también la semántica del lenguaje. Ni UML ni ODL tienen una semántica formalmente definida y, sin embargo, ODL se considera un lenguaje formal.

Sí existe, en cualquier caso, un claro criterio de formalización: la **automatización**. Aunque la automatización constituye una formalización más relajada que la que puede hacerse en lenguajes basados en las matemáticas, puede ser considerada, en muchas ocasiones, como una formalización suficiente. Esta idea, como veremos a continuación, es defendida por diversos autores, dentro y fuera del ámbito de la ingeniería del software:

En Aracil (1986b) se equipara el concepto de "modelo formal" con el de "modelo programable en un computador" y en Blum (1996) se habla de "modelo informático

formal" como sinónimo de "programa informático" y se afirma que la estructura interna de la base de datos de una aplicación consiste en un modelo[7] formal que se oculta al diseñador. Esta opinión es contraria a la expresada en Vossen (1995), en donde se afirma que *"muchos desarrolladores de sistemas parecen no prestar atención en lo relativo a los modelos formales como un fundamento sólido para sus sistemas, sino que simplemente diseñan un "lenguaje de definición de datos" en el que puedan codificar las características de su modelo de datos"*.

En nuestra opinión, en Vossen se mezclan dos aplicaciones distintas de la formalización que, aunque muy relacionadas entre sí, no son estrictamente iguales: la formalización del modelo en sí mismo y la formalización de los esquemas definidos en dicho modelo. Un lenguaje de definición de datos con una sintaxis formal que esté basado en un cierto modelo no constituye la formalización de éste, sino que permite la formalización de los esquemas que en él se describan. La formalización del modelo propiamente dicho, lleva consigo la descripción de dicho modelo en términos de un lenguaje formal. Si, por el contrario, el lenguaje de definición de datos tiene una sintaxis y también una semántica formal, entonces podremos hablar de modelo formal (ya que la semántica del lenguaje de definición de datos es, precisamente, el modelo[8]).

A diferencia de Vossen, SQL/MM:MAD-024 (1996) considera que ODL es un lenguaje forma; para Vossen se trataría tan sólo de un lenguaje de definición de datos ya que ODL no tiene una semántica formal.

A pesar de todas estas discrepancias, lo que sí parece claro es que la automatización es el único criterio de formalización comúnmente aceptado, ya que ésta lleva asociada la formalización tanto sintáctica como semántica. Sin embargo, la automatización es una formalización más relajada que la de los lenguajes basados en la lógica o en las matemáticas, lo que en algunos casos también se critica.

[6] Al menos, en Booch et al. (1997) no se presenta dicha notación.
[7] Nótese que Blumm habla de *modelo* para referirse a lo que nosotros hemos determinado llamar *esquema* y cómo esta confusión terminológica oscurece el debate sobre la formalización; especialmente, si no somos conscientes del doble significado del término *modelo*.
[8] Es importante recordar que todo lenguaje lleva asociado una sintaxis y una semántica. Un modelo de datos al que se le añade una sintaxis se convierte en un lenguaje cuyo semántica es, precisamente, el modelo.

Pero, formalizar, en el sentido estricto de la palabra (es decir, utilizando lenguajes basados en la lógica o en las matemáticas) también tiene inconvenientes. Por ello, a continuación se analizan los principales inconvenientes[9] y ventajas que la formalización estricta puede proporcionar, estudiando la conveniencia o no de formalizar MIMO. Un estudio más detallado sobre los beneficios e inconvenientes de la formalización puede encontrarse en Bowen y Hinchley (1995) y en Luqui y Goguen (1997).

3.1.2.2. Inconvenientes de la formalización

Podemos hablar de tres grandes inconvenientes de la formalización:

1. **Necesidad de verificación dicha formalización**: formalizar un modelo significa formalizar su semántica y su sintaxis (si es que la tiene). Pero, para ello, es necesario describirlo en términos de un lenguaje formal. Para que este lenguaje sea formal tiene que estar descrito, igualmente, en términos de un lenguaje formal y así sucesivamente. De este modo llegaríamos al primer lenguaje en la cadena, que no puede ser formal. Podemos ilustrar el razonamiento con una vieja paradoja que dice: “quien afeita al hombre que afeita a todos los hombres que no se afeitan a sí mismos”; o lo que es lo mismo, ¿quién demuestra el demostrador de teoremas?

1. **Complejidad**: no cabe duda de que la formalización estricta, debido a la complejidad del aparato matemático, aumenta en gran medida la dificultad, tanto de definición como de comprensión del modelo. A modo de ejemplo se puede citar Gurevich y Huggins (1993), como un intento fallido de dotar al lenguaje C de una semántica formal.

1. **Justificación de la necesidad de formalización**: es necesario evaluar, en cada caso concreto, en qué medida el aumento de complejidad asociado a la formalización estricta es compensado con los beneficios que ésta aporta.

[9] Ya que finalizaremos por concluir que la automatización es una formalización suficiente para el caso que nos ocupa, comenzaremos citando los inconvenientes de la formalización estricta debido a que el apartado de ventajas llevará asociada una mayor discusión.

A continuación se exponen los beneficios de la formalización, analizando en que medida éstos constituyen una justificación suficiente para la formalización de MIMO.

3.1.2.3. Ventajas de la formalización

Existen diversas ventajas derivadas de la formalización estricta:

1. **Independencia de la implementación**: la formalización de un modelo en un lenguaje independiente de la máquina (que, en cualquier caso, no tendría que ser necesariamente un lenguaje matemático) puede suponer, para modelos comerciales, el beneficio de la portabilidad. Sin embargo, en el caso que nos ocupa, lo único que se pretende con la formalización es definir un modelo de un modo no ambiguo y sin inconsistencias, por lo que consideramos que la independencia de la máquina no justificaría la necesidad de formalización estricta de MIMO.

1. **Facilidad para razonar y demostrar propiedades de modelos**: lo que facilitaría la comparación entre modelos o permitiendo demostrar su equivalencia. Este podría ser un beneficio claro de la formalización de MIMO ya que éste tendrá que ser comparado con los modelos que aglutina (SQL3, ODMG-93 y MU). Sin embargo, ninguno de estos tres modelos tiene una semántica formalmente definida y sólo los dos primeros proponen una sintaxis generada mediante una gramática de contexto libre. Por ello, la formalización matemática de MIMO no facilitaría, en modo alguno, la realización de una comparación más rigurosa entre los cuatro modelos.

En KADS-II (1993), se proponen, además, los siguientes argumentos a favor de la formalización de modelos de conocimiento, que son igualmente aplicables a los modelos de datos:

3. La formalización **incrementa la precisión sintáctica y semántica**: sin embargo, la automatización es suficiente para garantizar la precisión, tanto sintáctica como semántica.

3. La formalización es un paso **necesario para la transformación automática de modelos de conocimiento en código ejecutable**: si el objetivo de la formalización es obtener el código ejecutable, no cabe duda de que la automatización proporciona directamente, y sin necesidad de pasos intermedios, dicho código.

3. La formalización **facilita la documentación y comprensión del modelo**: en este sentido, tampoco la formalización estricta es recomendable ya que, como hemos visto, dificulta la documentación y comprensión del modelo con respecto a otras formalizaciones más relajadas.

Por último, citaremos una de las principales ventajas que habitualmente se atribuye a la formalización estricta:

6. **La formalización garantiza la consistencia sintáctica y semántica**, lo que, de ser cierto, podría justificaría la formalización de MIMO a pesar de la complejidad que ello conlleva. Sin embargo, como veremos a continuación, la consistencia de un modelo puede garantizarse sólo en parte.

En Blum (1996) se habla de necesidad de formalización de los modelos[10] de diseño y de la imposibilidad de formalizar modelos conceptuales. El motivo es claro: la formalización no puede garantizar que un determinado modelo conceptual recoge la semántica deseada (es decir, la semántica del universo del discurso que representa).

Atendiendo a la clásica distinción entre validación y verificación, podríamos decir que la formalización puede constituir una verificación parcial de un modelo de datos, pero nunca una validación. La formalización puede verificar la consistencia e integridad del modelo en el sentido de que no existan ambigüedades, que un mismo concepto no se describa contradictoriamente, etc. Sin embargo, su verificación semántica, muy

[10] Aunque la cita original de Blum habla de modelos para referirse a lo que nosotros hemos denominado esquema, en algunas ocasiones resulta confuso, por lo que hemos optado por mantener su propia terminología; se deja al lector interpretar cuando Blum habla de esquemas y cuando lo hace de modelos.
No obstante, el razonamiento que se propone es igualmente aplicable a esquemas que a modelos. En este último caso, el modelo a formalizar (o implementar), MIMO, constituiría el universo del discurso. De él se puede obtener (en el propio MIMO o en otro modelo) el esquema conceptual y el esquema de diseño. Éste último será para Blum el esquema (o modelo) formal.

relacionada con la validación, no es posible garantizarla ya que el significado se lo atribuye el autor del modelo. Es posible describir una teoría o un modelo perfectamente consistente en sí mismo, pero sin ninguna conexión con la realidad. Así, por ejemplo, "los coches tienen perros" es sintácticamente correcto (también lo sería su representación en términos de un modelo de datos, por ejemplo el E/R donde la entidad perro se relacionaría con la entidad coche). La formalización del modelo que soportara esta estructura garantizaría su corrección sintáctica, pero ninguna formalización podría garantizar que semánticamente es consistente. Sólo el usuario podría decidir si tiene algún sentido que los coches posean perros.

Esta es precisamente la idea defendida por Blum (1996) quien afirma que existen dos dominios en el proceso del software: el dominio del problema (modelado conceptual) y el dominio de la implementación (modelado formal). Desde la perspectiva de lo necesario[11], el formalismo de la implementación es irrelevante; lo importante es la experiencia en el dominio de la aplicación. Desde la perspectiva de la implementación, sin embargo, el modelo formal expresa el criterio de aceptación del producto. Blum continúa explicando como un modelo puede ser incorrecto en cuanto a que no recoge las necesidades de la organización y sin embargo correcto respecto a su modelo formal y afirma que el proceso del software siempre comienza con una percepción informal de las necesidades y siempre finaliza con un **modelo formal de computación** (es decir, **el programa**).

Por tanto, podemos concluir que la formalización garantiza la consistencia sintáctica, pero no la semántica, que sólo podrá ser garantizada por el experto en el dominio de la aplicación. En el caso de MIMO, sólo su autor podrá determinar si la semántica que se haya descrito (en un lenguaje más o menos formal) es la que él verdaderamente quería definir. Además, la corrección sintáctica es perfectamente garantizable con una formalización basada únicamente en la automatización.

3.1.2.4. Aproximación escogida

[11] Se refiere aquí Blum a cómo el modelo responde a las necesidades de la organización. Obsérvese que las necesidades de la organización se estudian en la etapa de modelado conceptual.

Nuestra opinión es que, si bien es necesario un cierto grado de formalización en todo modelo de datos, su descripción en un lenguaje constituye una formalización suficiente si dicho lenguaje permite la automatización del modelo. Un lenguaje de tales características aporta los beneficios de sencillez y comprensibilidad de los que carecen, en general, los lenguajes estrictamente formales.

Coincidimos totalmente con Bowen y Hinchey[12] (1995) en que *"claramente (con las debidas disculpas a Eistein), el desarrollo de un sistema debe ser tan formal como sea necesario, pero no más formal"*. Nosotros hemos argumentado los beneficios que la formalización estricta podría aportar a MIMO y los hemos sopesado con los inconvenientes que de tal formalización se derivan y hemos llegado a la conclusión de que la formalización más apropiada para nuestro caso es la automatización.

Por todo ello, no describiremos MIMO en un lenguaje estrictamente formal, sino que su implementación (la cual lleva implícita su descripción en un lenguaje[13] artificial, no ambiguo y sintácticamente formal) constituirá su formalización.

3.2. Análisis de los modelos de objetos existentes

Uno de los primeros objetivos parciales que fijamos para poder llevar a cabo el trabajo que nos ocupa, era el de seleccionar los modelos de objetos que habrían de ser integrados por MIMO. Aunque sabemos a priori que MIMO deberá subsumir los modelos de objetos del SQL3 (3.2.3) y del ODMG-93 (3.2.2) (por ser los estándares de modelos de implementación para bases de objetos), así como el modelo del Método Unificado -MU- (3.2.4) (ya que es la integración de las dos metodologías de objetos más extendidas e importantes en la actualidad), se han estudiado otros muchos modelos. Este estudio nos ha permitido analizar las limitaciones de cada uno de ellos (limitaciones que habrán de ser evitadas en MIMO), así como de sus aportaciones (que deberán ser

[12] Es importante destacar que este artículo constituye una defensa de la formalización. En él, los autores exponen las críticas más comunes a ésta y finalizan concluyendo, al igual que nosotros, que es necesaria cierta formalización, pero nunca más de la necesaria.

[13] La implementación se realizará en POET (ver capítulo 5). El lenguaje de definición de POET es una extensión del C++. C++, al igual que ODL considerado por SQL3/MM: DBL-024 como lenguaje formal, está descrito en una gramática de contexto libre, lo que le dota de una sintaxis con un cierto grado de formalismo.

consideradas en MIMO). A continuación se presenta un resumen de dicho estudio, planteado desde el punto de vista de la estática y sintetizando los aspectos más significativos de cada modelo.

Nos centraremos en la parte estática debido a que MIMO, en su primera versión, es fundamentalmente un modelo estático; de la dinámica recoge tan sólo la posibilidad de definir el comportamiento de los objetos. Aunque ésta es una limitación que nosotros mismos hemos impuesto a esta primera versión, somos conscientes de que un modelo de objetos no puede restringirse al aspecto estático, por lo que la incorporación de la dinámica será la primera extensión del modelo en futuros trabajos.

Es importante señalar que la finalidad de este estudio no es la de presentar una comparación entre los diversos modelos, sino destacar sus principales aportaciones e inconvenientes. Aunque somos conscientes de que dicha comparación facilitaría el estudio de las carencias de modelado actuales, el estudio comparativo de los modelos presentados, además de no ser nuestro objetivo, no es trivial si se desea hacer con rigor. El motivo es que se presentan modelos de diferentes niveles de abstracción (análisis, diseño e implementación) con objetivos diferentes y que, por tanto, soportan conceptos diferentes; además la terminología empleada por ellos no es homogénea por lo que para establecer una comparación sería necesario empezar por unificar la terminología.

Aunque en la literatura se pueden encontrar diversas clasificaciones de modelos, en general, cada una de ellas compara modelos de un determinado nivel de abstracción. Así por ejemplo, en Brodie (1994) se presenta una comparación de los modelos clásicos (jerárquico, red y relacional) y de los modelos semánticos (E/R, TAXIS, RM/T, etc.), pero ambas comparaciones se realizan por separado y en base a diferentes criterios.

Por todo ello, la única comparación que se establecerá será entre MIMO y los modelos que subsume (nivel de análisis de MIMO con el MU y nivel de construcción de MIMO con ODMG-93 y con SQL3). Dichas comparaciones se realizarán en los capítulos 4 y 5, una vez que se haya definido una terminología común.

Comenzamos por aquellos modelos que han tenido una influencia más directa en MIMO: el modelo de objetos del OMG, ODMG/93, SQL3, MU, MERISE, EXPRESS y

MEDEA, para finalizar (3.8) resumiendo los aspectos principales de otros modelos de interés.

3.2.1. Modelo de objetos del OMG

OMG (Object Management Group) es un consorcio formado en 1989 por HP, Sun y una docena más de compañías importantes del mercado de la informática. Actualmente cuenta con más de 600 miembros, entre los cuales se encuentran casi todos los fabricantes de software y de hardware, así como usuarios finales. Su principal objetivo es estandarizar y promover la tecnología de objetos en todos sus campos, permitiendo la interoperabilidad en entornos distribuidos y heterogéneos.

OMG define una arquitectura (Object Management Architecture, OMA) para entornos distribuidos que se especifica en Mark y Stone (1995). OMA se compone de:

- **Agente de petición de objetos.-** Conocido por sus siglas inglesas ORB (Object Request Broker), su misión es la de garantizar la portabilidad e interoperabilidad de objetos a través de una plataforma de sistemas distribuidos. La arquitectura del ORB se recoge en lo que se denomina CORBA (Common Object Request Broker Architecture) en OMG (1995a).

- **Servicios de objeto (OS, -Object Services-)**.- Proporciona las operaciones básicas para el modelado lógico y para el almacenamiento físico de los objetos. Entre otros servicios, se encarga del manejo de clases, la creación de objetos, etc. OMG (1995b).

- **Facilidades comunes**.- Proporciona un conjunto de funcionalidades genéricas que pueden ser utilizadas en distintas aplicaciones (facilidades de bases de datos, facilidades de correo electrónico, etc.).

- **Objetos de aplicación**.- Son aplicaciones útiles para distintos usuarios desarrolladas a partir de clases de objetos. De estas clases, algunas han sido construidas específicamente para la aplicación y otras a partir del conjunto de facilidades comunes.

Además de los componentes citados, OMA define un modelo de objetos básico (OMG/OM) denominado *Core Object Model*. Dicho modelo es el núcleo central de la arquitectura de CORBA y debe ser satisfecho por los distintos fabricantes para garantizar la portabilidad e interoperabilidad de sus productos.

"Core Object Model"

Definido por el OMTF (Object Model Task Force[14]) constituye el núcleo central del OMG ORB y del OMG OS. Así mismo ODMG también amplía este modelo para obtener el ODMG-93 como un superconjunto del OMG/OM. Su primera definición se realizó en 1992 y su última revisión en 1995.

OMG/OM fue diseñado con dos objetivos: el primero, asegurar la portabilidad de aplicaciones y de bibliotecas de tipos; y el segundo, garantizar la interoperabilidad de componentes software en un entorno distribuido y heterogéneo.

El modelo define únicamente los conceptos de objeto, no-objeto, operación, interfaz, capacidad de sustitución, tipo y herencia, que se exponen a continuación.

- OMG/OM distingue entre **objetos** y **no-objetos**. Ambos se denominan objetos denotables. Los objetos tienen un identificador único, inmutable, mientras existe el objeto y que es independiente de las propiedades o el comportamiento de éste. En OMG/OM no se especifica un conjunto de tipos no-objetos ya que éste dependerá de cada implementación concreta.

- El comportamiento de los objetos se define mediante **operaciones**. El modelo no permite definir su implementación, tan sólo permite especificar su signatura. Es un modelo de objetos clásico en el que cada operación se asocia a un sólo tipo de objeto. Todas las operaciones de un mismo tipo de objeto han de tener nombres diferentes. Los tipos no-objetos no tienen operaciones.

[14] OMG se estructura en comités técnicos (Technical Committee, TC), grupos de trabajo (Task Force, TF) y grupos de interés especial (Special Interest Group, SIG). Los TF están compuestos por miembros del TC e invitados especializados. Su cometido es el de resolver problemas técnicos específicos de una determinada área y elaborar propuestas y soluciones para los TC. Los TC se encargan de elaborar propuestas de estándar y sus posibles extensiones. Las decisiones finales las toma el BOD (OMG Board of Directors).

- Se denomina **interfaz** a una colección de signaturas de operaciones. Entre los interfaces se establece un tipo especial de relación que depende de las posibilidades de sustitución entre ellos: se dice que un interfaz A es sustituible por un interfaz B si un cliente que espera usar un objeto con interfaz B puede, en su lugar utilizar una referencia a un objeto con interfaz A. Sin embargo, el que dos objetos tengan interfaces sustituibles no implica que uno de ellos se pueda usar en lugar del otro.

- La posibilidad de sustitución de un interfaz es una condición necesaria pero no suficiente para determinar si un objeto puede ser usado en lugar de otro. El resto de los factores que permiten decidir si dos objetos son sustituibles vienen determinados por el **tipo**. Un tipo consta de un interfaz y de sus interrelaciones con uno o más tipos. OMG/OM sólo soporta interrelaciones de herencia. Cuando un objeto del tipo A es sustituible por un objeto del tipo B se dice que A es un **supertipo** de B y B un subtipo de A. En OMG/OM todo tipo nuevo se especifica por **herencia** de tipos existentes, de tal modo que el nuevo tipo creado es un subtipo del tipo del que hereda. Soporta herencia simple y múltiple.

- Todo objeto es un ejemplar directo de un sólo tipo, que se especificará en el momento de la creación del objeto. El conjunto de todos los ejemplares de un mismo tipo recibe el nombre de **extensión**.

El OMG/OM ha tenido un impacto importante dentro del paradigma de la orientación al objeto, ya que ha sido el primer estándar de modelos de objetos. El problema es que se trata de un modelo muy básico (se pensó como un núcleo común a partir del cual se pudieran derivar otros modelos) por lo que, por sí mismo, no puede ser considerado un modelo para bases de objetos. Sin embargo, quizá podría verse como tal, si se consideran el modelo de objetos y los servicios de objetos (OS) en conjunto ya que, según se afirma en Wells y Thompson (1994), las funcionalidades proporcionadas por el conjunto de los servicios de objetos (OS) superan a las proporcionadas por SGBD (tanto de objetos como relacionales). El Core object model, como ya hemos dicho, ha sido

ampliado por CORBA (para soporte de objetos distribuidos) y por ODMG-93 (para soporte de bases de objetos).

3.2.2. ODMG-93: estándar para bases de objetos

ODMG-93 es un estándar para sistemas de gestión de bases de objetos (SGBO), definido por un destacado grupo de fabricantes de dichos sistemas: el grupo ODMG (Object Database Management Group). ODMG se formó gracias a la iniciativa de Rick Catell en el verano de 1991. Catell, debido al lento progreso de los estándares en el ámbito de los SGBO, convocó a los fabricantes de SGBO para formar un consorcio que trabajara en la definición de un estándar de SGBO. Dicho estándar fue aprobado en 1993. Actualmente se han realizado ya algunas revisiones a la versión 1.0 en lo que se conoce como ODMG-93 versión 1.1 en Cattell (1994a). Existe ya una versión 1.2, Cattel (1995), cuyas actualizaciones se centran en el lenguaje de consulta (OQL).

En un primer momento eran cinco los componentes del grupo (Tom Atwood -Object Design-, Joshua Duhl -Ontos-, Guy Ferran -O2 Technology-, Mary Loomis -Versant- y Drew Wade -Objetivity-) además de Rick Catell -SunSoft-. Ellos fueron los autores de la versión 1.0. Posteriormente el grupo se ha ido ampliando contando, en febrero del 94, con más de 20 integrantes.

Los objetivos principales del ODMG-93 son: facilitar la portabilidad a nivel de aplicaciones y facilitar la interoperabilidad entre sistemas, incluso entre bases de datos distribuidas y heterogéneas.

Los componentes principales de la **arquitectura** ODMG-93 son los siguientes:

- **Modelo de objetos**.- El modelo de objetos de ODMG-93 (ODMG/OM) (que se expone posteriormente) está basado en el OMG/OM (ver 3.2.1). El núcleo del modelo de OMG fue diseñado para ser un denominador común entre el ORB (para distribución de objetos a través de una red), sistemas de bases de objetos, lenguajes de programación de objetos y otras aplicaciones. ODMG/OM amplía OMG/OM con capacidades de bases de datos (tales como atributos, interrelaciones, etc.).

- **Lenguaje de definición de objetos (ODL)**.- La sintaxis del lenguaje de definición de objetos está basado en el lenguaje de definición de interfaz (IDL) de CORBA.

- **Lenguaje de consulta de objetos (OQL)**.- Es un lenguaje declarativo. Su sintaxis se basa en la del SQL3 (concretamente en el núcleo del SQL-92) pero su semántica no coincide debido, en gran parte, a las limitaciones que el modelo de datos del SQL3 impone por su necesidad de compatibilidad con el modelo relacional. Sin embargo, estos dos lenguajes tienen actualmente una tendencia a la aproximación. Ya en 1994 el propio Cattell (1994a) hablaba de la convergencia, afirmando: *"esperamos que OQL y SQL converjan en un futuro próximo"*. En la actualidad, dicha convergencia es casi una realidad para el SQL2 (OQL V1.2 mantiene un grado de compatibilidad con el SQL2 de, aproximadamente, un 90%, Wade (1996)), y está muy cerca de serlo para el SQL3 (ver epígrafe 3.2.3).

- **Vinculación con los lenguajes C++ y Smalltalk.-** Es posible realizar funciones de definición y manipulación de objetos por medio de estos dos lenguajes. ODMG-93 define un ODL, y un OQL; no define OML (lenguaje de manipulación de objetos), por lo que sus funciones se realizarán a través de los vínculos con C++ y Salltalk.

Modelo de objetos (ODMG/OM)

Se encuadra dentro de los llamados modelos “revolucionarios” o “puristas” de la orientación al objeto, ya que rompe por completo con las anteriores tendencias relacionales. Es un modelo que nace en el ámbito de los SGBO (Sistemas de Gestión de Bases de Objetos) con el objetivo de integrar un modelo de objetos con los lenguajes de programación OO existentes. En contraposición, surgen los modelos conocidos como “evolucionistas” o “no puristas” (ver 3.2.3) a los que se llega por la necesidad de ampliar los sistemas de bases de datos relacionales con capacidades de objetos. El ODMG/OM está marcado substancialmente por esta aproximación a los lenguajes de programación orientados al objeto, de tal modo que, en ODMG-93, la base de datos y el

lenguaje de programación de aplicaciones se integran de un modo totalmente transparente. Esta transparencia elimina la necesidad de realizar una correspondencia explícita entre la representación de los datos en la base y la representación de los mismos en el lenguaje de programación.

El elemento básico de ODMG-93 es el objeto. Éstos se clasifican en tipos, de modo que todos los objetos de un mismo tipo tienen propiedades y comportamiento común. El comportamiento se define por el conjunto de operaciones que pueden ser realizadas sobre cualquier objeto del tipo. Los valores que toma el conjunto de propiedades de un objeto definen el estado de éste. Las propiedades de un objeto son sus atributos y sus interrelaciones con otros objetos. Tanto las operaciones como las propiedades de un objeto se denominan características.

En ODMG/OM se define una jerarquía de tipos, en la que se distinguen Objetos_denotables y características:

- **Objeto_denotable**
  - Objeto
  - Literal
- **Característica**
  - Operación
  - Propiedad
    - Atributo
    - Interrelación

La jerarquía de objetos tiene como raíz el tipo "objeto_denotable" y tiene dos posibles líneas de descomposición ortogonales entre sí: objetos mutables/inmutables y objetos atómicos/estructurados, que se clasifican del siguiente modo:

- Objeto_denotable
  - Objeto
    - **Objeto_atómico**
    - **Objeto_estructurado**
  - Literal
    - **Literal_atómico**
    - **Literal_estructurado**

donde los objetos son objetos_denotables mutables y los literales son objetos_denotables inmutables.

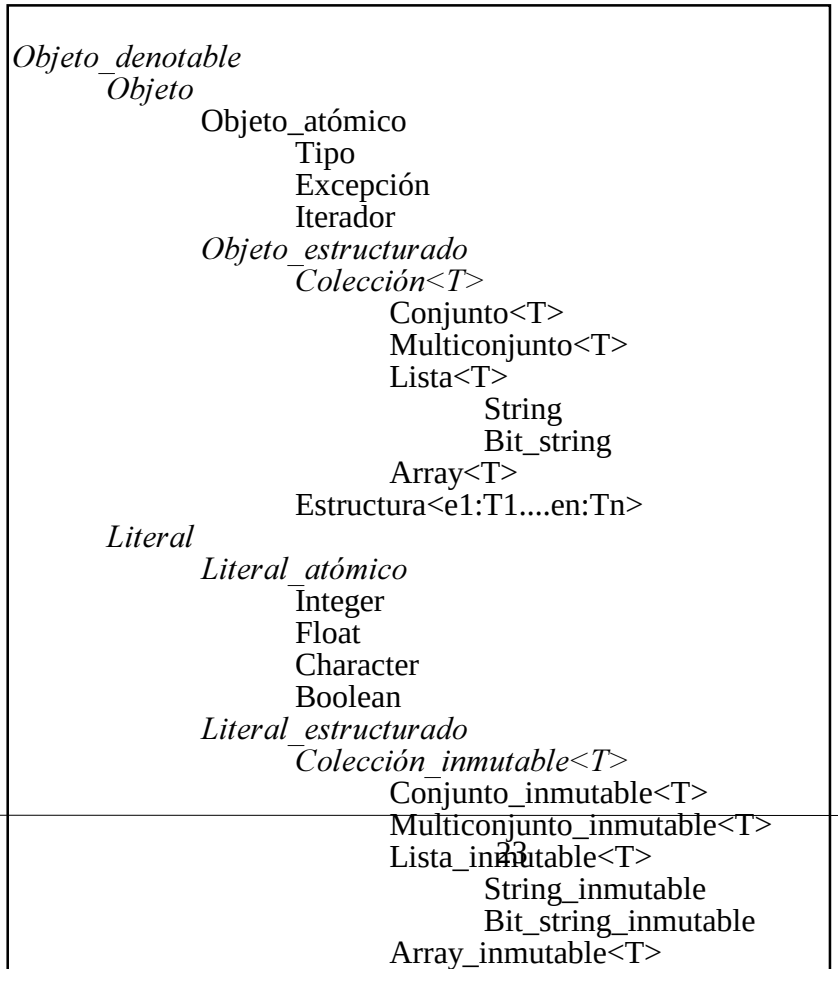
Todo objeto denotable tiene **identidad propia**. La diferencia está en la representación utilizada para la identificación en objetos y literales:

- **Objetos**.- la representación de su identidad es una combinación de bits generada únicamente con el fin de identificar un objeto. A ésta combinación de bits se le da el nombre de identificador de objeto (OID).

- **Literales**.- la representación de su identidad es la codificación en bits de su valor.

Los objetos son a veces llamados objetos_mutables ya que, gracias al OID, el valor de sus propiedades puede variar sin alterarse la esencia del objeto. Sin embargo, los literales son objetos_inmutables ya que, al identificarse por su valor, si éste cambia el objeto cambia también.

La figura 3.1 muestra la jerarquía completa de tipos del ODMG/OM:



Los tipos en cursiva son tipos abstractos no ejemplificables.

Un tipo tiene una interfaz y una o varias implementaciones. La **interfaz** es la parte visible y define las propiedades y operaciones que pueden ser invocadas por los objetos del tipo. La **implementación** especifica las estructuras de datos y métodos que soportan, respectivamente, las propiedades y operaciones descritas en el interfaz. Se define **clase** como la combinación de la especificación del interfaz de un tipo y una o más implementaciones definidas para dicho tipo.

Los tipos son objetos en sí mismos por lo que también pueden tener propiedades. En la interfaz se definen tanto las propiedades de los objetos como las **propiedades del tipo**, que pueden ser:

- ***Supertipos***.- Permiten definir jerarquías de supertipos/subtipos en las que un subtipo hereda todas las características de sus supertipos. El subtipo puede definir características propias y redefinir las heredadas.
- ***Extensión***.- Se denomina extensión al conjunto de todos los ejemplares de un tipo.

- ***Claves***.- Una clave es una propiedad (clave simple) o conjunto de propiedades (clave compuesta) que permite identificar unívocamente los ejemplares de un tipo.

Las **propiedades de los objetos** son:

- ***Operaciones***.- En la interfaz del tipo se especifica únicamente su signatura. La implementación se define en C++ o Smalltalk. Toda operación se define sobre un único tipo y no se soporta el concepto de operación libre. El nombre de una operación debe ser único para cada tipo de objeto, sin embargo, sí se soporta polimorfismo de operaciones entre diferentes tipos de objeto.

- ***Atributos***.- Se especifican en el interfaz mediante un nombre y un tipo de datos. Los atributos toman literales como valores.

- ***Interrelaciones***.- Se especifican mediante una signatura de interrelación en la interfaz. La signatura define el tipo del otro objeto involucrado en la interrelación, así como el nombre de los caminos transversales que permiten acceder al objeto u objetos relacionados. La interrelación en sí misma no tiene nombre. Todas las interrelaciones son binarias y se establecen entre objetos.

De las numerosas críticas que el ODMG-93 ha sufrido, véase a modo de ejemplo Kim (1994), quizá la más comúnmente aceptada es la de no ser compatible con el SQL. En cualquier caso, como ya se ha dicho en varias ocasiones a lo largo de este trabajo, este problema está ya en vías de solución. Además, como señala Kim, el ODMG-93 no contempla aspectos característicos y de gran importancia en una base de datos como son el soporte de restricciones y disparadores, soporte de vistas, etc. En la primavera de 1995 Melton (editor del SQL3) se reunió con varios miembros del ODMG. En dicha reunión, que constituyó un paso importante para la creación del SQL/ODMG Merger Group, se acordaron ciertos puntos que debería modificar cada estándar a fin de acercarse mutuamente, ver DBL: YOW-031. El ODMG se comprometió a incorporar, entre otras funcionalidades, el soporte de vistas, el soporte de restricciones, los valores nulos y el tipo de dato tabla; estas dos últimas funcionalidades ya se han incorporado a la versión 1.2 del ODMG-93, Catell (1995), aunque según afirma en Wade (1996) sólo

se soportan en OQL (no en ODL, lo que significa que es posible obtener este tipo de datos como resultado de una consulta, pero que no se proporciona sintaxis para su definición). Esto es, en nuestra opinión, poco recomendable ya que el ODL y el OQL deberían tener un mismo modelo de objetos subyacente.

Entre las principales ventajas del estándar cabe citar dos de especial importancia: permite la portabilidad a nivel de aplicaciones, Bancilhon y Ferran (1994) y proporciona facilidades para la definición y manipulación de estructuras de datos complejas (colecciones y estructuras), Kim (1994).

En nuestra opinión, ODMG-93 tiene un modelo con una semántica muy limitada; en parte por ser un modelo de implementación y en parte por las carencias ya mencionadas. Sin embargo, coincidimos con Kim en que su soporte de tipos complejos es potente, mucho más que el del SQL3 (ver 3.2.3), y éste es un aspecto importante para una base de objetos. También es cierto que permite un alto grado de portabilidad. Además, pensamos que el ODMG-93 ha supuesto una contribución importante en el paradigma de la orientación al objeto, ya que ha proporcionado a los sistemas comerciales un estándar en el que apoyarse; se puede decir que el ODMG-93 ha llegado, aunque no completo, “a tiempo”. Esta ha sido quizá, como veremos a continuación, la principal traba con la que se ha encontrado el SQL3.

3.2.3. SQL3: futuro estándar para SGBDR extendidos

El lenguaje SQL (Structured Query Language) nace en el grupo del IBM Research Laboratory en San José de California encabezado por Chamberlin. El primer prototipo, SEQUEL-XRM, se implementó entre 1974 y 1975. Posteriormente y en los mismos laboratorios se implementó otro prototipo, el llamado System R, cuyo éxito provocó la aparición en 1979 del primer sistema comercial basado en SQL (ORACLE).

A partir de aquí comienzan a aparecer nuevos productos relacionales tanto de IBM como de otras casas comerciales y surge la necesidad de estandarización. Dos

organismos se encargaron de ello: por una parte el Instituto de Estándares Nacional Americano (American National Standars Institute, ANSI) y por otra, la Organización Internacional para la Normalización (International Standarization Organization, ISO). Estos dos organismos, aunque trabajan de modo casi paralelo en el desarrollo del lenguaje, no siempre coinciden en sus versiones debido a las diferencias temporales de sus reuniones. De este modo, el primer estándar de SQL es aprobado en 1986 por ANSI y en 1987 por ISO. En 1989, ISO, en el documento *Database Language SQL with Integrity Enhancement*, conocido como Addendum, define un conjunto de ampliaciones al estándar orientadas a mejorar la integridad de los datos. En este mismo año se publica el ISO/ANSI *(working draft) Database Language SQL,* documento elaborado por los grupos de trabajo de ANSI X3H2 y el ISO/IEC JTC1/SC21/WG3[15], y que define una versión bastante ampliada del estándar original para SQL. Esta versión (conocida como SQL2), revisada y mejorada, fue propuesta como candidata para una posible estandarización, y su aceptación, en otoño del 92, ha hecho del SQL2 el lenguaje estándar actual para bases de datos relacionales. El SQL2 mejora substancialmente el tratamiento de la integridad referencial, además de incluir nuevas capacidades como son: tratamiento de tablas temporales, definición de aserciones, definición de dominios (aunque la semántica de éstos no se recoge en el SQL2, si se contempla su definición sintáctica), incorporación de nuevos tipos de datos (v.g. el tipo STRING), lenguaje de manipulación del esquema, combinación externa, SQL dinámico, etc.

Desde que aparece el concepto de orientación al objeto, ha estado presente la idea de dotar al SQL con las capacidades propias de este paradigma. Se trataba, por un lado, de hacer frente a la necesidad de que los objetos fueran persistentes y, por otra parte, de permitir a las aplicaciones relacionales utilizar dichas capacidades. La primera aproximación al paradigma tiene lugar en 1988 cuando T. Murata (Japón) propone la idea de añadir el concepto de subtabla y supertabla. En este mismo año, John Kerridge (The University, Sheffield, United Kingdom), propone incluir en lo que ya de un modo informal se llamaba SQL3, los Tipos Definidos por el Usuario ("User Defined Types", -

[15] ISO está formado por distintos organismos de estandarización nacionales (ANSI en EEUU, AENOR en España, etc.). Se divide en subcomités (SC) y estos a su vez en grupos de trabajo (WG). El WG3 se encarga de la estandarización en bases de datos y el DBL, dentro del WG3, de la estandarización de

UDTs-), que son estructuras de registros que pueden utilizarse como el tipo de datos de una columna en una tabla. En 1991, un grupo de investigadores de Digital Equipment Corporation proponen revisar la noción de UDTs añadiendo muchos de los conceptos de la orientación al objeto, entre otros el concepto de identificador único. Más adelante, David Beech, de Oracle Corporation, propone nuevas extensiones al SQL3 pero, sin duda, su mayor aportación fue la unificación de los modelos relacional y orientado al objeto. A partir de aquí y debido al reconocimiento de la importancia de los esfuerzos que se estaban realizando, otras casas comerciales se unieron para la realización de dicho trabajo, enviando a sus expertos en el tema de objetos (cabe citar a Phil Shaw, primero como representante de IBM y posteriormente de Oracle, Nelson Mattos de IBM, Amelia Carlson primero por HP y actualmente por Sybase, Krishna Kulkarni por Tandem y Jim Melton primero por Digital y actualmente de Sybase). Andrew Eisenberg, de Digital, inicia las extensiones al lenguaje procedimental necesarias para soportar el comportamiento de los objetos. Las primeras extensiones de objetos fueron recogidas inicialmente en un documento conocido como "MOOSE" (Major Object-Oriented SQL Extensiones), en DBL: ARL-078 (1991), aunque según el propio Melton (1994), en un primer momento éstas fueron totalmente ignoradas por los vendedores de SGBD orientados al objeto.

En Julio de 1992 se publica el documento titulado (ISO/ANSI) Working Draft Database Lenguage SQL (SQL3), DBL: CBR-003 (1992), en el cual se recogen las ampliaciones anteriormente mencionadas con el fin de dar soporte a bases de datos orientadas al objeto manteniendo la compatibilidad con el SQL2 puramente relacional. Desde entonces los comités de estandarización ISO/ANSI han seguido trabajando en esta línea, a la vez que se añadían al estándar nuevas capacidades como extensiones a bases de datos temporales o acceso remoto a datos. El borrador del SQL3 ha llegado a ser tan amplio que en la reunión del grupo de trabajo ISO/IEC JTC1/SC21 WG3 DBL celebrada en Munich en 1994, se decidió dividir el documento en seis partes con el fin de aprobar, si no el estándar entero, partes de éste en menores plazos de tiempo, así como de facilitar a los fabricantes la instrumentación de partes aisladas. El núcleo del

lenguajes de bases de datos. Información detallada acerca de la estructura y funcionamiento de ISO puede encontrarse en De Miguel y Piattini (1992).

modelo de datos reside en la parte 2 denominada “Foundation” y todas las extensiones (temporal, módulos persistentes, etc.) se realizan a partir de ella. En la actualidad, y tras las reuniones del grupo que tuvieron lugar en Londres, DBL: MCI-001 (1996) y en Kansas City, DBL: MAD-001 (1996), el SQL3 se compone de nueve partes, entre ellas se encuentra, como veremos en los siguientes apartados, la que define el modelo de objetos. Aunque en un primer momento se hablaba, como posible fecha de estandarización, del año 95, aún se siguen discutiendo aspectos importantes del modelo y se prevé que no sea norma internacional hasta el 98/99. Quizá la principal ventaja del nuevo modelo del SQL3, asegurar la convivencia del mundo relacional y el orientado al objeto, ha sido a la vez el principal inconveniente a la hora de agilizar su proceso de estandarización ya que, como veremos a continuación, han sido muchos los esfuerzos realizados hasta logra un modelo de objetos estable y aceptado por todos los países que participan en el proceso de estandarización.

Evolución del modelo de objetos del SQL3

Desde que se comenzó a llevar a cabo la idea de dotar al SQL con extensiones propias del paradigma de la orientación al objeto (aproximadamente en 1991) se han planteado distintas posibilidades en cuanto al modelo de objetos a establecer. Se trataba de conseguir integrar el modelo relacional con un modelo orientado al objeto. A partir de este momento, se han planteado distintas soluciones a dicho problema, pero ninguna de ellas ha sido totalmente satisfactoria para el grupo DBL de ISO (encargado de la estandarización del SQL3). Los distintos modelos propuestos desde el nacimiento del borrador del estándar se han basado en las siguientes ideas:

1. En un primer momento, se dotó al lenguaje de la posibilidad de definir tipos abstractos de datos (Abstract Data Type, ADT). Cada ADT se implementaría en una tabla (la tabla corresponde a la noción de clase). Un objeto sería pues un ejemplar de una tabla. Un ADT podía ser opcionalmente definido con o sin OID; de este modo se soportarían los conceptos de valor y objeto. Por otro lado, se mantenía el concepto tradicional de tabla. El modelo se recoge en DBL: CBR-003 (1992).

2. Posteriormente se decidió flexibilizar la sintaxis y permitir la definición de dos especies distintas de ADTs: los ADT valor y los ADT objeto. En esta aproximación, la extensión de un objeto sería siempre una columna de una tabla, en contraste con la primera aproximación en la que los objetos vivían en filas de tablas. El modelo se recoge en DBL: YOW-004 (1995).

3. De aquí se pasó a considerar la posibilidad de definir un tipo especial de tablas, llamadas tablas extensión, y que constituirían la única “vivienda” posible para los objetos. Una tabla extensión es una tabla con una única columna, en la que se ubicaría el objeto. Nunca podría encontrarse, por tanto, un objeto en una columna de cualquier otra tabla que no fuera una extensión; en su caso, habría una referencia a un objeto. Un mismo tipo de objeto podría tener distintas tablas extensión, DBL: LHR-004 (1995).

Sin embargo, y dado que ninguna de estas soluciones era óptima, ya que todas ellas dejaban problemas importantes por resolver, se siguieron ensayando nuevas alternativas. Una de ellas, propuesta por los expertos de UK en DBL: LHR-029 (1995), DBL: LHR-031 (1995) y DBL: LHR-032 (1995), estaba basada en la idea de mantener unos sitios específicos donde guardar los objetos. Estos sitios no eran sino columnas de tablas. Cada uno de ellos tenía un localizador que permitiría diferenciar unos objetos de otros.

Esta propuesta, junto con la de las tablas extensión, iban a ser discutidas en la reunión del comité celebrada en Enero de 1996 en la ciudad de Londres. Sin embargo, una vez allí, ambas fueron desestimadas debido a nueva propuesta presentada por ANSI, DBL: LHR-077 (1995). En dicho documento se proponía un nuevo modelo de objetos, que pasaremos a exponer a continuación, así como incluir en el SQL3 una nueva parte (parte 8) llamada "Extended Object”. De este modo se eliminaría parte del modelo de objetos de la parte 2 ("Foundation"), donde se encontraba hasta dicho momento, dejando en ésta lo relativo a valores ADT.

La aprobación de esta propuesta podría suponer una agilización en el proceso de estandarización de "Foundation" y una demora en el correspondiente proceso para el

modelo de objetos. En la reunión del comité celebrada en Kansas City, DBL: MAD-001 (1996), se aprobó definitivamente la propuesta, pero con el compromiso de trabajar en ella paralelamente a Foundation, de tal modo que ambas partes puedan ser aprobadas simultáneamente.

Así pues, podemos decir que el SQL3, después de un periodo aproximado de seis años, tiene por fin una propuesta de modelo de objetos consistente y aceptada por los expertos de los distintos países que contribuyen a su definición. Las bases de partida para dicho modelo se recogen en DBL: MAD-010 (1996). En la actualidad, y después de la última reunión del comité que tuvo lugar en Madrid (enero-febrero de 1997), las partes 2 (Foundation) y 8 (Extended Object) han alcanzado el estado de borrador de comité (committee draft, -CD-), por lo que se puede considerar que el modelo de objetos del SQL3 es ya un modelo estable.

A continuación se hará un breve resumen de su estado actual, si bien queremos subrayar que todo cuanto aquí se diga esta aún sujeto a cambios.

Modelo de objetos (SQL3/OM)

El modelo de objetos[16] del SQL3 es un modelo evolucionista, ya que se obtiene como una extensión al modelo relacional del SQL 92. Esto presenta una gran ventaja ya que facilita la convivencia de dos generaciones de bases de datos, permitiendo el mantenimiento de los sistemas ya existentes, así como la creación de otros que se adapten a las nuevas tendencias. Sin embargo, la necesidad de preservar la compatibilidad respecto a las versiones anteriores de SQL restringe, en gran medida, la libertad a la hora de incluir el modelo de objetos, planteándose así lo que ha sido uno de los principales problemas en la definición del SQL3: la dificultad en la integración de los dos modelos, el relacional y el orientado al objeto. Esta integración, en nuestra opinión, se ha venido haciendo desde un primer momento con no mucha “limpieza”, convirtiéndose en uno de los temas más discutidos y criticados del borrador, ver Manola y Mitchell (1994). El último modelo, propuesto hace tan sólo unos meses, es el que se

[16] Sería más propio, quizá, hablar de “modelo de datos” ya que en SQL3, a diferencia de otros modelos como el del ODMG-93, no todo son objetos. Sin embargo, preferimos hablar de “modelo de objetos” ya que es la parte del modelo de interés para el tema que nos ocupa.

expone y aunque parece ser el definitivo aún pueden realizarse algunas modificaciones substanciales. De hecho, algunos aspectos del modelo que aquí se presenta han sido ya modificados. Las modificaciones se basan en las decisiones tomadas en la última reunión del comité que se celebró en el Ministerio de Industria y Energía, en Madrid, del 3 al 12 de febrero de 1997. En ese momento la presente Tesis ya estaba en su fase final, por lo que las decisiones que hemos tomado para la realización de MIMO se basan en el modelo propuesto en DBL: MAD-010 (1996). No obstante, en la exposición, resaltaremos las modificaciones de mayor relevancia.

El nuevo modelo permite:

1. Mantener el concepto de tipo de objetos soportado en propuestas anteriores como OBJECT TYPE. Se soportará ahora permitiendo filas con nombres y eliminando la definición de ADT objeto.

2. Mantener el concepto de tipo de valores, soportado en propuestas anteriores como VALUE TYPE. Los valores se pueden encontrar en las columnas de las tablas.

3. Mantener la compatibilidad con versiones anteriores permitiendo la aplicación de los conceptos de la orientación al objeto a datos, ya existentes, que estén soportados por el modelo relacional[17].

Para ello se proponen tres elementos básicos: tipos fila con nombre, tipos valor y tipos referencia. Con estos tres elementos tenemos el modelo de objetos básico del SQL3, que se define como sigue:

Un **tipo de objeto** es un tipo fila con nombre al que se le añade comportamiento. Cada ejemplar de un tipo fila es un **objeto** y se identifica por un valor único e inmutable

[17] Obsérvese la importancia de este beneficio aportado por el nuevo modelo, en relación con los anteriores propuestos. Los anteriores modelos de objetos soportaban la compatibilidad con el modelo relacional, pero no permitían la posibilidad de convertir al modelo de objetos esquemas que hubieran sido definidos utilizando el modelo relacional.

que genera el sistema. El **tipo referencia** permite soportar la noción de objeto compartido; además, las referencias permiten añadir comportamiento a las filas (utilizando referencias como parámetros en la llamada a una función) a la vez que convierten al SQL3 en un lenguaje fuertemente tipado. Los objetos pueden vivir en tablas o en columnas.

Los **tipos valores** serán usados como dominios de las columnas de una tabla o de los atributos de los tipos de objeto. Actualmente no se permiten referencias a tipos de dato valor, pero está previsto añadir esta capacidad en el SQL4.

El **comportamiento**, tanto de los tipos valor como de los tipos objeto, se implementa mediante funciones[18]. El SQL3 es ahora un lenguaje completo, por lo que la implementación de las funciones, a diferencia de ODMG-93, puede hacerse en el propio lenguaje, o utilizando otros lenguajes aceptados en el estándar como C, PASCAL, ADA, etc.

SQL3 soporta tipos de datos ***predefinidos*** y tipos ***abstractos***[19] de datos. Los primeros son proporcionados por el sistema, mientras que los segundos pueden ser definidos por un estándar, una implementación o una aplicación. Tanto unos como otros pueden servir de base para la definición de nuevos tipos por medio de lo que SQL3 llama tipos distintos. Un ***tipo distinto*** es un nuevo tipo que se obtiene de otro ya existente y que tiene la misma representación del tipo del que se deriva. SQL3 también soporta tipos de datos ***colección***, concretamente los tipos de datos conjunto, multiconjunto y lista[20]. Los elementos de una colección pueden ser elementos de un tipo de datos primitivo, de un tipo abstracto de datos o colecciones. Además SQL3 soporta la

[18] Aunque el borrador del estándar habla de rutinas, su editor, Jim Melton, siempre habla de funciones; concretamente la primera propuesta del modelo de objetos actual se refiere siempre a funciones por lo que nosotros hemos decidido mantener esta terminología.

[19] Esta clasificación de tipos, heredada del SQL-92 con ciertas ampliaciones, puede que se vea ligeramente modificada al añadir el nuevo modelo de objetos.

[20] En la reunión de Madrid se aprobó una propuesta de ANSI por la que el SQL3 inculirá el soporte básico para el tipo array, DBL: MAD-175 (1996).

semántica completa de los dominios[21] (recuérdese que el SQL-92 tan sólo soporta su definición). La figura 3.2 muestra la jerarquía completa de tipos predefinidos en la que, a diferencia del ODMG-93, no se incluyen los tipos colección[22]:

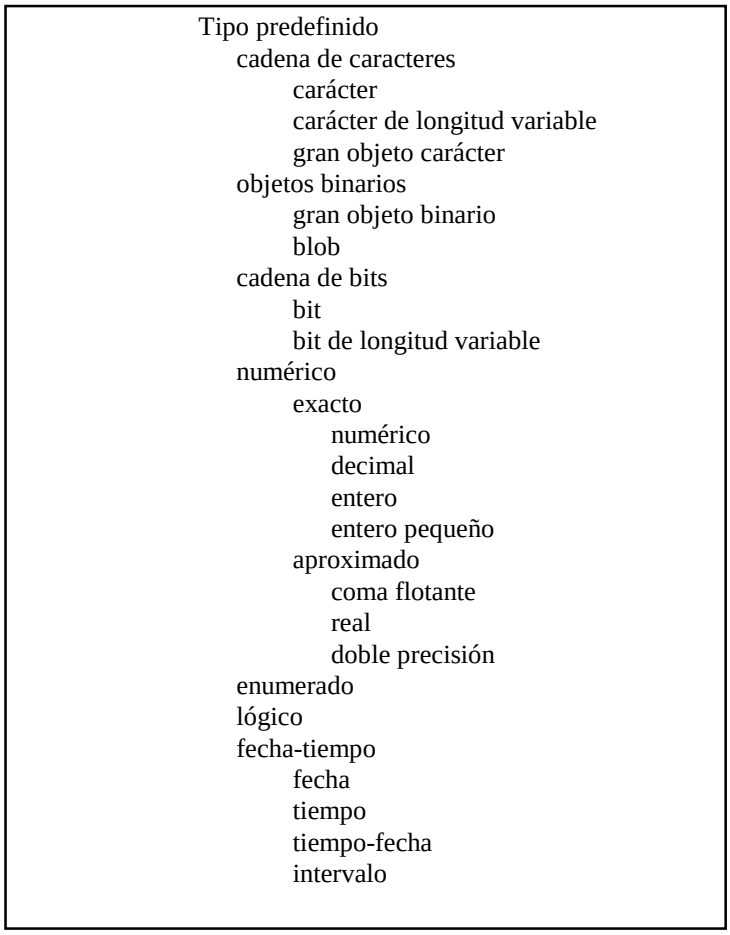


Figura 3.2: jerarquía de tipos primitivos

Se soportan jerarquías de supertipo/subtipo de modo que un tipo puede tener tantos subtipos como se desee. Igualmente, un tipo puede ser declarado subtipo de tantos tipos como queramos. Un subtipo hereda todas las características de sus

[21] En Madrid se presentó una propuesta, DBL:MAD-231 (1997) en la que sólo se permite la utilización de dominios en la definición de columnas. Para la definición de atributos, parámetros y variables se utilizarán, en lugar de los dominios, tipos definidos por el usuario y tipos distintos. La propuesta no se aceptó, pero tampoco se rechazó, por lo que la cuestión queda abierta para un ulterior estudio.

[22] El estándar define, en realidad, plantillas de tipo colección que permitirán la generación de tipos colección. La jerarquía de tipos muestra los tipos soportados por el estándar, por lo que las plantillas no se

supertipos directos (no se permite ***herencia inhibida***), que pueden ser varios (***herencia múltiple***), de modo que un ejemplar del subtipo puede sustituir a un ejemplar cualquiera del supertipo (***herencia de sustitución***). Además de las características heredadas, el subtipo puede tener características propias, de modo que éste será una especialización del supertipo (***herencia de especialización***). Todo ejemplar de un subtipo es, a su vez, ejemplar de todos sus supertipos (***herencia de inclusión***).

En versiones anteriores estas jerarquías se definían sobre tipos abstractos de datos (valor u objeto) y entre tablas. En la actualidad, el estándar las mantiene para tablas y para tipos abstractos valor, aunque suponemos que el concepto de jerarquía de tablas será modificado con el fin de soportar jerarquías de tipos de objetos de un modo consistente[23].

Existen aún muchos aspectos sin aclarar en el nuevo modelo de objetos, ya que actualmente la parte 8 del SQL3, DBL: MAD-010 (1996), tan sólo incluye la primera propuesta realizada del modelo en DBL: LHR-077 (1995).

Aunque el modelo de objetos del SQL3 es mucho más rico y completo que el del ODMG-93, su necesidad de integración con el modelo relacional ha retrasado en gran medida su definición.

Una de las limitaciones más importantes del SQL3 es que no permite la implementación de un ADT de un modo independiente a su definición; esto impide tener varias implementaciones para un mismo tipo, Manola y Mitchell (1994), además de dificultar la portabilidad[24].

introducen. La jerarquía del ODMG-93 sí las incluye, aunque Cattell (1995) hace esta misma consideración.

[23] En la reunión de Madrid se aprobó una propuesta, DBL: MAD-187, por la que las jerarquías de tablas sólo se permiten, a partir de ahora, entre tablas definidas sobre tipos fila con nombre. En el modelo de objetos del SQL3, antes de aprobar esta propuesta y tal como nosotros habíamos detectado, no se integraba bien el concepto de subtabla con el concepto de tipo fila con nombre.

[24] Aunque también en Madrid ha habido novedades en este sentido. DBL: MAD-226 (1996) es una propuesta de los expertos de Gran Bretaña, que plantea la necesidad de eliminar la definición de rutinas dentro de la definición de ADTs; la propuesta fue aceptada. AFNOR (Association FranÇais de NORmalisation) presentó una propuesta en este mismo sentido, DBL: MAD-235 (1997), que fue rechazada por motivos técnicos.

Otra crítica bastante generalizada al SQL3 que, aunque afecta más a la parte dinámica, comentamos debido a su interés, es la de que mantiene un modelo de objetos generalizado (es decir, una operación se distingue por todos sus argumentos) a diferencia del modelo clásico del ODMG-93 (en el que cada operación se distingue por el primer argumento). Sin embargo, en la reunión, ya mencionada en el epígrafe 3.2.2, entre Melton y miembros del ODMG, éste fue, precisamente, uno de los aspectos del SQL3 que se decidió modificar, DBL:YOW-031. El motivo por el que el SQL3 va hacia un modelo clásico es que éste es el modelo soportado tanto por C++, como por OQL y OMG (debido principalmente a que permite una mayor eficiencia en entornos distribuidos).

Además, al SQL3 se le acusa, ver Wade (1996), de ser excesivamente restrictivo en el tipo de datos que una consulta puede obtener como resultado, ya que éste siempre habrá de ser una tabla. No obstante, este aspecto está siendo tratado por el Object Merger Group.

Aunque, quizá, la crítica más fuerte es, paradójicamente, la realizada en el tercer manifiesto, Darwen y Date (1995), donde se afirma: *"Nosotros buscamos una base firme para el futuro de los datos. No creemos entonces que el lenguaje de base de datos SQL sea capaz de proporcionar tal base"*. Según Darwen y Date, las características de la Orientación al Objeto son ortogonales a las características del Modelo Relacional por lo que *"...el modelo relacional no necesita extensiones, ni correcciones, ni sumisiones, ni, sobre todo, perversiones, para adaptarse a algunos lenguajes de bases de datos que podrían representar la base que nosotros buscamos"*.

Aunque tanto ODMG-93 como SQL3, debido a que son modelos de implementación, presentan grandes limitaciones semánticas, el SQL3 proporciona un mayor poder de expresión derivado, en gran parte, del soporte de restricciones.

A pesar de todo, no cabe duda de que MIMO habrá de integrar el modelo de objetos del SQL3 y el del ODMG-93 (por lo que también se integra el OMG/OM), ya que son los estándares más importantes de la actualidad para bases de objetos.

3.2.4. MÉTODO UNIFICADO/UML

En los últimos años han surgido una proliferación importante de metodologías de análisis y diseño orientado al objeto, la mayor parte de ellas, como adaptaciones a metodologías estructuradas ya existentes, véase a título de ejemplo, Shaller y Mellor (1990), Jacobson (1993), Coleman et al. (1994), Robinson y Berrisford (1994), Yourdon et al. (1995), Martin y Odell (1995), Graham (1995), Henderson-Sellers y Edwars (1995). El método unificado, Booch y Rumbaugh (1995) es un intento de integrar dos de ellas: OOD (Object-Oriented Design) de Booch (1994) y OMT (Object Modeling Technique) de Rumbaugh et al. (1991). Además incorpora aspectos importantes de Objetory, Jacobson (1993).

El Método Unificado (MU) define un modelo estático, y un modelo dinámico. Además, define un metamodelo que permite definir el modelo formal subyacente del método. Dicho metamodelo se describe en forma textual y gráfica utilizando para ello la propia notación del Método Unificado.

En el momento en que se escribió este apartado sólo se había publicado la versión 0.8 del MU. Posteriormente se publicaron dos *addemda*: Unified Modelling Language V0.9, Booch et al. (1996a) y Unified Modelling Language V0.91, Booch et al. (1996b) publicándose, en enero de 1997, la versión 1.0, Booch et al. (1997). Unified Modelling Language (UML) es un lenguaje de modelado para orientado al objeto. Su modelo subyacente, aunque con algunas modificaciones, es el de la versión 0.8 del MU. Debido a que la versión 1.0 aparece cuando la presente Tesis Doctoral ya está en su fase final, no se incorporan las últimas actualizaciones de UML, por lo que todo cuanto en ella se diga es referente a la versión 0.8. No obstante, se señalarán los principales aspectos del modelo de objetos que han variado. Es importante señalar que las modificaciones más significativas del MU se acercan a MIMO, de tal modo, que algunas de las críticas al MU que nosotros habíamos planteado han sido subsanadas en la V1.0.

UML V1.0 ha sido presentado, en enero de 1997, al OMG Analisys & Design Task Force con el fin de convertirlo en la notación estándar para el desarrollo de

sistemas orientados al objeto. Según sus propios autores la V1.0 es ya una versión estable e utilizable.

A partir de ahora hablaremos del MU y no de UML[25] que es su nombre actual, ya que nos estaremos refiriendo a la V0.8.

El método unificado se utilizará en el presente trabajo en dos sentidos: por un lado su modelo nos servirá, junto con otros, como base del nuestro; y por otro lado, su notación será la base principal (aunque no la única) de la nuestra. A continuación pasamos a exponer los conceptos recogidos en el modelo del método unificado.

Modelo de Objetos (MU/OM)

Se trata de un modelo que es la base de una metodología de análisis y diseño orientados al objeto. En él se unen, por tanto, aspectos del modelo estático y del dinámico. Nos centraremos aquí sólo en aquellas partes que son de interés (estática y, aunque no completa, dinámica[26]) para nuestro trabajo remitiendo al lector interesado en otras, a la fuente original.

Es un modelo basado en **tipos**. Presenta una jerarquía en la que la raíz (TypeDecl) tiene dos subtipos: ClassDecl, de donde colgarán todos los tipos clase que siempre son definidos por el usuario, y NonclassDecl que se especializa en los distintos tipos predefinidos y los tipos definidos por el usuario (TypedefDecl). La jerarquía de tipos predefinidos dependerá del lenguaje soportado por cada herramienta concreta. Existen tres especies de tipos clase:

[25] Aunque cuando hablamos del Método Unificado utilizamos las siglas en castellano (MU), para UML hemos preferido mantener las siglas inglesas ya que es la terminología más habitual. Obsérvese que normalmente se habla del Método Unificado, pero no del Lenguaje de Modelado Unificado.

[26] De la dinámica se considera sólo, al igual que en los demás modelos estudiados, la relativa al comportamiento de los objetos los cuales, debido al enfoque del paradigma en que nos movemos, no son estructuras estáticas.

- ***Clase simple***.- se define como una nueva estructura que puede ser utilizada como cualquier otro tipo[27].

- ***Clase plantilla***.- es una estructura que permite, por ejemplificación, la generación de nuevas clases. No es un tipo en sí mismo.

- ***Clase ejemplificada***.- se obtiene como ejemplificación de una clase plantilla y puede ser utilizada como un nuevo tipo.

La figura 3.3 muestra la jerarquía de tipos del MU:

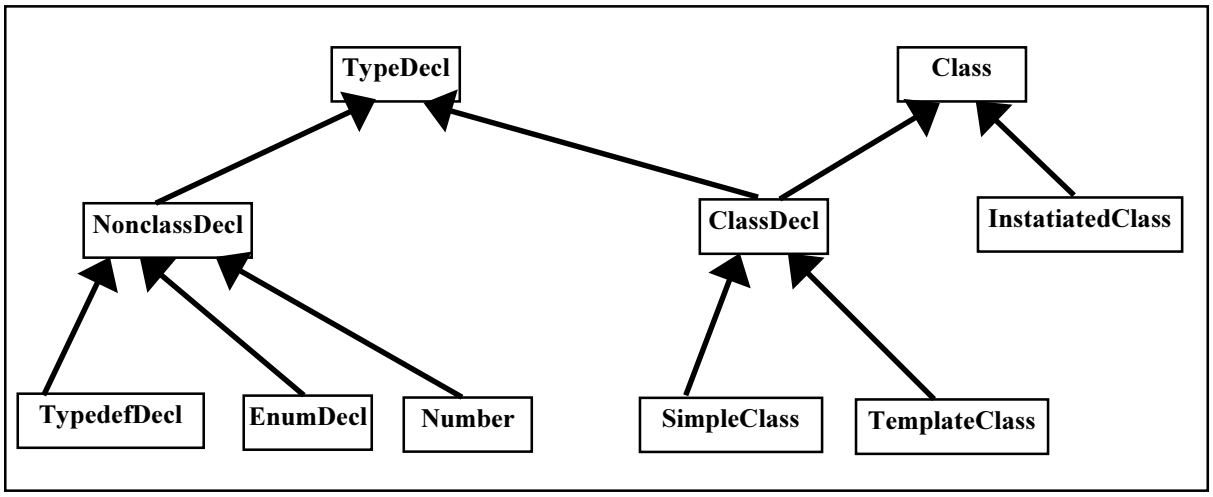


Figura 3.3: jerarquía de tipos del MU

Tanto las clases simples como las clases plantilla pueden anidarse; es decir, la definición de una clase puede contener a otra, dando lugar a las ***clases anidadas***.

El modelo soporta ***clases virtuales*** (aquellas que pueden tener subtipos y que pueden ser ejemplificadas); ***clases hoja*** (pueden ser ejemplificadas, pero no pueden tener subtipos); ***clases diferidas*** (no son directamente ejemplificables y deben tener

[27] Aunque no es muy consistente decir que una clase pueda ser utilizada como cualquier otro tipo, se ha tomado la traducción literal del original. Como también se puede observar, por el párrafo anterior y siguientes, en el MU se mezclan las clases y los tipos de un modo un tanto confuso.

subtipos); y ***clases extensibles*** (pueden ser extendidas pero no modificadas). El modelo está abierto a la incorporación de nuevos tipos de clases además de los expuestos.

Una clase se compone de **miembros** que pueden ser ***atributos*** u ***operaciones***. Cada miembro tiene un nivel de visibilidad, admitiéndose tres posibles: público (visible para cualquier objeto), privado (sólo visible por los objetos de la propia clase) y protegido (visible por los objetos de las subclases de la jerarquía). Cada miembro tiene un nombre y un ámbito; el ámbito puede ser, de clase (corresponde al concepto de variable de clase), o de ejemplar (corresponde al concepto de variable de ejemplar). Se soportan ***atributos derivados***.

Una ***operación*** define la signatura y lo que MU llama especificaciones sin interpretar (uninterpreted specification), que son nuevos elementos que se pueden añadir dependiendo del lenguaje concreto al que se aplique el método. En el caso de las operaciones las especificaciones sin interpretar podrían ser pre y post-condiciones.

Las clases mantienen entre sí **interrelaciones** que pueden ser: de generalización/especialización, de agregación o de asociación.

Las relaciones de **agregación** permiten obtener objetos por composición. MU/OM soporta dos tipos de agregación: física, sí el objeto componente puede estar en un sólo objeto compuesto, y de catálogo, cuando el objeto componente puede pertenecer a más de un objeto compuesto. Sin embargo, la agregación de catálogo se identifica únicamente por las cardinalidades: cuando el objeto componente participa con cardinalidad máxima uno, se trata de una agregación física; sí la cardinalidad máxima es superior a uno, la agregación es de catálogo. En nuestra opinión, es cierto el segundo supuesto, pero el primero no, ya que cuando la cardinalidad máxima es uno la agregación no tiene porqué ser necesariamente física. La agregación puede implementarse por referencia o por valor y el MU permite recoger esta semántica.

Las **asociaciones** son interrelaciones niveladas entre dos o más clases. Las interrelaciones pueden ser o no derivadas (al igual que los atributos derivados, son aquellas que se obtienen a partir de otras). Las asociaciones se pueden combinar con dos

tipos de restricciones, la restricción de inclusión (subset) y la restricción de exclusividad (or).

Una asociación puede tener miembros propios (atributos, operaciones e incluso interrelaciones) que se agrupan en un tipo especial de clase (clases de asociación); las **clases de asociación** son aquellas que se unen a una asociación permitiendo que ésta tenga sus propios miembros. Un caso especial de este tipo de clases son aquellas que sólo contienen atributos (**atributos de asociación**).

Una variante de los atributos de asociación da lugar a las **asociaciones cualificadas**. Una asociación cualificada es aquella que tiene un cualificador. Un cualificador es un valor de un atributo de asociación que es único dentro del conjunto de enlaces que tiene un objeto en dicha asociación. Es decir, un objeto y un valor de un cualificador identifican unívocamente un objeto a través de la asociación. El cualificador no tiene porqué ser un valor simple, pudiendo estar formado por una lista de valores.

Una **generalización** es la relación que se establece entre una superclase y sus subclases. Una subclase puede serlo de varias superclases dando lugar al concepto de herencia múltiple. Soporta dos tipos de generalización (or-generalización y and generalización) que se estudian en el epígrafe 4.4.4.

Clases, tipos y asociaciones tienen **ejemplares**. Un ejemplar de un tipo es un **valor** (ValueInstance) no actualizable; si se modifica, se trata de otro valor. Un ejemplar de una clase es un **objeto**. Cada objeto tiene un nombre que permite identificarlo (corresponde a la noción de OID -Object IDentifier-), y una agregación de valores (que forman el estado del objeto) que son a su vez ejemplares de los tipos de los atributos de la clase. Los ejemplares de las asociaciones son los **enlaces** (links) cada uno de los cuales es una tupla de referencias a objetos.

El MU será integrado por MIMO a fin de recoger la semántica que los modelos de los estándares no soportan. En nuestra opinión el MU tiene importantes aportaciones, entre ellas la unificación de tres metodologías de objetos ya existentes, lo que marca un punto de partida hacia la convergencia de modelos.

Es una metodología para el desarrollo de sistemas de información por lo que está muy influenciada por los lenguajes de programación (especialmente por el C++)[28] dejando de lado aspectos importantes de las bases de datos. En nuestra opinión, existe demasiada dependencia de los lenguajes ya que, en algunas ocasiones, sus funcionalidades como modelo semántico se limitan debido a restricciones impuestas por los modelos de implementación. Este es, en principio nuestro criterio, que se irá argumentando a medida que profundicemos más en el estudio de los modelos (ver capítulo 4).

Principales modificaciones del MU en UML V0.9, V0.91 y V1.0

La V0.9 de UML presenta sólo actualizaciones de carácter sintáctico. UML V0.91 incluye las modificaciones sintácticas a la V0.8 realizadas en la V0.9 y una modificación importante en el modelo de objetos: soporta generalizaciones exclusivas y solapadas (la V0.8, soportaba, tan solo, generalizaciones solapadas). Tal como explicaremos en el capítulo 4, está era una de las principales deficiencias que, en nuestra opinión, presentaba el modelo de objetos del MU. En UML V1.0 se mantienen las modificaciones de los addemda anteriormente mencionados y se incluye una sintaxis explícita para el soporte de generalizaciones exclusivas/solapadas y totales/parciales[29].

3.2.5. MERISE orientada al objeto

La metodología MERISE, desde su nacimiento, Tardieu et al. (1983), se ha visto sometida a numerosas extensiones, como por ejemplo la segunda generación de MERISE en Tardieu (1988), MERISE/2 en Panet et al. (1991) o MERISE-objeto en Rochfeld (1992).

Su modelo de objetos (OOM) se basa en el modelo E/R extendido, aunque en la actualidad, ver Morejon (1994), con modificaciones sustanciales. Soporta interrelaciones de grado dos o superior, así como generalizaciones, con herencia simple

[28] Una prueba de ello es la similitud existente entre la sintaxis propuesta para UML para modelado de sistemas de tiempo real y C++.

[29] Todos estos conceptos se discuten en profundidad en el capítulo 4.

y múltiple, que pueden ser de cuatro tipos, total/parcial y exclusiva/solapada. Soporta también generalización múltiple paralela. Define un rico sistemas de restricciones. Se admiten atributos derivados así como atributos multivaluados y agregados (es el soporte para el concepto de meronimia).

Un objeto se define como la unión en una misma unidad de un conjunto de datos y los tratamientos asociados. Los tratamientos en OOM son operaciones que se introducen en el modelo conceptual mediante la noción fase/tipos (phases-types). La fase corresponde al concepto de método en la terminología de objetos.

Una característica distintiva del modelo con respecto a los demás modelos estudiados es que soporta el concepto de interrelaciones entre interrelaciones así como de generalizaciones entre interrelaciones, lo que dota al modelo de una gran flexibilidad y semántica.

En nuestra opinión se trata más de un modelo E/R extendido, que de un modelo de objetos, ya que no se tratan aspectos como la distinción, o no, entre valores y objetos, la identificación de los objetos, la noción de clase o de extensión, etc. Su principal aportación es que se trata de un modelo con una gran potencia para el modelado conceptual (en nuestra opinión supera, sin lugar a dudas, la capacidad expresiva para el modelado conceptual del MU).

3.2.6. El lenguaje EXPRESS

STEP/EXPRESS[30], en Spiby (1994) y Schenck y Wilson (1994), es un lenguaje para el modelado de información desarrollado por STEP. Este grupo cuenta con varios modelos de normativa diferentes cuya integración lógica es posible debido, fundamentalmente, a que todos ellos están descritos en un lenguaje común con un modelo común, EXPRESS. En su definición de EXPRESS han contribuido una serie de lenguajes entre los que cabe destacar el ADA, C++, Pascal y SQL.

[30] EXPRESS es un lenguaje para el modelado de información en sistemas de automatización industrial, desarrollado por ISO TC184/SC4/WG5 (el WG5 recibe el nombre de "STEP Development Methods").

En Berre y Oldevik (1995) se definen las extensiones para orientación al objeto de EXPRESS, lo que se conoce como EXPRESS-OODL. Este lenguaje extiende la semántica de EXPRESS con los conceptos de IDL. Concretamente se toma de EXPRESS la parte de definición estática (estructura de los objetos) y se incorpora IDL a fin de permitir la definición del comportamiento en términos de operaciones.

El lenguaje EXPRESS se centra en la definición de **entidades**, siendo una entidad una "cosa" de interés. Las entidades se definen en términos de **datos** (que representan sus propiedades) y **comportamiento**. El comportamiento se representa mediante restricciones (éstas pueden ser, de unicidad, de no nulidad, de cardinalidad, reglas, etc.); los datos como atributos que pueden ser explícitos (si se le asigna un valor directamente) o derivados (si su valor se calcula a partir de otro u otros atributos).

Distingue entre tipos de datos y tipos literales. Un literal es un valor constante que se define a sí mismo y se incluyen los siguientes **tipos literal**: binario, entero, real, cadena de caracteres y lógico. Entre los **tipos de datos** se incluyen:

- ***Tipo simple***.- numérico, real, entero, cadena de caracteres, booleano (lógica bi-valuada), lógico (lógica tri-valuada) y binario.
- ***Tipo agregación***.- Es una generalización sobre los tipos conjunto, multiconjunto, lista y vector, de modo que un dato de tipo agregación puede ser sustituido por un dato de cualquiera de los tipos citados.
- ***Tipo definido***.- Son tipos definidos por el usuario como extensión a los tipos primitivos.
- ***Tipo entidad***.- Son tipos que permiten definir los objetos que se desean representar.
- ***Tipo enumerado***.- Es un conjunto ordenado de valores representados por nombres.
- ***Tipo genérico***.- Es un tipo genérico que puede ser visto como un supertipo que representa a todos los demás tipos de datos.

- ***Tipo selección***.- Permite definir un tipo como una lista de tipos especificados, de tal modo que el dominio del tipo selección es la unión de los dominios de dichos tipos.

Los tipos entidad se relacionan en jerarquías **supertipo/subtipo** soportándose herencia simple y múltiple. Además se soportan **interrelaciones binarias** entre dos tipos de entidad, así como interrelaciones de meronimia. En cada tipo de entidad es posible definir uno o varios atributos identificadores.

El lenguaje proporciona también sintaxis para especificar algoritmos que pueden ser procedimientos o funciones.

3.2.7. MEDEA

MEDEA es una metodología de desarrollo de bases de datos orientada al objeto, que se define en una Tesis Doctoral, Piattini (1994), realizada dentro de este mismo grupo de investigación.

MEDEA **cubre las tres fases principales del ciclo de vida** de una base de datos: análisis, diseño e instrumentación. Una de sus características distintivas y que ha contribuido a la selección de esta metodología como punto de partida para MIMO, es que, a diferencia que la metodología unificada, que procede del área de los lenguajes de programación (ver 3.2.4), MEDEA se concibe como una metodología **orientada al desarrollo de bases de datos**, aportando al análisis y diseño orientado al objeto los conceptos relativos a las bases de datos que apenas han sido tratados por otras metodologías. Por ello, se centra principalmente en la parte estática del modelo, aunque teniendo siempre presente aspectos de la dinámica que no pueden dejarse de lado en el paradigma de la orientación al objeto.

En MEDEA se presenta un modelo de objetos formal basado en los principales modelos existentes en el momento de su definición. Dicho modelo cubre de un modo uniforme las fases de análisis, diseño e instrumentación de una base de datos. Su sistema de tipos, en De Miguel y Piattini (1995), soporta los sistemas de tipos del SQL-92, SQL3 y ODMG-93. Además, en MEDEA se realiza una discusión precisa de los

conceptos que en ella se recogen, así como de los términos y acepciones utilizados. Este importante trabajo de unificación de terminología en un ámbito en el que la homonimia y sinonimia son la práctica habitual, será reutilizado en la definición de nuestro modelo con el fin de conseguir mantener un lenguaje técnico preciso, bien definido e intuitivo. Jensen et. al (1992) señalan la importancia de estas características para las bases de datos temporales; nosotros pensamos que son igualmente aplicables al paradigma de la orientación al objeto.

3.2.8. Otros modelos de interés

3.2.8.1. SUMM

SUMM (Semantic Unification Meta-Model) (1992) es un ejemplo importante de integración de modelos. Desarrollado por el Dictionary/Methodology Committee del IGES/PDES[31], establece las bases técnicas para abordar el problema de la integración entre modelos. En SUMM se presentan los principales beneficios de una integración de modelos así como las dificultades más relevantes que supone la realización de esta tarea.

El meta-modelo se describe en lógica de predicados de primer orden con algunas extensiones.

En nuestra opinión, SUMM es un metamodelo muy completo y definido con un gran rigor, pero de una gran complejidad. Es probable que su escasa aceptación se haya debido precisamente a la complejidad de su descripción (más que a la del propio modelo).

3.2.8.2. El método OOram

OOram (Object Oriented Role Analysis and Modeling), Wold y Lehne (1996), es un método genérico que constituye un marco para la creación de diferentes

[31] La Organización IGES/PDES (IPO) gestiona la contribución de los EEUU para los estándares de intercambio de modelos de datos de productos (STandard for the Exchange of Product model data, -STEP-). PDES es un acrónimo de Product Data Exchange using STEP; IGES es un acrónimo de Initial Graphics Exchange Specification.

metodologías. La idea de OOram es que no existe una única metodología válida para todos los propósitos, por lo que es necesaria la existencia de varias, de modo que cada una de ellas se adapte al propósito concreto para el que va a ser utilizada. El método OOram permite la creación de una familia de metodologías orientadas al objeto aplicables a distintos propósitos.

Soporta los conceptos de clase, tipo, objeto y papel, donde una clase es la implementación de un tipo (se permiten distintas implementaciones de un mismo tipo).

Los objetos de un mismo tipo pueden desempeñar distintos papeles; un modelo de papeles permite abstraer los objetos según la función que éstos desempeñen. Un papel, al igual que una clase, es una descripción de un conjunto de objetos pero con una diferencia sustancial: la clase describe un conjunto de objetos con características comunes; el papel describe un conjunto de objetos que desempeñan un mismo papel.

El modelo de OOram permite describir toda la semántica de los papeles: atributos que pueden mostrar los objetos que desempeñan un papel; responsabilidad de los objetos según sus distintos papeles; el conjunto de mensajes que un papel puede enviar a otro, etc.

No cabe duda que el modelado orientado al objeto tradicional, donde los objetos pertenecen a una clase dependiendo exclusivamente de sus características y sin posibilidad de migración de una clase a otra, deja muchos problemas de modelado sin resolver. Es evidente la necesidad de nuevos modelos que soporten, tanto el concepto de papel, como el de migración de objetos[32]. Aunque existen ya varios trabajos en este sentido, ver Mendelzon et al. (1994), Weiringa et al. (1995a), Weiringa et al. (1995b), Kristensen y Østerbye (1996), o el propio SUMM (3.2.8.1), el método OOram puede constituir un avance importante en este sentido.

[32] El MU V0.8 no soporta la migración de objetos. Sin embargo, este concepto se ha introducido ya en UML V1.0 mediante lo que se denomina "Dynamic classification" (clasificación dinámica). La clasificación dinámica permite determinar cuando un objeto puede cambier de clase durante la ejecución (notesé que, como ya se ha dicho, UML es un método orientado al desarrollo de sistemas de información, más influenciado por los lenguajes de programación que por las bases se datos).

Aunque la versión actual de MIMO no soporta el concepto de papel, en el capítulo 4 (4.5.2) se propone una posible extensión de MIMO para el soporte de papeles.

3.2.8.3. COMMA

COMMA (Common Object Methodology Architecture), Henderson-Sellers (1994) y Henderson-Sellers et. al (1995), es un proyecto cuyo objetivo es la definición de un conjunto de reglas y notación que permita la construcción de un metamodelo válido para cualquier metodología, a lo que el autor le da el nombre de metamodelado.

COMMA tiene como objetivo, además de contribuir a la investigación, el de colaborar en las labores de estandarización. Por ello, los resultados del proyecto son utilizados como entradas del OADTF (Object Analisys & Design TaskForce) del OMG, cuya misión es, entre otras, la de trabajar en la elaboración de metamodelos. COMMA tiene también relación con otros intentos importantes de metamodelo, como el MU (su metamodelo será un componente adicional de la propuesta de COMMA al OMG que se realizó a finales de 1996) y OMEGA/OPEN, Henderson-Sellers et al. (1995) (una metodología que integrará SOMA, en Graham (1994), MOSES, en Henderson-Sellers y Edwards (1995) y Martin/Odell, en Martin y Odell (1995)). Además de las metodologías nombradas, en COMMA se unen un total de 14 metodologías de orientación al objeto, Henderson-Sellers y Bulthuis (1996a).

COMMA define un metamodelo, Henderson-Sellers y Bulthuis (1996b), Henderson-Sellers y Firesmith (1997) y aunque la integración de modelos que realiza (modelos de metodologías) no es la misma que la realizada por MIMO (modelos de las distintas fases del ciclo de desarrollo de una base de datos), COMMA es un ejemplo de la importancia actual de los metamodelos, así como de los beneficios de éstos para solucionar el problema de la heterogeneidad del que hablábamos en el capítulo anterior. Tal y como señala el propio Henderson-Sellers (1996a), *"La convergencia está en aire"*.

3.2.8.4. OPEN

OPEN (Object-Oriented Process, Environment and Notation), Henderson-Sellers (1996b), Henderson-Sellers y Richard Dué (1997), es una metodología de orientación al objeto de tercera generación que integra las metodologías de MOSES y SOMA. Además incorpora aportaciones de Firesmith, Martín/Odell, ROOM, OOram, Synthesis and Mainstream Objects. En la actualidad se está estudiando la posibilidad de definir un metamodelo y una notación que permita la integración entre OPEN y UML.

Una de las principales características de OPEN es que proporciona soporte para bases de datos así como para Interfaces Gráficos de Usuario (IGUs), concurrencia, sistemas distribuidos, lógica difusa, inteligencia artificial y agentes inteligentes.

Existe una relación importante entre el OPEN y el proyecto COMMA, no sólo por la actual tendencia a la integración, sino también porque ambos proyectos están liderados por Henderson-Sellers. Es importante destacar que OPEN y COMMA soportan el mismo metamodelo de objetos.

3.2.8.5. Chimera

Chimera, en Bertino et al. (1993), Bertino et al. (1994) y Ceri et al. (1993), es el interfaz conceptual de IDEA (Intelligent Database Environment for Advanced Applications) un proyecto ESPRIT desarrollado en el Politécnico de Milano en colaboración con otras universidades y empresas. Chimera[33] está pensado para ser un lenguaje de bases de datos de nueva generación, integrando los paradigmas de las bases de datos orientadas al objeto, deductivas y activas.

El modelo de objetos de Chimera distingue entre valores (atómicos o complejos) y objetos. La representación de un valor es el valor en sí mismo, su estado no puede cambiar y su manipulación se realiza sólo a través de operaciones definidas por el usuario. Por el contrario, la representación de un objeto es su OID, por lo que el estado de los objetos sí puede variar; además los objetos se manipulan a través de operaciones definidas por el usuario.

[33] Recuérdese que una Chimera, en la mitología Griega, es un monstruo con cabeza de león, cuerpo de cabra y cola de serpiente.

Chimera distingue entre tipos valor y clases valor según que su extensión sea implícita o explícita (por enumeración). También distingue entre clase y tipo de objeto. La definición de una clase de objetos consta de una signatura y su implementación y lleva siempre asociada una extensión (que podrá ser explícita o implícita). Cada clase lleva asociado un tipo que se obtiene automáticamente de la definición de clase.

El modelo soporta herencia simple y múltiple, atributos derivados, restricciones y disparadores. Incluye también soporte de vistas, algo poco habitual en los modelos de objetos.

Chimera es un modelo riguroso y muy rico. Sin embargo, su objetivo se aleja totalmente de los objetivos de MIMO ya que, trata de integrar distintos paradigmas de las bases de datos, lo que hace que sea excesivamente grande y complejo.

3.2.8.6. DISCO

DISCO, Costilla et al. (1995), es un prototipo de un sistema de bases de objetos desarrollado en la E.T.S. de Ingenieros de Telecomunicaciones de la Universidad Politécnica de Madrid, en un proyecto parcialmente subvencionado por el programa nacional “Tecnologías de la Información y de las Comunicaciones”.

El modelo de objetos de DISCO incluye una parte importante del modelo relacional que los modelos de objetos “revolucionarios”, como el ODMG-93, no contemplan (soporta claves primarias, restricciones de integridad referencial, dependencias funcionales y multivaluadas, vistas).

La estructura del modelo de DISCO es una red semántica compuesta de nodos y enlaces etiquetados. Tanto nodos como enlaces son tipos. La implementación de los tipos es una clase persistente que contiene todas las instancias (valores u objetos del tipo). La persistencia se realiza en tablas (DISCO se implementa sobre un sistema relacional). Soporta agregación y generalización de tipos con herencia múltiple. El comportamiento se implementa mediante métodos.

3.2.8.7. El modelo de ROSES

ROSES (Rules Objects and Events), en Costa et al. (1996), es un Sistema de Gestión de Bases de Conocimiento y Eventos que está siendo desarrollado en el Departamento de Lenguajes y Sistemas Informáticos de la Universidad Politécnica de Cataluña. Su objetivo fundamental es el de reducir la distancia entre la especificación de una aplicación y su implementación.

Su modelo de objetos tiene tipos de datos predefinidos y clases. Una clase tiene una definición (que determina la estructura y el comportamiento dinámico de sus objetos) y una población. La descripción de una clase incluye la especificación de atributos (univaluados o multivaluados), la especificación de supertipos (ROSES soporta herencia simple y múltiple), claves, etc. El comportamiento viene determinado por eventos.

Los subtipos pueden ser solapados o disjuntos, para lo que ROSES introduce el concepto de partición. Todas las subclases de una misma superclase se agrupan en particiones, de modo que las subclases que pertenecen a una misma partición son disjuntas entre sí.

ROSES soporta, además, el modelado de eventos. El modelo de eventos, basado en el de clases, permite la definición de clases de eventos externos. La correspondencia entre éstos y el estado de los objetos se realiza mediante la inducción de eventos "estructurales" de las clases de objetos (inserciones de objetos, eliminaciones de objetos y actualizaciones de atributos).

El modelo de ROSES tiene, en nuestra opinión, dos importantes características en un modelo de objetos: permite el solapamiento entre subclases sin necesidad de recurrir a la herencia múltiple (este aspecto se discute en el apartado 4.4.4) y permite modelar, por medio de las particiones, los estados por los que pasa un objeto durante su vida; esta última posibilidad está muy relacionada con el tema de los papeles y la migración de objetos, aspectos cuya importancia ha sido señalada en el epígrafe 3.2.8.2.

Sin embargo, uno de los aspectos más limitados en ROSES es, quizá, su soporte de interrelaciones. Éstas se soportan, al igual que en los modelos de implementación,

mediante referencias, lo que, en nuestra opinión, restringe en gran medida sus posibilidades semánticas.

3.2.8.8. ESQL2: un SQL orientado al objeto con semántica F-lógica

ESQL2, Gardarin y Valduriez (1992) es un lenguaje compatible con el SQL-92 y que integra, de un modo uniforme, los conceptos de las bases de datos relacionales, orientadas al objeto y deductivas.

ESQL2 soporta un rico sistemas de tipos extensible mediante la definición de tipos abstractos de datos (ADT) con comportamiento. Para la implementación de ADT se permiten vínculos con diferentes lenguajes de programación. Los objetos son ADTs con un identificador único (OID). Soporta objetos complejos mediante referencias.

Incluye una serie de ADTs genéricos que permiten, por ejemplificación, la generación de nuevos tipos complejos. Tales ADTs genéricos son tuplas y colecciones (lista, conjunto, multiconjunto y vector). Los objetos complejos definidos mediante ADTs (genéricos o no) podrán usarse como dominios de tablas del mismo modo que los tipos de datos simples.

Se soportan jerarquías de ADTs con herencia, de modo que un subtipo puede redefinir los métodos heredados de su supertipo e incluir atributos o métodos propios. Se soporta el polimorfismo de herencia.

En nuestra opinión, el ESQL2, en cuanto a sus extensiones al modelo de objetos, no es si no una implementación del primer modelo de objetos propuesto para el SQL3 (ver 3.2.3).

3.2.8.9. El lenguaje TM

TM, en Balsters et al. (1993), es un lenguaje para la descripción de esquemas conceptuales de bases de datos desarrollado en la Universidad de **T**wente y en el Politécnico de **M**ilano. Esta basado en un modelo formal (FM), en Vreeze (1990) que

se describe en lógica de predicados de primer orden. Es realidad, TM es FM pero con una sintaxis más intuitiva.

TM incorpora las capacidades básicas de la orientación al objeto tales como: soporte de objetos complejos (mediante estructuras de registro anidadas y colecciones), identificadores de objetos, herencia simple y múltiple, polimorfismo, interrelaciones entre tipos de objetos mediante referencias, etc.

Su característica distintiva es la incorporación de:

- ***Descripciones predicativas de conjuntos***.- TM permite la definición de tipos de datos conjunto por intensión (mediante predicados), además de por enumeración. De este modo el conjunto de valores de un atributo de tipo conjunto es derivado y no depende del estado actual de la clase a la que pertenece.

- ***Restricciones estáticas con diferentes granularidades***.- Soporta restricciones de objeto, de extensión de clase y de estado de base de datos.

La principal aportación del FM es que supone un importante intento de proporcionar a la orientación al objeto un fundamento teórico que se basa en la teoría de conjuntos aplicada al modelo relacional.

3.2.8.10. MGCO2

MGCO2, Gargouri et al. (1995a) y Gargouri et al. (1995b), es un modelo definido con el fin de permitir realizar diseños conceptuales en un modelo de objetos e implementarlos en un modelo relacional.

Su modelo estático define: clases, objetos y propiedades. Las clases se pueden interrelacionar, soportando restricciones como la dependencia en existencia. Soporta cuatro tipos de generalización: total/parcial y solapada/exclusiva. Permite identificadores de objetos y atributos derivados.

El modelo se presenta con una definición formal de cada uno de los conceptos expresada en cálculo de predicados de primer orden. comprobar.

3.2.8.11. El modelo PDM

PDM (PROBE Data Model) es el modelo de datos de datos de un SGBD orientado al conocimiento denominado PROBE. PDM, en Manola y Dayal (1994), es una extensión del modelo de datos funcional Daplex y muestra la integración de las tendencias funcional, relacional y orientada al objeto. Las principales extensiones realizadas a Daplex son las necesarias para el manejo de los requerimientos de las nuevas aplicaciones para bases de datos, tales como aplicaciones de ingeniería y cartografía, que necesitan incluir semántica espacial y temporal.

Daplex incluía ya capacidades de orientación al objeto como el soporte de objetos complejos, clases o tipos de objetos, generalización y herencia, incorporación de comportamiento a los objetos así como atributos derivados, etc. PDM extiende el modelo, principalmente añadiéndole semántica espacial y temporal, e incorpora una definición formal del modelo de datos basada en el álgebra. Incorpora además, vistas, lenguaje de consulta de datos, y funciones calculadas[34].

3.2.8.12. El modelo de OASIS

OASIS, Pastor y Ramos (1995), es un lenguaje de definición de clases para modelar sistemas de información usando una aproximación de orientación al objeto. Su modelo estático permite la definición de atributos derivados, restricciones (estáticas y dinámicas) sobre los atributos, así como restricciones sobre las operaciones (pre y post-condiciones). Un aspecto significativo de OASIS y que creemos que es importante destacar, es la semántica de las agregaciones y generalizaciones que su modelo soporta.

OASIS permite los siguientes tipos de **agregación**:

[34] Una función calculada es aquella en la que sus valores de salida se definen por intensión a partir de un procedimiento que los calcula. En Manola and Dayal (1994) puede encontrarse una comparación entre *Funciones Calculadas* y *Funciones Almacenadas*.

- Agregación ***disjunta*** o ***no disjunta***.- dependiendo de que un mismo objeto componente pueda formar parte de diferentes tipos de objetos compuestos.

- Agregación ***univaluada*** o ***multivaluada***.- dependiendo de la cardinalidad entre el componente y el compuesto.

- Agregación ***estática*** y ***dinámica***.- dependiendo de que el objeto compuesto tenga una composición fija o que, por el contrario, sus componentes puedan variar en el transcurso del tiempo.

- Agregación ***flexible*** o ***estricta***.- dependiendo de que el objeto componente pueda existir sin necesidad de formar parte de ningún objeto compuesto o no.

- Agregación ***inclusiva*** o ***referencial***.- dependiendo de que el objeto componente sólo tenga existencia dentro del objeto compuesto (encapsulado en éste) o que por el contrario no exista una inclusión completa del objeto componente en el compuesto.

Se distingue otro tipo especial de agregación, dinámica y multivaluada, en la que el objeto compuesto representa una colección de objetos componentes. Este concepto, en OASIS, recibe el nombre de **asociación**.

En OASIS es posible modelar la **herencia** mediante generalizaciones o especializaciones (se proporcionan dos sintaxis diferentes). Una **especialización** puede ser ***temporal*** (responde al concepto de papel) o ***permanente*** (en este caso se obliga a que todo objeto de la clase padre sea también miembro de la clase hija, por lo que la destrucción de un objeto especializado implica la destrucción de su correspondiente objeto padre); también se distingue entre especialización con ***dependencia en identificación*** (el OID del hijo es el mismo que el del padre) o con ***independencia en identificación*** (cuando el hijo tiene su propio OID).

Una **generalización** puede ser ***disjunta*** (cada objeto de la clase padre puede pertenecer a una sola de las clases hijas) o ***no disjunta*** en caso contrario.

Por defecto, tanto agregaciones como generalizaciones son no disjuntas.

3.3. Necesidad de un aglutinador de modelos: MIMO

Como ya se ha dicho, es difícil realizar una comparación rigurosa de los modelos descritos. Cada uno de ellos se propone para un nivel de abstracción diferente, con unos objetivos diferentes y una terminología diferente. Por ello, en nuestro estudio sólo se ha pretendido resaltar para cada modelo sus características más relevantes y distintivas.

De todos los modelos, hay tres que, por su especial importancia, es necesario destacar: el MU para los niveles de análisis-diseño y el ODMG-93 y SQL3 para los niveles de diseño-implementación. Estos tres modelos presentan diferencias importantes: el MU, por ser un modelo de análisis, es difícilmente comparable con el SQL3 y el ODMG-93; estos últimos, al proceder de distintos paradigmas, presentan diferencias sustanciales en cuanto a la filosofía del propio modelo. Mientras el SQL3 es un modelo relacional extendido con capacidades de objetos, el ODMG-93 es un modelo de objetos puro.

Debido a esta heterogeneidad y disparidad de modelos se hace necesaria la definición de un aglutinador, MIMO, que permita integrar en un único modelo los conceptos de análisis, diseño e implementación, recogiendo a su vez los diferentes conceptos procedentes del paradigma relacional extendido y de los modelos de objetos puros.

4 MIMO: un Metamodelo de Objetos

Más por lo mismo que es imposible conocer directamente la plenitud de lo real, no tenemos más remedio que construir arbitrariamente una realidad, suponiendo que las cosas son de una cierta manera. Esto nos proporciona un esquema, es decir, un concepto o enrejado de conceptos. Con él, como a través de una cuadrícula miramos luego la efectiva realidad, y entonces, sólo entonces, conseguimos una visión aproximada de ella. En esto consiste el método científico. Más aún, en esto consiste todo el uso del intelecto.

José Ortega y Gasset, *La rebelión de las masas*

En este capítulo se define MIMO como un aglutinador de los principales modelos de objetos existentes. Comenzaremos con una introducción en la que se exponen, de un modo informal, los principales conceptos de MIMO (4.1). A continuación, se define cada uno de estos conceptos discutiendo los problemas existentes en la literatura, las distintas posibilidades que podemos tomar, así como la solución adoptada en cada caso. Para ello, se dividen los conceptos de MIMO en tres apartados: las estructuras básicas (4.2), el sistema de tipos (4.3) y el sistema de interrelaciones (4.4). Para finalizar, se presentan dos ejemplos de extensibilidad de MIMO como uno de los beneficios del modelo propuesto (4.5).

En este capítulo se va a abordar el objetivo principal del trabajo: la definición de un modelo, MIMO, que verifique la hipótesis planteada al comienzo de nuestra investigación (1.1). MIMO, no pretende ser un modelo más que aumente la heterogeneidad provocada por la variedad de los ya existentes. Tampoco es una simple "suma" de los principales modelos, sino que va más allá: soporta el modelado en las distintas fases del desarrollo de bases de datos, integrando, en análisis y construcción, el modelo del MU y en construcción y explotación, los modelos de SQL3 y ODMG-93. Además, incorpora características importantes de algunos de los modelos descritos en el capítulo 3, principalmente de los modelos de MEDEA (en aspectos característicos del modelado para bases de datos), STEP/EXPRESS y OOM.

La selección de los modelos subsumidos por MIMO se ha basado, por una parte en la necesidad de soportar los distintos niveles de abstracción de cada una de las fases del desarrollo de una base de datos y por otra en la universalidad del modelo elegido; en este sentido, los modelos de implementación tenían que ser, sin lugar a duda, los de los estándares para bases de objetos y, en cuanto al nivel de análisis, el MU[1] se espera que llegue a ser un estándar de "facto", ya que integra dos de las metodologías de orientación al objeto más extendidas de la actualidad.

Sin embargo, a MIMO se le exigen otra serie de características además de las expuestas. No sólo debe soportar los niveles de abstracción de las distintas etapas del desarrollo, sino que ha de hacerlo de un modo homogéneo, sin rupturas drásticas entre cada uno de los niveles[2]. Debe permitir la generación de esquemas SQL3 y ODMG-93, requisito que se deberá cumplir si, efectivamente, MIMO integra los modelos citados. Y además, como modelo conceptual, deberá proporcionar la mayor riqueza semántica posible sin penalizar excesivamente la sencillez del modelo. Esta capacidad semántica se mantendrá, en la medida de lo posible, en los modelos de diseño e implementación.

[1] Aunque en la actualidad se está trabajando por convertir UML en estándar OMG, su modelo subyacente, como ya se ha dicho, es una adaptación del MU, por lo que nuestro trabajo no se ve afectado en modo alguno.

[2] Recuérdese que, debido a que MIMO es un modelo y no una metodología, hablamos de niveles de abstracción y no de fases de desarrollo. En cada fase se podrán representar los conceptos según el nivel de abstracción de dicha fase.

Para la definición de MIMO nos hemos planteado los siguientes principios: en primer lugar se ha priorizado MIMO en sí mismo (buscando consistencia, capacidad semántica, etc.); y en segundo lugar, acercarse lo más posible a los modelos que integra. Sin embargo, como veremos a lo largo del presente capítulo, en muchas ocasiones nos hemos visto obligados a alejarnos de los modelos subsumidos por MIMO, con el fin de llegar a la definición de un modelo que cumpliera todos los requisitos que, a priori, le hemos exigido.

El capítulos 5 permitirá validar que el modelo que se describe a continuación cumple cada una de las características que a priori se le han exigido.

4.1. Visión general de MIMO

En MIMO se recogen todos los elementos básicos de un modelo de objetos. Además, debido a la necesidad de cubrir el SQL3, se ha buscado que en MIMO se puedan recoger de una forma sencilla los conceptos del modelo relacional. A continuación se da, a modo de introducción, una visión general de los aspectos más relevantes de MIMO para pasar posteriormente a un estudio más detallado y riguroso de cada uno de ellos.

MIMO permite describir parcelas del mundo real (universo del discurso) en el mundo informático, pasando por los niveles de análisis, construcción (se engloban aquí diseño e implementación) y explotación. Para ello, es necesario proporcionar estructuras que permitan representar, en cada una de estas etapas, los entes[3] del universo del discurso. Llamamos **objeto**[4] a la representación de un ente en un sistema de información y **valor** a la representación de una de las propiedades que caracteriza a un objeto. Por ejemplo, si el sistema de información es el de una biblioteca, la representación de un libro será un objeto, ya que el libro es un ente susceptible de conocimiento dentro de nuestro universo del discurso; la representación del número de páginas, título, etc. serán

[3] El diccionario de la Lengua Española, RAE (1992) define **ente** como "*lo que es, existe o puede existir*".
[4] **Objeto**, según la RAE (1992), es "*todo lo que puede ser materia de conocimiento o sensibilidad por parte del sujeto, incluso este mismo*".

valores ya que, tanto número de páginas como título son propiedades del ente a representar.

Aunque algunos modelos, en general puristas[5] de la orientación al objeto como el ODMG-93 o el Smalltalk, no realizan esta distinción entre valores y objetos, MIMO la mantiene debido a la diferencia conceptual y funcional existente entre ambos. Sin embargo, esta diferencia se suaviza en relación a los modelos evolucionistas como el del SQL3, manteniéndose MIMO en una posición intermedia entre estas dos tendencias, quizá, demasiado radicales en este aspecto. En realidad, el que algo sea valor u objeto sólo depende de que ese algo sea o no susceptible de conocimiento en sí mismo. Esta distinción procede, precisamente, del carácter "accidental" que Aristóteles atribuye a los entes que son susceptibles de estudio en las ciencias técnicas (ver 2.1). Así por ejemplo, el color de las tapas del libro es un valor (ya que no es materia de conocimiento en sí mismo), mientras el color, en una fábrica de pinturas, podría ser un objeto (podría desearse conocer de cada color, su tonalidad, brillo, etc.). Por tanto, pensamos, en contra de la opinión de los puristas de la orientación al objeto, que existe una distinción conceptual y funcional significativa entre valor y objeto, pero que sin embargo, su representación final en el sistema informático no ha de ser muy distante, alejándonos así de las tendencias menos puristas; si un mismo "algo" puede ser considerada valor u objeto, y el que se considere de uno u otro modo tan sólo depende del interés concreto de cada sistema, su representación final ha de ser, sino idéntica, muy similar.

En el mundo real es posible aplicar el principio de abstracción para agrupar entes que comparten unas determinadas características. La aplicación de este mismo principio a los objetos y a los valores nos lleva a la distinción entre **tipos de objeto** (agrupan objetos con características comunes) y **tipos de valor** (agrupan valores con características comunes). Dependiendo del nivel en que nos encontremos hablaremos de **tipos** (análisis) o de **clases** (en construcción y explotación). Una clase siempre será la implementación de un tipo, bien sea éste un tipo de objeto o un tipo de valor. MIMO define un rico sistema de tipos valor imprescindible para un buen modelado. En dicho sistema se incluyen tipos primitivos, tipos definidos por el usuario, colecciones, etc.

[5] Utilizamos los términos purista/no purista, revolucionario/evolutivo indistintamente y sin ninguna connotación sobre nuestra posición respecto a estas dos tendencias.

Del mismo modo que un ente tiene una identidad, única e invariable durante toda su existencia, que le distingue del resto de los entes y que permite determinar que se trata del mismo a pesar de las transformaciones que éste sufra durante su periodo de vida, un objeto (representación de un ente) debe tener, igualmente, identidad única. Para ello MIMO soporta el concepto de identificador de objeto (IDO), a la vez que mantiene la posibilidad de definir claves primarias.

MIMO presenta un sistema de interrelaciones[6] que permite modelar las relaciones que unos entes establecen con otros. El sistema de interrelaciones de MIMO se construye a partir de un conjunto de interrelaciones primitivas (interrelaciones niveladas, interrelaciones de generalización e interrelaciones de meronimia) que puede ser ampliado combinándolas con una serie de restricciones. Una característica esencial y distintiva de MIMO con respecto a los modelos estudiados es que considera el tipo de interrelación nivelada como un subtipo del tipo de objeto. Los motivos que han llevado a esta decisión, así como las implicaciones que ello conlleva, son analizadas en el epígrafe 4.4.

A continuación pasamos a estudiar con más detalle cada uno de estos conceptos. Para cada uno de ellos se dará una definición y se planteará una breve discusión en aquellos casos en que se considere necesario. Posteriormente se definirán en profundidad el sistema de tipos y el sistema de interrelaciones de MIMO.

4.2. Caracterización de los constructores básicos de MIMO

Definición 1: Objeto y valor

[6] Se utiliza "interrelación" como traducción de "relationship" a fin de distinguir los términos "relation" y "relationship" del inglés. **Interrelación**, según la RAE (1992), es "*correspondencia mútua entre personas, cosas o fenómenos"*. Según esta definición el concepto de interrelación tiene una connotación de reciprocidad que no siempre se da en las "relationship" (v.g. la interrelación de amistad es recíproca, no así la de compra-venta en la que el rol de los participantes en la interrelación es completamente diferente); además "relationship" tiene un nivel de abstracción mayor, correspondería a la capacidad de relacionarse o "relacionamiento". Quizá el término "relación" se ajusta más a este significado pero, la traducción de "relation" está directamente vinculada con "relación", por lo que no es válido.

*"Denominamos **objeto** a la representación, en un sistema de información, de un ente del mundo real que es susceptible de conocimiento en sí mismo".*

Todo objeto encapsula un conjunto de características[7] que pueden ser: características estructurales (describen su estructura), características dinámicas (describen su comportamiento), características de relacionamiento (describen las interrelaciones que mantiene con el resto de los objetos) y características de integridad (recogen la restricciones semánticas del objeto). Cada características de un objeto corresponde a una propiedad del ente al que representa.

*"Denominamos **valor** a la representación de una característica estructural de un objeto[8] (y por tanto, a la representación de la propiedad del ente correspondiente). El valor pueden describirse en función de otros valores dando así lugar a un **valor compuesto**".*

Un valor, al igual que un objeto, tiene un conjunto de características estructurales (cuya representación es también un valor), dinámicas, de relacionamiento y de integridad. Los valores se distinguen de los objetos porque éstos últimos tienen una característica estática especial, denominada IDentificador de Objeto (IDO) que los dota de identidad, algo de lo que los valores carecen.

Definición 2: Tipo de datos (valor y objeto)

Aunque el estudio de los tipos de datos en MIMO se realiza, por su importancia dentro del modelo, en un epígrafe aparte, se da aquí una primera aproximación con el fin de poder continuar con la descripción del modelo.

- Se llama **Tipo Objeto** (TO) a la *"descripción de un conjunto de objetos que tienen las mismas características"*.
- Se llama **Tipo Valor** (TV) a la *"descripción de un conjunto de valores que tienen las mismas características"*.

[7] Ver definición 7

[8] Está definición lleva consigo una importante consecuencia: las características estáticas no pueden ser objetos. Analizaremos las implicaciones de esta consecuencia en apartados posteriores.

- Llamamos **tipo de datos**, *bien a un tipo objeto bien a un tipo valor* (ver 4.3).

Definición 3: Ejemplar

*"Denominamos **ejemplar** de un TO, o de un TV, a cada uno de los objetos, o valores, que satisfacen dicho tipo".*

*"Denominamos **ejemplar de clase***[9] *a cada uno de los objetos o valores de dicha clase".*

Un término más habitual en la literatura para hacer referencia a este concepto, pero que sin embargo no se ajusta al significado que en este contexto se le atribuye, es el de instancia[10]. Ya que el castellano introduce el vocablo ejemplar[11], que traduce directamente "instance" del inglés y que es ya utilizado con el mismo sentido en otros entornos (v.g. biología, psicología...), hemos preferido adoptar dicho término en lugar de instancia a pesar de que éste último esté más extendido. Al hecho de crear un ejemplar de un tipo le llamaremos ejemplificación (en lugar de instanciación que es el término que habitualmente se emplea).

Definición 4: Extensión de tipo

Definimos **extensión de tipo**, basándonos en OMG (1995) como *"el conjunto de ejemplares que satisfacen dicho tipo".*

La extensión de un tipo puede ser:

- ***extensión real***.- cuando el tipo se define por enumeración (es el caso, por ejemplo, de un tipo enumerado, o de un dominio definido por extensión).

- ***extensión virtual***.- cuando el tipo se define a través de algún predicado, o rango que deben satisfacer los ejemplares de dicho tipo (por ejemplo el tipo entero, los dominios definidos por intensión, o los tipos de objetos).

[9] Ver definición 5

[10] Instancia, definido por la RAE (1992) como *"memorial, solicitud"*.

[11] Ejemplar, definido por la RAE (1992) como *"Cada uno de los individuos de una especie o género"*.

El sistema de interrelaciones de MIMO permite definir jerarquías de tipos/subtipos (ver 4.4) en las que la extensión de un subtipo ha de ser siempre un subconjunto de la extensión de sus supertipos.

Definición 5: Clase

Se define la **clase** como *"la implementación de un tipo (valor u objeto)"*.

En MIMO existen tipos a nivel de análisis y clases a nivel de construcción y explotación. Un tipo en análisis se convierte en una clase en construcción. Esto ocurre tanto con los TV como con los TO. Aunque no es muy habitual hablar, por ejemplo, de la clase del tipo entero, nosotros consideramos que también en estos casos (tipos valor que incorpora el sistema) existe tipo e implementación, y además ésta podrá variar de unos sistemas a otros. Por tanto, diremos que en MIMO sólo hay tipos en análisis y clases en construcción y explotación. Sin embargo, cuando la implementación del tipo la realiza el sistema, hablaremos de tipo en todos los niveles para no contribuir a la confusión terminológica, pero teniendo siempre presente la distinción conceptual que hemos establecido.

En MIMO se introducen los conceptos de clase abstracta y clase diferida, cuyos significados es conveniente discutir, debido a que estos dos términos se vienen usando de un modo muy confuso en la literatura. Una ***clase diferida***, según Meyer (1988), *es "una clase que contiene rutinas diferidas",* es decir, una clase que delega la implementación de alguno de sus métodos a sus subclases; una consecuencia directa de esta definición es que una clase diferida no puede ser directamente ejemplificada, a lo que Meyer se refiere como la regla de no ejemplificación de las clases diferidas y que define como sigue: *"regla de no-ejemplificación de las clases diferidas: la creación de ejemplares no puede ser aplicada a una entidad cuyo tipo viene determinado por una clase diferida"*. Sin embargo, en muchas ocasiones esta consecuencia se ha transformado en la definición de clase diferida, lo que hace variar sustancialmente el significado de éste término.

Algunos autores introducen el término de clase abstracta para hacer referencia a las clases no ejemplificables y otros, utilizan dicho vocablo como sinónimo de clase

diferida, Marcos et. al (1997b). Por todo ello, existe una confusión respecto a estos dos términos que ha contribuido a equiparar dos conceptos que son diferentes.

Este es, por ejemplo, el caso del MU donde se define clase diferida como sigue: *"clase diferida es aquella que debe tener subclases y que no puede ser ejemplificada (también conocida como clase abstracta pero usamos aquí el término de Meyer por ser más descriptivo"*, Booch y Roumbaugh (1995). En el MU se equipara el concepto de clase abstracta y clase diferida y, además, se atribuye a Meyer la definición de clase diferida como aquella que no es ejemplificable; sin embargo nosotros pensamos, y Meyer así lo plantea, que el hecho de que una clase diferida no sea ejemplificable es tan sólo una consecuencia de su definición.

En realidad, el concepto de clase diferida en el MU corresponde a una consecuencia directa de la definición dada por Meyer para este mismo término; además, la equivalencia entre la definición de clase diferida dada por Meyer y la del MU no es estricta y puede llevar a confusiones entre los conceptos correspondientes a ambos términos.

En nuestra opinión clase abstracta y diferida son conceptos diferentes, si bien están estrechamente relacionados, Marcos et. el (1997b):

- Definimos **clase abstracta** como *"aquella que no tiene una correspondencia directa con la definición de ningún ente identificable del universo del discurso y que se introduce en el esquema con el fin de aumentar su nivel de abstracción"*. Una consecuencia directa de esta definición es que una clase abstracta no podrá ser nunca ejemplificada directamente.

- Definimos **clase diferida**, basándonos en Meyer, como *"aquella que delega la implementación de alguno de sus servicios*[12] *a sus subclases y que se introduce a fines de reutilización"*. Por tanto, una clase diferida tampoco puede ser ejemplificada directamente.

[12] Ver definición 11

Tanto las clases abstractas como las diferidas son no ejemplificables, por lo que necesariamente deberán tener subclases. Sin embargo, estos dos términos hacen referencia a dos conceptos diferentes con diferentes funcionalidades, ya que en muchas ocasiones es necesario definir clases no ejemplificables, independientemente de que éstas implementen o no sus servicios.

Además, la distinción que nosotros establecemos entre los conceptos tratados afecta a la etapa de desarrollo en la que cada uno de ellos se introduce: la noción de clase abstracta corresponde al nivel de análisis, ya que se introduce con fines de clasificación, mientras que la clase diferida es una noción de construcción, pues es en esta fase donde se plantean las distintas posibilidades de implementación. Esta opinión es también defendida en Wirfs-Brock et al. (1990) donde se define clase abstracta como aquella que no puede ser directamente ejemplificable, *"las clases que no están capacitadas para producir sus propio ejemplares se llaman clases abstractas"*; para Wirfs-Brock et al. (1990), una clase abstracta puede ser diseñada, o no, como una clase diferida: *"Las clases abstractas pueden ser diseñadas de dos modos diferentes. Pueden proporcionar la implementación funcional completa de su comportamiento. O pueden proporcionar una plantilla del comportamiento que será definido por los métodos de sus subclases específicas"*.

La idea de Wirfs-Brock apoya nuestra propuesta de distinguir entre clases abstractas, como un concepto de análisis, y clases diferidas, como un concepto de construcción. Por ello, y aunque en la literatura generalmente sólo se habla de clase abstracta, es necesario introducir también el concepto de **tipo abstracto**[13] como aquel que no es directamente ejemplificable. Tendremos, pues, tipos abstractos en análisis y clases abstractas y diferidas en construcción y explotación.

Aunque en principio parece que toda clase no ejemplificable (abstracta o diferida) se utiliza con fines de abstracción y reutilización, sin embargo, existen algunos casos en lo que esta afirmación no está tan clara. Por ello, podría pensarse que existen clases no ejemplificables cuya naturaleza es diferente a la de las clases

[13] Obsérvese que tipo abstracto en este contexto tiene una acepción diferente a la utilizada para este mismo término por algunos modelos como, por ejemplo, el ADT del SQL3.

abstractas y a la de las diferidas. Nosotros hemos encontrado dos ejemplos que podrían responder a este hecho, Marcos et al. (1997b):

1. El primer caso podría ser el de la herencia paralela que aparece, entre otros modelos, en el del MU; en la figura 4.1 se plantea un ejemplo tomado de Booch y Rumbaugh (1995):

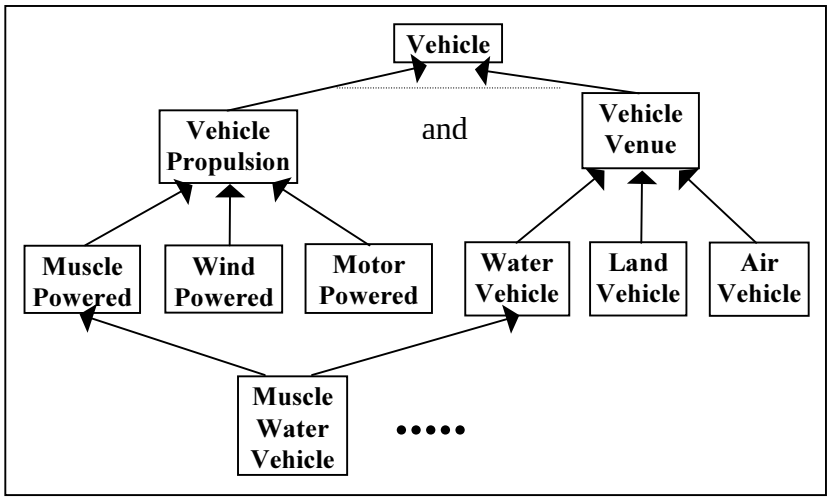


Figura 4.1: ejemplo de herencia paralela en el MU

En este ejemplo, Muscle, Wind y Motor Powered, así como Water, Land y Air Vehicle, son clases no ejemplificables. Sin embargo, esta restricción no se debe a que tales clases se hayan introducido, al igual que ocurre con las clases abstractas, por motivos de abstracción respecto a sus subclases, sino que las subclases se introducen por necesidades del modelo: éste es el modo en el que el MU soporta el solapamiento entre subtipos[14] (ver 4.4.4). No obstante, aunque los motivos últimos por los que se introducen clases no ejemplificables en una jerarquía de herencia paralela no coincidan con los motivos de introducción de clases abstractas en un modelo, el resultado final es similar: una jerarquía de clasificación en la que

[14] Es importante señalar que en UML V0.91, Booch et al. (1996b) y UML V1.0, Booch et al. (1997), ya se soporta el solapamiento directamente sin obligar, por tanto, a la creación de clases abstractas, por lo que la discusión planteada es válida tan sólo para el MU V0.8, Booch y Rumbaugh (1995). La notación

las clases no ejemplificables constituyen una abstracción respecto a sus subtipos. Por ello, quizá, estas clases no ejemplificables podrían también considerarse clases abstractas.

2. Un segundo caso es el que plantean algunos modelos de objetos que exigen que cada familia de tipos tenga un único supertipo máximum. Esta restricción obliga, tal como señala Melton (1994), a la creación forzada de un tipo no ejemplificable que se utilizará como supertipo máximum. Un ejemplo es el tipo “Denotable_Object” de la jerarquía de tipos del ODMG-93. Este caso responde al de una clase no ejemplificable, aunque sería discutible si es o no abstracta. En realidad se trata de una clase genérica respecto a sus subclases, pero introducida de un modo artificial, por lo que, probablemente, las subclases no tendrán características comunes que puedan ser heredadas de esta nueva superclase.

A pesar de que éste es el único ejemplo de clase no ejemplificable y no abstracta que hemos encontrado, es posible que existan otros, por lo que pensamos que es conveniente distinguir entre clases no ejemplificable, clases abstractas y clases diferidas.

Por tanto, tal y como se expone en Marcos et al. (1997b), existen clases abstractas y clases diferidas. Ambas son siempre no ejemplificables, ver figura 4.2, si bien lo contrario no es cierto.

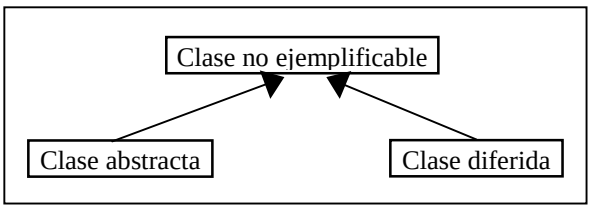


Figura 4.2: generalización de clases no ejemplificables

empleada para las generalizaciones múltiples (AND-Generalization en UML) también se ha modificado en la V0.9, Booch et al. (1996a).

En la figura 4.2 se presenta una generalización parcial (pueden existir clases no ejemplificable que no sean ni abstractas ni diferidas) y solapada (una clase abstracta puede ser también diferida y viceversa). Una clase abstracta es la construcción de un tipo abstracto. Las clases diferidas se introducen en construcción y pueden, o no, provenir de tipos abstractos. La clase no ejemplificable pura (ni abstracta, ni diferida) es una noción de construcción y los únicos ejemplos de aplicación que hemos encontrado se deben a restricciones del modelo.

Sin embargo, MIMO es un modelo muy flexible que apenas impone restricciones en el modelado. Además, ninguno de los modelos estudiados propone esta distinción y los casos en que encontramos útil su aplicación son excepcionales. Por ello, y aunque conceptualmente defendemos la clasificación propuesta en la figura 4.2, en MIMO no se introducirá el concepto de clase no ejemplificable pura; es decir, la jerarquía mostrada será total: toda clase no ejemplificable es abstracta o diferida.

Definición 6: Extensión de clase

Definimos extensión de clase como el *"conjunto de ejemplares de una clase en un momento dado"*.

La extensión de una clase es un subconjunto de la extensión del tipo al que implementa.

La extensión de una clase puede ser:

- ***extensión real***.- cuando se define por enumeración. Tienen extensión real:
    - ◊ Las clases de los tipos de objeto en explotación
    - ◊ Las clases de los tipos valor con extensión real (ver definición 4), tanto en construcción como en explotación.
- ***extensión virtual***.- cuando se define a través de algún predicado, o rango que deben satisfacer los ejemplares de dicha clase. Tienen extensión virtual:
    - ◊ Las clases de los tipos de objeto en construcción

- ◊ Las clases, cuyos tipos valor tienen extensión virtual (ver definición 4), tanto en construcción como en explotación.

MIMO soporta interrelaciones de generalización entre clases (ver 4.4) en las que la extensión de una subclase siempre ha de ser un subconjunto de las extensiones de cada una de sus superclases.

Definición 7: Característica (de tipo/clase y de ejemplar)

*"Se llama **característica** a cada una de las propiedades de un tipo/clase que, en conjunto, permiten diferenciarlo de los demás tipos/clases"*.

Se distinguen las siguientes especies de características:

a) **características estáticas**, que se dividen a su vez en:

a.1) **características estructurales**: atributos[15], IDO[16] (sólo aplicable a tipos de objeto).

a.2) **características de relacionamiento**: interrelaciones[17]

a.3) **características de integridad**: restricciones de integridad[18]

b) **características dinámicas**: servicios[19].

Una característica puede serlo de tipo/clase (cuando afecta por igual a todos los ejemplares del tipo/clase) o de ejemplar (cuando afecta de forma distinta a cada ejemplar). En algunos modelos, como Nelson (1991), se distingue sólo entre atributos de clase y de ejemplar. Los primeros son aquellos que se comparten tanto en nombre como en valor por todos los ejemplares de la clase, mientras que los atributos de ejemplar sólo se comparten en nombre, pero cada ejemplar tiene su propio valor. En nuestra opinión, esta distinción es válida para las características en general no sólo para

[15] Ver definición 8
[16] Ver definición 10
[17] Ver definición 14
[18] Ver definición 13

[19] Ver definición 11

los atributos, ya que también existen servicios, interrelaciones y restricciones de tipo/clase. Así, por ejemplo, en ODMG-93, el supertipo es una característica del tipo, ya que se trata en realidad de una interrelación entre tipos y no entre ejemplares (ver 4.4.2.1).

Las características de un objeto pueden clasificarse, según su nivel de visibilidad, en privadas (accesibles a los servicios del propio objeto), protegidas (accesibles, además, por los objetos definidos en subtipos del tipo del objeto) y públicas (accesibles por todo objeto que tenga privilegios para ello).

Definición 8: Atributo

*"Se llama **atributo** a una característica de un tipo/clase que, junto con sus otros atributos, definen la estructura de los ejemplares de dicho tipo/clase".*

Los atributos se pueden clasificar desde dos puntos de vista:

a) ***Lógico***, por el que se distingue entre atributos base (cuando el valor del atributo se asigna, y obtiene, de modo directo) y derivados (cuando el valor del atributo se calcula o infiere a partir de otro u otros atributos).

b) ***Físico***, por el que se distinguen atributos reales (si almacenan su valor), o virtuales (si su valor se obtiene en el momento de la recuperación).

Los atributos base son siempre reales, mientras que los derivados pueden ser reales o virtuales. En análisis sólo existe distinción entre atributos derivados o base. En construcción se especificará, por motivos de eficiencia, cuando un atributo derivado es real o virtual.

En muchos sistemas -como en Postgres- se distingue un tipo especial de atributo denominado de "referencia" que se utiliza para representar asociaciones entre objetos, y que, como es lógico tiene como tipo de datos el tipo de identificador de objeto. Este tipo de atributo es proporcionado también por algunos estándares como ODMG-93 y ha sido recientemente incluido en el SQL3, DBL: MAD-004 (1996). Sin embargo, MIMO no

soporta el atributo de referencia ya que, como veremos al estudiar su sistema de interrelaciones, en MIMO existen interrelaciones en análisis y en construcción lo que elimina la necesidad del atributo de referencia.

Existe un tipo especial de atributos (las claves) que se mantienen en el modelo debido a motivos semánticos y por compatibilidad con SQL3 y con ODMG-93.

Definición 9: Clave primaria y clave ajena

*"Denominamos **clave primaria** a un atributo o conjunto de atributos que identifican unívoca y mínimamente los ejemplares de un tipo de objetos y de la clase que lo implementa".*

La clave primaria lleva asociada la regla de integridad de entidad que especifica que los atributos que la componen no pueden tomar valores nulos.

SQL3 soporta el concepto de clave primaria y claves alternativas. En MIMO se recoge el concepto de clave primaria y el de clave alternativa mediante una restricción[20] de unicidad a la que opcionalmente se le puede añadir una restricción de no nulidad. ODMG-93 sólo soporta el concepto de clave alternativa.

*"Denominamos **clave ajena** a un atributo o conjunto de atributos que son clave primaria, o alternativa, en otro tipo de objetos".* La clave ajena debe verificar siempre la restricción de integridad referencial[21].

Aunque ODMG-93 no soporta el concepto de clave ajena SQL3 sí. En MIMO se mantendrá por compatibilidad con este último estándar. En realidad, MIMO no necesita soportar el concepto de clave ajena para implementar interrelaciones tal y como se hace en SQL3, ya que las interrelaciones en MIMO tienen existencia propia, incluso en implementación. Por tanto, la clave ajena del SQL3, como referencia a una clave primaria, se podría soportar a través de interrelaciones. Sin embargo, la clave ajena del SQL3, como referencia a una clave alternativa, no se puede reflejar en MIMO, salvo

[20] Ver definición 13

[21] Ver definición 13

manteniendo el concepto de clave ajena. Es por este motivo por el que se ha decido mantener en MIMO la clave ajena.

Definición 10: Identificador de objeto (IDO)

Definimos IDO, basándonos en las definiciones propuestas en OMG (1995) y en Weiringa y De Jonge (1995) como: *"característica estructural de un objeto que se vincula con éste mediante una conexión fija uno-a-uno y que es independiente del resto de sus características"*.

La conexión fija uno-a-uno entre el IDO y el objeto implica la inmutabilidad del IDO durante toda la vida del objeto; además, tampoco se permitirá su reutilización una vez que el objeto haya desaparecido.

El IDO permite diferenciar entre objetos que son iguales en un momento determinado (sus características estructurales tienen los mismos valores, tienen las mismas características de integridad, mantienen interrelaciones con los mismos objetos y también sus características dinámicas son las mismas, pero sin embargo tienen diferentes IDOs) e idénticos (cuando su IDO también coincide).

Durante los últimos años los conceptos de clave primaria e IDO se han usado, junto con el de subrogado (identificador interno), con cierta confusión respecto a su naturaleza y ámbito de aplicación. En Wieringa y De Jonge (1995) se puede encontrar un estudio comparativo entre los conceptos de clave primaria, IDO y subrogado. Los diferencias más significativas se resumen en la tabla 4.1 y se explican a continuación:

Clave	IDO	Subrogado
Concepto de Base de Datos	Concepto de modelado	Concepto de implementación
Representan información actualizable	Representan información no actualizable	Representan información no actualizable
Único en cada estado de un elemento (relación o tipo de objeto) de una BD	Único en el universo del discurso	Único en una base de datos
Problema de transferencia de información frecuente	Problema de transferencia de información infrecuente	Problema de transferencia de información entre diferentes sistemas de BD
Asignado por el usuario de BD	Asignado por un asignador de IDOs	Asignado por el sistema

Visible para el usuario	Visible para el usuario	No visible para el usuario

Tabla 4.1: comparación entre clave, IDO y subrogado

El problema de la transferencia de información tiene lugar en distintas situaciones, como por ejemplo cuando se combinan varias bases de datos, en bases de datos federadas, etc. y se produce debido a que cada sistema tiene su propio esquema de identificación. Un tema muy relacionado con el problema de transferencia de información es el de la migración de objetos entre clases, ver Wieringa et al. (1995a) y Wieringa et al. (1995b). Si, tal y como se plantea en Wieringa y De Jonge (1995), se considera el IDO como un concepto de análisis que identifica a cada objeto independientemente de la clase a la que pertenezca y de la base de datos en la que se encuentre, el problema de transferencia de información no se dará para los IDOs.

Para Wieringa y De Jonge, el IDO es un concepto de análisis por lo que no lo asignan ni el usuario, ni el sistema, sino un asignador genérico al que estos autores denominan asignador de objetos.

En MIMO no habrá subrogados ya que se considera un concepto de implementación de cada sistema concreto. El IDO es, en MIMO, un concepto de análisis, construcción y explotación ya que identifica al objeto durante todo su período de vida. En análisis, los objetos tienen una existencia virtual por lo que su IDO también será virtual. Sólo se materializará en explotación, en el momento de creación de cada objeto. Sin embargo, MIMO no especifica como debería ser la implementación de dicho IDO, sólo las características que debe cumplir. Cada sistema concreto tendrá su propia representación de IDO a la que llamaremos subrogado. Un subrogado deberá tener una correspondencia uno-a-uno con el IDO al que representa, al igual que un IDO debe tener una correspondencia uno-a-uno con el objeto al que identifica. Sobre cómo soportar IDOs puede consultarse Bertino (1991).

Es evidente que aunque MIMO no especifique explícitamente condiciones para los subrogados, sí especifica algunas de modo implícito. Por ejemplo, MIMO soporta el concepto de ejemplificación múltiple (ver epígrafe 4.4) por lo que un objeto podrá pertenecer a varios tipos/clases simultáneamente. Por tanto, el IDO no puede representarse como el par (identificador de clase, identificador de objeto), ya que

entonces un mismo objeto en dos clases sería dos objetos distintos. Además, este tipo de representación del IDO, no permitiría la migración de objetos entre clases, ya que una migración implicaría la transformación de un objeto en otro.

En MIMO se mantienen claves primarias e IDOs y sus diferencias más notables son las siguientes:

- **Identificador de objeto**: es definido implícitamente (por el sistema o por un asignador de IDOs). Mantiene una relación uno-a-uno (inmutable y no reutilizable) con cada ente del universo del discurso, durante todo su período de vida.

- **Clave primaria**: es opcionalmente definida por el usuario y su valor puede ser modificado por éste en cualquier momento, por lo que no se garantiza la identidad de un objeto durante toda su vida. Los valores de la clave, a diferencia de los IDO, son reutilizables. Además, dos objetos diferentes que estuviesen en distintas clases podrían tener el mismo valor de clave primaria. Por tanto, en el caso de existir movilidad de objetos entre clases, la clave primaria no garantiza la identificación de éstos durante todo su periodo de vida. Piénsese, por ejemplo, en un estudiante (Pepe Pérez) cuya clave primaria viene determinada por su nombre y apellidos. Dicho estudiante deja de serlo para pasar a ser profesor. La clave primaria de los profesores está igualmente determinada por su nombre y apellidos y existe un profesor denominado Pepe Pérez. Para poder dar de alta al estudiante Pepe Pérez como profesor, es necesario modificar su valor de clave primaria (por ejemplo introduciendo su segundo apellido), por lo que, de no existir el IDO, se perdería su identidad. Igualmente, si un mismo objeto (por ejemplificación múltiple) pertenece a dos clases distintas (por ejemplo, es profesor y estudiante simultáneamente) y en cada clase tiene una clave primaria diferente (por ejemplo, el profesor se identifica por su DNI y el alumno por su número de matrícula), entonces, de no existir IDO, no sería posible reconocer que se trata del mismo objeto.

Muchos autores consideran que la clave primaria puede sustituir al IDO, a la vez que proporciona un valor semántico que el IDO no tiene. Esta es la tesis propuesta por los defensores de los sistemas de bases de datos relacionales extendidos, ver Stonebraker et al. (1990). En MEDEA se propone la posibilidad de sustitución del IDO por una clave primaria más una restricción de inmutabilidad: una clave primaria inmutable sería equivalente a un IDO. Sin embargo, nosotros no compartimos esta teoría y en MIMO se mantiene el concepto de clave primaria opcional y el de identificador de objeto obligatorio. El motivo es que, como ya se ha explicado, el valor de una clave primaria es único dentro del tipo/clase donde se definió, mientras que el IDO es único en toda la base de datos. Además el IDO es único para cualquier instante de tiempo mientras que la clave primaria sólo es única para los objetos que viven actualmente (no para los pasados ni futuros) ya que es reutilizable. Por tanto, una clave primaria, aún inmutable, no garantiza la identidad de un objeto en los distintos estados que podría atravesar durante su periodo de vida.

Definición 11: Servicio

*"Se llama **servicio** a una característica de un tipo/clase que, junto con sus otros servicios, definen el comportamiento de los ejemplares de dicho tipo/clase".*

Todo servicio se compone de una signatura (especificación formal del mismo) y de un cuerpo (implementación de dicho servicio) que no es visible.

El tipo define únicamente la signatura del servicio y la clase su implementación. De este modo, un mismo tipo podrá tener distintas implementaciones. El servicio es todo: signatura e implementación.

A nivel de análisis tendremos servicios (concretamente, signaturas de servicios), y éstos se implementarán, dependiendo del sistema, mediante funciones, procedimientos, etc. Por unificación nosotros hablaremos de servicios para referirnos al comportamiento de un objeto en cualquiera de sus fases.

Los servicios pueden llevar asociadas ciertas **restricciones** como: ***precondiciones***, predicados que tendrán que verificarse antes de iniciarse su ejecución, y ***postcondiciones***, predicados que tendrán que verificarse a su finalización. Igualmente, los servicios pueden llevar asociadas **condiciones de excepción** de tal modo que, cuando se verifique dicha condición, se realizará la acción o acciones definidas en la excepción.

Un servicio puede ser:

a) **Constructor** (crea objetos)

b) **Destructor** (destruye objetos)

c) **Actor o manipulador**, que se divide a su vez en:

c1) Selector u observador (accede a un objeto sin alterarlo)

c2) Modificadores o mutadores (cambian los valores que toman los atributos y/o las interrelaciones que mantiene)

d) **Copiador** (reproduce objetos)

e) **Migrador** (mueve un objeto de una clase a otra)

f) **Reconstructor** (recupera objetos previamente destruidos)

Definición 12: Estado de un objeto

Definimos estado de un objeto como *"el conjunto de valores que toman los atributos de un objeto en un momento dado y conjunto de interrelaciones que mantiene en ese mismo momento con otro u otros objetos".*

Un objeto puede variar de estado a lo largo de su vida y el IDO le permite mantener su identidad a pesar de que sus propiedades cambien.

Definición 13: Restricción de integridad

"Es un predicado o proposición que toma los valores verdadero o falso para uno o varios ejemplares".

Las restricciones se agrupan en Tipos de Restricción. *"Definimos Tipo de Restricción (TR) como una plantilla de predicado o proposición que toma los valores verdadero o falso para uno o varios ejemplares genéricos (es decir, que aún no se han especificado)"*. Cuando la plantilla del predicado se completa y se especifica para que ejemplar o conjunto de ejemplares debe verificarse, se obtiene una restricción.

Los TR pueden clasificarse en función de tres dimensiones:

a) Hablaremos de TR de **estado** cuando éstas se refieren a los valores que tienen los atributos, o a las interrelaciones existentes, un momento determinado; y hablaremos de TR **transición** cuando éstas se refieran al cambio que experimentan los valores o las interrelaciones.

b) Dependiendo de que el TR se aplique a los valores de un ejemplar o a más de un ejemplar, tendremos TR **simples o compuestas**.

c) Dependiendo al tipo de elemento del modelo al que se aplique tendremos, TR de **atributos**, de **servicios** (pre y postcondiciones), de **interrelaciones**[22] y de **tipo/clase** (invariantes de tipo/clase). Las invariantes pueden ser intratipo/intraclases, si afectan a ejemplares de un mismo tipo/clase (estas a su vez pueden ser simples o compuestas) o intertipo/interclase, si afectan a los ejemplares de varios tipos/clases. MIMO distingue, además de las pre y post-condiciones[23], los siguientes TR (que podrían clasificarse de acuerdo a cualquiera de los criterios anteriormente expuestos :

[22] Las restricciones aplicables a los tipos de interrelación se estudian en el epígrafe 4.4.

[23] Ver definición 11

- Restricción[24] de ***unicidad***.- dos ejemplares no pueden tener el mismo valor en el atributo, o los atributos, a los que se aplique.

- Restricción de ***no nulidad***.- el atributo al que se aplique tiene que tener un valor asignado. Si la restricción de no nulidad se define sobre un conjunto de atributos, esto significa que, al menos uno de ellos, deberá tener un valor asignado.

- Restricción de ***verificación***.- se define como una condición sobre el valor que puede tomar uno o varios atributos del tipo/clase en el que se define.

- Restricción de ***integridad referencial***.- implica que todo valor de la clave ajena debe tener correspondencia con un valor de la clave primaria, o alternativa, a la que dicha clave ajena hace referencia o ser nulo.

- ***Aserción***.- se define como una restricción de verificación que se define sobre dos o más tipos/clases.

Definición 14: Tipo de interrelación

Se define **tipo de interrelación** (TI) como una *"asociación que se establece, bien entre tipos (objeto o valor) bien entre clases".*

En el epígrafe 4.4 se realiza un estudio riguroso del sistema de interrelaciones de MIMO.

4.3. Caracterización del Sistema de Tipos

De los modelos estudiados, dos han sido los que han influido principalmente en el sistema de tipos de MIMO: son los modelos de los estándares SQL3 y ODMG-93. El motivo es que el sistema de tipos es una de las partes más elaborada de los modelos de

[24] Un TR hace referencia a un conjunto de restricciones similares. Una restricción es un ejemplar concreto de un TR. Así, por ejemplo, especificar que el DNI debe tener un valor único es una restricción del TR de unicidad. Por ello, en MIMO se distingue entre TR y restricción. Sin embargo, y por compatibilidad con a terminología habitual, en algunas ocasiones hablaremos de restricción en lugar de hacerlo de TR, pero siendo conscientes de que en realidad son conceptos distintos.

implementación, no así de los modelos que proceden de metodologías que se centran más en otros aspectos como el sistema de interrelaciones.

La necesidad de soportar estructuras cada vez más complejas ha llevado a una proliferación en los tipos de datos, lo que ha propiciado que en la actualidad tengamos una gran riqueza semántica. Aunque en muchos aspectos el sistema de tipos de los distintos modelos coincide, aún existen divergencias importantes entre ellos. Además, realizar una clasificación sistemática y rigurosa de ellos no es una tarea trivial, sobre todo si se tiene en cuenta que los criterios de clasificación utilizados en cada modelo son diferentes.

Es importante destacar que, como se puede apreciar en la definición de tipo de datos dada en el epígrafe 4.2 según la cual un **tipo de datos** es *"bien un tipo de objeto bien un tipo valor"*, MIMO mantiene la clásica diferencia entre objetos y valores. Esta diferencia se recoge tanto en SQL3, como en MEDEA. ODMG-93 es algo más confuso a este respecto ya que, a pesar de que para este estándares, en un principio, todo son objetos (*Denotable_Object*), finalmente han de recurrir a su separación, distinguiendo entre objetos mutables (*Object*) e inmutables (*Literal*).

La exposición del sistema de tipos la realizaremos según el siguiente esquema: en primer lugar (4.3.1) definimos cada uno de los tipos soportados por MIMO; a continuación (4.3.2), se definen los criterios de clasificación que permiten establecer la relación existente entre unos y otros tipos de datos, aplicando dichos criterios a los tipos de MIMO previamente definidos. A lo largo de la exposición se plantearán, en aquellos casos en que se consideren relevantes, las diferencias con SQL3 y ODMG-93.

4.3.1. Tipos de datos soportados en MIMO

MIMO define los siguientes tipos de datos:

Tipo Objeto (TO)

"Descripción de un conjunto de objetos que tienen las mismas características ".

Tipo Valor (TO)

"Descripción de un conjunto de valores que tienen las mismas características ".

Tipo básico

"Aquel que no se puede construir a partir de ningún otro". Se incluyen aquí los siguientes tipos de datos: carácter, numérico, entero, numérico flotante, bit, blob[25], día, mes, año, hora, minuto, segundo, intervalo e IDO.

Dominio

"Grupo nominado y homogéneo de valores de un cierto tipo de datos a los que se puede aplicar una restricción; esta restricción puede ser, bien la enumeración de un subconjunto de los valores del tipo sobre el que se define, bien un predicado que deben verificar todos los valores del tipo sobre el que se define".

Tipo Enumerado

"Grupo de valores simples que no se toman de ningún tipo básico, sino de una lista definida por el usuario". Se incluye aquí el tipo booleano.

Tipo Agregado

"Se llama tipo agregado a un tipo estructura, colección o vector".

Tipo estructura

"Tipo de datos compuesto por un número fijo de elementos que no tienen por qué ser del mismo tipo". Se incluyen aquí los tipos fecha, tiempo y fecha-tiempo.

Tipo Colección

"Tipo de datos con un único elemento el cual puede tomar un número variable de valores todos ellos del mismo tipo":

[25] BLOB, Binary Large OBject

Lista.- *"colección en la que los valores que la componen tienen un orden y pueden estar duplicados"*. Se incluyen aquí la tira de caracteres y la tira de bits.

Conjunto.- *"colección en la que los valores que la componen no tienen orden, ni pueden estar duplicados"*.

Multiconjunto.- *"colección en la que los valores que la componen no tienen orden y pueden estar duplicados"*.

Tipo Vector

"Tipo de datos compuesto por un número fijo de elementos todos ellos del mismo tipo".

De las definiciones dadas encontramos que hay dos aspectos que difieren en gran medida de las definiciones que habitualmente se encuentran en la literatura y cuya discusión se plantea a continuación:

- La primera es la desaparición del tradicional **Tipo Abstracto de Datos** (TAD[26]) que permite definir nuevos tipos valor que incorporan comportamiento, restricciones y que tienen capacidad para interrelacionarse. En el epígrafe 4.2 se expusieron los motivos que nos han llevado a la decisión de eliminar estos tipos de datos como tipos especiales. En realidad, nosotros consideramos que todo tipo de datos tiene un comportamiento y una implementación asociada. Por ejemplo, el tipo entero lleva asociadas las operaciones aritméticas. Del mismo modo, un TAD definido por el sistema (al igual que un TAD definido por el usuario), como por ejemplo un conjunto, también lleva operaciones asociadas cuya implementación no es visible para el usuario. Además dicha implementación (tanto para el tipo entero como para el tipo conjunto) podría variar de un sistema a otro por lo que cada tipo podría tener varias clases.

[26] Nos referiremos a TAD para hablar de lo que tradicionalmente se conoce como tipo abstracto de datos con el fin de evitar ambigüedades respecto a los tipos abstractos (tipos no ejemplificables) definidos en MIMO.

Consideramos que también los tipos valor (no TAD) pueden tener restricciones asociadas, aunque algunas de ellas sean impuestas por el propio sistema, por lo que tampoco ésta es una característica distintiva de los TAD.

El último punto es el de las interrelaciones. En nuestra opinión, también los tipos valor tienen capacidad para interrelacionarse. Así, por ejemplo, todos los tipos de datos numérico podrían interrelacionarse a través de una generalización, aunque ésta fuera también incorporada por el sistema y transparente para el usuario.

Esta aproximación es más cercana a la del ODMG-93 que a la de ningún otro modelo de los estudiados. En ODMG-93 se considera que todos los tipos (objetos o literales) tienen comportamiento y capacidad para interrelacionarse, aunque el comportamiento y las interrelaciones los defina el sistema. No obstante, no coincidimos con ODMG-93 en cuanto a las interrelaciones posibles entre literales, ya que el estándar considera que si el 3 es menor que el 5, estos dos literales mantienen una interrelación. En nuestra opinión se trata de una relación de orden entre los valores de un tipo, pero no de una interrelación tal y como en MIMO se define. En el apartado 4.4 se especifican los tipos de interrelación soportados en MIMO para los TV.

- El otro aspecto que consideramos importante discutir es la definición del **tipo vector,** ya que en MIMO no se incluye entre las colecciones como habitualmente se suele hacer, ver a título de ejemplo, Cattell (1994) y Gardarin y Valduriez (1992). Nosotros consideramos que un vector tiene algunas características de una colección pero no todas, y que también tiene características de una estructura. Si un vector es una colección, ¿cuál es la diferencia entre un vector y una lista?. Las dos diferencias que tradicionalmente se le atribuyen a estos dos tipos de datos son la posibilidad de indización y la longitud fija o variable. Hoy en día la mayor parte de los modelos permiten vectores cuya longitud puede variar en función de un parámetro (por ejemplo, ODMG-93 o C++) y listas indizables (ODMG-93 también las permite). Por tanto, si sólo se toman en consideración las diferencias expuestas, el vector y la

lista serían un mismo tipo de datos y sin embargo esto no es así. La diferencia, en nuestra opinión, es que un tipo colección tiene un único elemento el cual puede tomar un número variable de valores, todos ellos del mismo tipo, mientras que un tipo vector tiene un número fijo de elementos (al igual que una estructura) cada uno de los cuales puede tomar un único valor. Veamos un ejemplo:

Piénsese en una montaña rusa compuesta de diez coches en cada uno de los cuales viajará una persona. Para organizar la cola de espera se colocan unas barras paralelas a los lados de la taquilla. Pues bien, los diez coches corresponderían a un vector de personas donde cada coche es una posición del vector. Cada persona puede acceder directamente a un coche determinado (indización), y no es necesario que el coche uno esté ocupado para ocupar el dos. Aunque en principio hay diez coches (tamaño fijo), este número podría variar. La cola de espera es la lista. Las personas se colocan por orden de llegada y no hay límite en el número de personas que pueden esperar (tamaño variable). No tiene sentido plantearse que una persona pueda ocupar la tercera posición dejando libre la primera, lo que en un vector sí puede ocurrir (por lo que el concepto de indización en una lista no coincide con el de indización en un vector).

Planteado así, el vector tiene similitudes con una estructura, en la que también podría ocuparse la tercera posición manteniendo libre la segunda. Un vector podría verse como una estructura en la que todos sus elementos son del mismo tipo.

El vector podría dar lugar a otros tipos de datos como el vector circular (sería el caso de una noria) en el que no habría primer ni último elemento, de forma análoga la lista podría especializarse en una lista circular (un ejemplo, sería un grupo de niños jugando al corro). MIMO no introduce estos tipos de datos ya que ningún modelo de los estudiados los incorpora y pueden ser fácilmente simulados a partir de los tipos que proporciona.

En MIMO no se soportan matrices ya que una matriz siempre puede modelarse como un vector de vectores.

4.3.2. Clasificación de los tipos de datos soportados en MIMO

A la hora de establecer una clasificación rigurosa de los tipos de datos soportados por MIMO, es necesario, en primer lugar, definir los criterios de clasificación en los que nos vamos a apoyar.

4.3.2.1. Criterios de clasificación

Los criterios de clasificación de los tipos de datos varían ampliamente de unos modelos a otros:

- **SQL3** distingue entre tipos de datos **primitivos** y **abstractos,** DBL: MAD-003 (1996), según que éstos sean proporcionados por el sistema o definidos por el usuario. Los primeros sólo pueden tener valores primitivos, es decir, sin posible subdivisión lógica, mientras que los segundos pueden poseer valores primitivos o compuestos. Los tipos abstractos del SQL3 son siempre valores. Los objetos se soportan a través del tipo fila con nombre, DBL: MAD-004 (1996), que el estándar no clasifica, ni entre los tipos primitivos[27], ni entre los abstractos.

- **ODMG-93** establece una jerarquía de tipos en la que se distingue entre **objetos** y **literales**. Ambos, objetos y literales, tienen un supertipo común denominado **objeto_denotable**. En dicha jerarquía se incluyen todos los elementos del modelo (que en ODMG son siempre objetos denotables). Tanto objetos como literales pueden ser, a su vez, **atómicos** o **estructurados**. Los tipos que SQL3 llama primitivos (entero, carácter, fecha, etc.) aparecen en la jerarquía de ODMG como subtipos del tipo literal (literal atómico o estructurado). Existen

[27] Aunque, en nuestra opinión, tanto los tipos fila como las colecciones son tipos abstractos, el estándar no los incluye, explícitamente, como tales.

dos excepciones, los tipos string y bit_string que en ODMG se consideran colecciones.

En MIMO la clasificación se realiza atendiendo a tres dimensiones, las cuales no son totalmente ortogonales entre sí:

1. Según que la estructura (atómica o no) de los **elementos descritos**:

   1.1. Tipos de datos **simples**, si los elementos que describen -siempre valores- son atómicos.

   1.2. Tipos de datos **compuestos**, si los elementos que describen no son atómicos.

2. Según lo que es **objeto de descripción**:

   2.1. **Tipo de datos valor**.- *"descripción de un conjunto de valores que tienen las mismas características".*

   2.2. **Tipo de datos objeto**.- *"descripción de un conjunto de objetos que tienen las mismas características".*

3.- Según el **suministrador**:

   3.1.- tipos **primitivos** (prefabricados o predefinidos) que pueden ser suministrados bien por el sistema[28], bien por el administrador.

   3.2.- tipos **específicos**, si son definidos por el usuario.

Para construir tipos específicos se definen los tipos generadores. Un **tipo generador** *"es un tipo, no ejemplificable directamente, que permite construir distintos tipos específicos de datos"*. Un tipo generador puede ser primitivo o específico. Entre

[28] Entendido sistema en un sentido genérico, no necesariamente, un sistema informático. Así, por ejemplo MIMO proporciona un conjunto de tipos primitivos.

los primitivos[29] se incluyen, el generador de dominios, el de enumerados, el de estructuras, el de colecciones, el de TO y el de tipos plantilla. Un **generador de tipo plantilla** permite obtener **plantillas** de tipos para la generación, por ejemplificación de dichas plantillas, de nuevos tipos. Los tipos plantilla que se obtienen mediante un generador de tipo plantilla son generadores de tipo específicos. Los tipos plantilla proporcionados por el sistema (por ejemplo el generador de conjuntos) son tipos plantilla primitivos que se comportan, a su vez, como generadores de nuevos tipos por ejemplificación. Por ejemplo, un generador de tipos plantilla permitiría la generación de un tipo pila genérico (es decir una pila de cualquier tipo de datos). Este tipo pila es un generador de tipo específico. Al ejemplificar el tipo sobre el que se define la pila generaremos nuevos tipos (pila de caracteres, de enteros, etc.).

4.3.2.2. Clasificación del sistema de tipos de MIMO

En la tabla 4.2 se clasifica el sistema de tipos[30] de MIMO en base a los distintos criterios expuestos, especificando a la vez si el tipo tiene extensión virtual o real (ver epígrafe 4.2):

Tipo de datos	Atomicidad	Objeto de descripción	Suministrador	Extensión
Básico	Simple	Valor	Primitivo	virtual
Dominio	Simple	Valor	Específico	virtual/real
Enumerado	Simple	Valor	Primitivo[31]/Específico	real
Agregado	Compuesto	Valor	Primitivo/específico	virtual
TO	Compuesto	Objeto	Específico	virtual

Tabla 4.2: clasificación de los tipos de datos

[29] En general, se habla de que un sistema soporta tipos enumerados, dominios, etc., pero lo que en realidad proporciona el sistema es un generador de dominios, un generador de enumerados etc. a partir de los cuales el usuario puede obtener sus propios tipos específicos.

[30] Obsérvese que en la clasificación, los tipos enumerado, dominio, agregado y TO, a diferencia que en otras clasificaciones, hacen referencia a los generados y no a los generadores.

[31] El tipo booleano es un tipo enumerado incorporado y, por tanto, primitivo. El resto de los tipos enumerados, al igual que los dominios y la mayor parte de los agregados son generados por el usuario. Lo que realmente proporciona el sistema, como ya se ha indicado, son los constructores necesarios para su generación.

La clasificación que se muestra corresponde al sistema de tipos de MIMO, por lo que podría variar en otros sistemas. Así, por ejemplo, en MIMO lo dominios son específicos, pero esto no es una restricción del tipo dominio sino del modelo que no incorpora ningún dominio predefinido.

En cuanto a la extensión, diremos que un dominio tiene extensión virtual cuando se define por intensión y real cuando se define por extensión. El TO tiene extensión virtual; tendrá extensión real la clase que lo implemente y tan sólo en tiempo de explotación. Todo tipo con extensión real es ejemplificable, directa o indirectamente; un tipo con extensión virtual no es ejemplificable.

En la tabla 4.3 se muestran las diferencias existentes entre los distintos tipos agregados:

	Orden	Acceso	Nº de elementos	Homogeneidad del tipo	Duplicación
Estructura	No aplicable	Directo	Fijo	No	No aplicable
Vector	Aplicable según llegada Aplicable según valor	Directo Secuencial	Fijo	Sí	Sí
Conjunto	No	Secuencial	Variable	Sí	No
Multiconjunto	No	Secuencial	Variable	Sí	Sí
Lista	Implícito según llegada Aplicable según valor	Secuencial Aplicable directo	Variable	Sí	Sí

Tabla 4.3: comparación entre los distintos tipos agregados

El orden en función del valor de sus elementos[32], no es implícito ni al tipo vector ni al tipo lista. Aunque se puedan obtener tanto vectores como listas ordenadas, será necesario mantener dicho orden, de un modo explícito, cada vez que se inserta o se borra un elemento de una lista o de un vector. Sin embargo, el orden de llegada, es implícito en una lista ya que cuando un elemento llega a la lista siempre se coloca en la última posición, y aplicable en un vector ya que, aunque a un vector se le puede imponer la restricción de que los elementos se coloquen por orden de llegada, tal restricción no es característica de este tipo de datos.

El acceso a los elementos de un vector puede ser secuencial o directo; sin embargo, en una lista se accede a sus elementos de modo secuencial, aunque también sería posible indizarla a fin de poder acceder de modo directo, esta característica no es inherente al tipo de datos lista[33].

[32] Aunque, por simplificar la terminología, hablemos de elementos de una lista hay que subrayar que en realidad se trata de cada uno de los valores del elemento (que hemos dicho que es único) de la lista.

[33] Aunque en nuestra opinión tampoco es una característica distintiva del tipo lista respecto al tipo vector ya que una lista podría ser o no indizada. Esta distinción se hace por una cuestión histórica, ya que los sistemas y lenguajes de programación tradicionales no implementaban listas indizadas mientras que los actuales sí. Esto es debido a que conceptualmente una lista sigue siendo una lista independientemente de que sus elementos sean o no accesibles de un modo directo.

4.4. Caracterización del Sistema de Interrelaciones

4.4.1. Introducción

Un aspecto fundamental y característico de los modelos de datos es su sistema de interrelaciones ya que éstas permiten recoger una parte importante de la semántica del universo del discurso. No todos los modelos proporcionan la misma capacidad semántica. Así, por ejemplo, SQL3 y ODMG mantienen bastantes limitaciones en éste sentido, por lo que restringir las interrelaciones a las soportadas por dichos estándares reduciría, en gran medida, el poder expresivo de MIMO. Para definir un sistema de interrelaciones con mayor riqueza semántica se han tomado como base, principalmente, MEDEA y el MU.

Es necesario, a fin de mantener su capacidad expresiva, que los modelos de datos ofrezcan la suficiente variedad de tipos de interrelación. Sin embargo, soportar un número elevado de interrelaciones puede llevar a un modelo excesivamente complicado. Es conveniente establecer un equilibrio entre la semántica y la complejidad del modelo. En MIMO se define un conjunto básico de interrelaciones primitivas que pueden combinarse con restricciones a fin de obtener una mayor capacidad semántica. De esta forma, en MIMO se mantiene la flexibilidad y la potencia semántica sin aumentar excesivamente su complejidad.

A continuación se presenta el Sistema de Interrelaciones (SI) de MIMO según el siguiente esquema: en primer lugar se plantea la discusión y definición aceptada para cada uno de los tipos de interrelaciones primitivas (4.4.2); posteriormente se exponen los tipos de restricción soportados (4.4.3); para finalizar se proponen nuevos tipos de interrelación que se obtienen aplicando las restricciones a las interrelaciones primitivas, a la vez que se discute el SI de MIMO en relación a los SIs de los modelos considerados (4.4.4).

4.4.2. Tipos de interrelaciones primitivas

4.4.2.1. Clasificación

Se define ***tipo de interrelación (TI)***, como una *"asociación que se establece bien entre tipos (objeto o valor) bien entre clases".*

MIMO divide los TI en niveladas y jerárquicas según que los elementos participantes mantengan la misma categoría en la interrelación, o exista entre ellos una subordinación. Entre estas últimas se incluyen las generalizaciones y meronimias.

Tipo de interrelación nivelada

"Es un TI de igual a igual que tienen lugar entre TO". Pueden establecerse entre dos (*binarias*) o más (*n-arias*) TO. Un caso especial es el TI *reflexiva* en la que los TO participantes son el mismo TO.

*"Denominamos **Clase de Interrelación (CI)** a la construcción*[34] *de un TI nivelada"*.

Recordamos que uno de los objetivos de MIMO es proporcionar una gran riqueza semántica tanto a nivel de análisis como a nivel de construcción. Por ello, los TI que se establecen entre TO, se convierten, en construcción, en clases de interrelación. Estas clases permiten asociar niveladamente dos clases (que serán la construcción de dos TO):

> Dados dos tipos de objeto TO1 y TO2 y un tipo de interrelación nivelada entre ellos TI1 y dos clases C1 y C2, donde C1 es la construcción de TO1 y C2 es la construcción de TO2, entonces existe necesariamente una asociación entre C1 y C2 denominada CI1 y, además, CI1 es la construcción de TI1.

Los valores no tienen capacidad de interrelacionarse niveladamente.

[34] Diseño e implementación.

*"Se llama **interrelación nivelada** a cada uno de los ejemplares de una CI"*. Cada interrelación asocia ejemplares concretos de cada clase participante en la CI. El conjunto de ejemplares de una CI es un subconjunto de los ejemplares del TI que construye[35].

Tipo de interrelación jerarquizadas

Distinguimos dos casos:

a) TI de Generalización

"Es un TI jerárquica, bien entre tipos (valor u objeto) bien entre clases[36]*, que representa una clasificación en la que los subtipos/subclases son una especialización de cada uno de sus supertipos/superclases (herencia simple y múltiple). En una jerarquía de generalización cada subtipo/subclase hereda todas las características (no se admite herencia inhibida) de sus supertipos/superclases, además de poder añadir características propias (herencia de especialización) o redefinir las heredadas"*.

La generalización, a diferencia de los TI nivelados, no tiene ejemplares.

a.1) TI de múltiple generalización

"Es una agregación de generalizaciones de un mismo tipo de objeto, o clase[37]*, que se especializa simultáneamente por diversos criterios"*.

b) Meronimia

"Es un TI que corresponden al concepto PARTE-DE y que se establecen entre tipos (valor u objeto) o entre clases[38]*, llamándose holónimo (todo) al tipo o*

[35] Es posible hablar de ejemplares de los TI ya que, como veremos, un TI nivelada es un subtipo de un TO por lo que, además, diremos que estos ejemplares son siempre objetos.
[36] En realidad, entre clases existen generalizaciones, no TI de generalización, ya que los tipos sólo tienen existencia en análisis. Sin embargo, por simplicidad, definimos globalmente los TI de generalización y las generalizaciones.
[37] En este caso se denomina *múltiple generalización.*
[38] Análogamente a lo que ocurre con las generalizaciones, entre clases existen meronimias, no TI de meronimia. Por el mismo motivo, definimos globalmente los TI de meronimia y las meronimias.

clase compuesta y merónimo (parte) a cada uno de los tipos o clases componentes".

La interrelación en una meronimia, es decir los ejemplares de ésta, siempre coincide con el objeto (o valor) compuesto. Cada interrelación asocia ejemplares concretos del holónimo con ejemplares de sus merónimos de modo, que el ejemplar de la meronimia es el propio holónimo.

En la literatura pueden encontrarse distintas clasificaciones de meronimia como las propuestas en Winston et. al (1987), en Kim et al. (1989) o en Pastor y Ramos (1995). MIMO distingue, al igual que MEDEA, los siguientes tipos de meronimia:

Compuesto/componente.- Los componentes realizan una función concreta en relación a otros componentes o al todo (por ejemplo, la rueda y el coche).

Miembro/colección.- Cuando no existe dicha relación funcional entre el todo y las partes (por ejemplo, un árbol y el bosque al que pertenece).

Además, dependiendo de que la agregación sea física o lógica, cada una de las anteriores podrá ser:

Física.- Una "parte" no puede pertenecer a más de un "todo".

Lógica (o de catálogo).- Una "parte" puede encontrarse en diferentes "todos"; la parte funciona como un modelo de un catálogo.

Esta última distinción entre agregación física y de catálogo es la que soporta el MU. Sin embargo, el MU considera que existe meronimia física cuando los merónimos participan con cardinalidad máxima uno, y de catálogo en caso contrario. En MIMO la cardinalidad no implica el tipo de meronimia; si la meronimia es física, la cardinalidad máxima habrá de ser uno, pero lo contrario no es cierto ya que podría darse el caso de una meronimia de catálogo con cardinalidad máxima uno.

En la tabla 4.4 se muestra la posibilidad de interrelación entre los distintos componentes del MIMO:

Componentes / Interrelaciones	**Tipos objeto**	**Tipos valor**	**Clase**	**Objetos**	**Valores**
TI nivelada	Aplicable	No aplicable	Aplicable[39]	_	_
TI de generalización	Aplicable	Aplicable	Aplicable[40]	_	_
TI de meronimia	Aplicable	Aplicable	Aplicable[41]	_	_
I. Nivelada	_	_	_	Aplicable	No aplicable
I. de Generalización	_	_	_	No aplicable	No aplicable
I. de Meronimia	_	_	_	Aplicable	Aplicable

Tabla 4.4: posibilidades de interrelación de los componentes de MIMO

Un TI de generalización se establece entre TOs (un estudiante es un tipo especial de persona), entre TV (una pila es un tipo especial de lista) y entre clases (la implementación de un estudiante es una especialización de la implementación de una persona). Sin embargo, esta especie de TI no tiene ejemplares ya que no tiene sentido una interrelación de generalización entre objetos o entre valores (no se puede decir, por ejemplo, que el objeto "Pepe", que es un estudiante, sea un tipo especial de objeto "Pepe", que es una persona, ya que ambos son el mismo objeto).

Una interrelación nivelada, por el contrario, se establece entre objetos (y el TI nivelada entre TO), ya que permite establecer asociaciones entre ejemplares concretos de TOs (por ejemplo, la interrelación nivelada "compra" asocia a un *cliente* con un *producto*; sin embargo, la interrelación de generalización por la que "un estudiante es una persona", se establece entre los TO, *estudiante* y *persona*).

Un TI nivelada se establece entre TO y un TI de generalización se establece entre TO, o TV. Un TI nivelada es ejemplificable (el ejemplar es la interrelación en sí misma) y la interrelación se establece entre objetos. Por el contrario, un TI de generalización no es ejemplificable, por lo que no existe el concepto de interrelación de generalización.

El TI meronímica es un tipo de interrelación que tiene características comunes a las niveladas y a las generalizaciones. El TI meronímica se establece entre TO o entre

[39] Aunque el TI no tiene sentido entre clases (ya que el tipo es un concepto de análisis), en construcción existen CI que soportan este mismo concepto. Por ello, podemos considerar que las clases pueden interrelacionarse niveladamente en MIMO.

[40] Estrictamente, generalización y no TI de generalización.

[41] Estrictamente, meronimia y no TI de meronimia.

TV, y la interrelación se establece entre objetos o valores. Una interrelación meronímica es, al igual que una interrelación nivelada, un ejemplar del TI pero, en este caso, el ejemplar de la interrelación coincide con el ejemplar del objeto o valor compuesto.

Los modelos tradicionales de implementación, SQL3 u ODMG-93 entre otros, implementan, los TI nivelada mediante atributos de referencia, los TI de meronimia a través de atributos de tipo estructura (meronimias compuesto/componente) o de tipo colección (meronimias miembro/colección) y los TI de generalización mediante características del tipo. En MIMO, sin embargo, se mantienen los conceptos de tipo[42] de interrelación nivelada, de meronimia y de generalización, tanto en análisis como en construcción. El motivo es el de proporcionar un modelo que permita mantener a nivel de construcción la mayor parte posible de la semántica recogida en análisis.

El TI nivelada entre clases se recoge en construcción a través de lo que hemos llamado clase de interrelación. El TI de meronimia se recoge en construcción mediante el concepto de meronimia entre clases y el TI de generalización mediante el concepto de generalización entre clases.

En la tabla 4.5 se muestra un resumen de los constructores de MIMO en cada los niveles de análisis y construcción:

[42] Se mantiene el concepto, aunque como veremos a continuación, con otro nombre, ya que el tipo en MIMO es una noción de análisis.

Análisis	Construcción
TV	Clase
TO	Clase
TI nivelada	Clase de interrelación
TI de generalización	Generalización entre clases
TI meronímica	Meronimia entre clases
TO abstracto	Clase abstracta
-	Clase diferida
Tipo de restricción	Restricción
Atributo derivado	Atributo derivado real o virtual

Tabla 4.5. Constructores de MIMO en los niveles de análisis y construcción

4.4.2.2. Tipos de Interrelación entre Tipos de Interrelación

La posibilidad de establecer interrelaciones entre TI, soportada ya en algunos en el año 85 por modelos como el KL-ONE, Brachman y Schmolze (1985), no es sin embargo contemplada, como tal, en casi ninguno de los modelos estudiados. Solamente OOM soporta este concepto, aunque de un modo más restrictivo que MIMO. En MIMO se planteó la posibilidad de recoger este especie de TI ya que, si un TI puede tener atributos, operaciones y restricciones propias, ¿porqué no podría tener también sus propios interrelaciones? En un primer momento se puede pensar que esta posibilidad complica innecesariamente el modelo, cuando en su lugar podrían utilizarse otras alternativas de modelado como la conversión de los TI en TO, añadir a los TI alguna restricción, etc. Sin embargo, aunque las alternativas de modelado planteadas puedan ser admisibles, e incluso recomendables en algunas situaciones, nos parece importante que esta decisión pueda ser tomada por el usuario y no impuesta por un modelo excesivamente restrictivo. Pensamos que un modelo, especialmente a nivel de análisis, debe proporcionar la mayor riqueza semántica posible; será decisión del diseñador si la utiliza o no. Además, tampoco es cierto que los TI entre TI compliquen el modelo sino que, por el contrario y tal como se muestra en Marcos et al. (1996b), permiten modelar un mismo universo del discurso de un modo más simple que en modelos que no soportan esta posibilidad.

El MU propone una solución alternativa: los TO asociados (“association class”). Es un TO que se asocia a un TI, permitiendo que ésta tenga, a través del TO asociado, atributos, restricciones e interrelaciones propias. Sin embargo, asociar un TO a un TI con el único fin de poder relacionar dicho TI con otros, nos parece artificial, poco intuitivo y confuso, ya que tal tipo de interrelación podría, en algunos casos, entenderse erróneamente como una interrelación ternaria. El planteamiento del MU, realmente, enmascara una interrelación entre un TO y un TI.

A continuación exponemos, a través de ejemplos, los distintos TI entre TI soportados por MIMO:

a) TI nivelada entre TI:

Supongamos una agencia de viajes que compra paquetes de reservas hoteleras con descuentos especiales para parejas. Dichas reservas se realizan con anterioridad a que ningún cliente las haya solicitado. Se establece una interrelación entre las agencias y los hoteles. Posteriormente una determinada pareja, constituida por una interrelación nivelada entre dos personas, puede comprar una de estas reservas estableciéndose así una interrelación entre la pareja y la reserva.

En los modelos tradicionales, carentes de este TI, este requisito de usuario se modelaría obligando a que el contrato lo realizara un solo componente de la pareja. Aunque esta solución pueda parecer la más correcta, lo que en realidad ocurre es que es la solución a la que estamos acostumbrados. Existen ejemplos en que quizá no lo veamos tan lógico: piense, por ejemplo, en una declaración de la renta conjunta; ésta debe realizarla el matrimonio. Otra solución habitual a este tipo de problemas, es la de transformar pareja y reserva en TO y establecer una interrelación entre ellos, solución que, en nuestra opinión, no es acertada ya que se fuerza artificialmente la conversión de un TI en un TO.

b) TI entre un TI y un TO:

Un ejemplo válido es el de la declaración de la renta. Un matrimonio, establecido por la interrelación de dos personas, puede realizar su declaración de forma individual, a través de una interrelación entre persona y declaración, o de forma conjunta, a través de una interrelación entre matrimonio (interrelación) y declaración (objeto).

c) Generalizaciones de TI:

El mismo ejemplo del matrimonio daría lugar a una generalización de un TI si se deseara modelar por separado la información relativa a los matrimonios civiles y a los eclesiásticos. “Matrimonio Civil” y “Matrimonio eclesiástico” serían dos subtipos del TI “Matrimonio”.

d) Meronimia entre TI:

Esta especie de meronimia es similar a la introducida por Witson et al. (1987) como meronimia entre actividades. Por ejemplo, un empleado “Dirige” un proyecto y esta dirección consta de unas labores de gestión y unas labores de planificación, por lo que el empleado “Gestiona” y “Planifica” el proyecto. “Gestiona” y “Planifica” son merónimos de “Dirige”. La meronimia entre actividades no es el único ejemplo de TI de meronimia entre TI que puede encontrarse.

En OOM se recogen los tres primeros TI entre TI, no así el TI meronímico (OOM tampoco soporta directamente la meronimia entre clases; este concepto lo recoge a través de atributos, al igual que los modelos de implementación). OOM, en nuestra opinión, introduce algunas características de orientación al objeto, pero es más un modelo E/R extendido que un modelo de objetos. En MIMO, además de introducir el TI meronímica entre TI, la concepción de los TI entre TI difiere sustancialmente de la de OOM ya que, tal y como veremos a continuación, en MIMO un TI nivelada es un subtipo del TO

El TI nivelada como un subtipo del TO

Un TI nivelada puede tener características propias (atributos, restricciones y servicios). Además, en MIMO se ha llegado a que, igualmente, pueden tener interrelaciones propias. ¿Cuál es, pues, la diferencia existente entre un TI nivelada y un TO? En nuestra opinión, la única diferencia semántica es que un TI nivelada asocia dos o más TO, y que para que exista la interrelación deben existir los correspondientes objetos. En MIMO se considera, por tanto, que un *"**TI nivelada** es un TO que asocia a dos o más TO y en el que la existencia de cada ejemplar depende de la existencia de los ejemplares de los TO que asocia",* (al igual que un TO que dependa en existencia de otro[43]). En la figura 4.3 se muestra la relación entre TO y TI en MIMO:

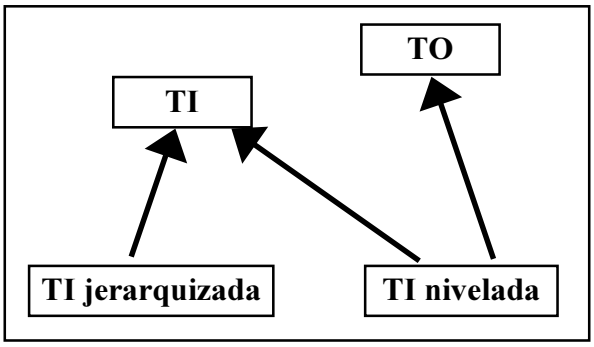


Figura 4.3.- Relación entre TO y TI en MIMO

Hay que tener en cuenta que un ejemplar de un TI tendrá ahora un IDO que le identifica unívocamente y lo distingue, por tanto, del resto de los ejemplares, ya que todo lo dicho para los TO (interrelaciones, restricciones, etc.) es aplicable a los TI nivelados. De éste modo, no se aumenta excesivamente la complejidad del modelo. La dificultad mayor puede consistir en la adaptación al nuevo estilo de modelado, Marcos et al. (1996b).

Como ya hemos dicho anteriormente, en MIMO es posible combinar los TI primitivos con un conjunto de Tipos de Restricción aplicables a dichos TI a fin de obtener una mayor capacidad semántica.

[43] Lo que en el modelo E/R se conoce como entidad débil

4.4.3. Tipos de restricciones aplicables a los TI primitivas

Como ya se ha señalado anteriormente, MIMO define un conjunto de tipos de restricción, que pueden ser combinados con los TI primitivos a fin de obtener nuevos TI. Tales tipos de restricción son los siguientes:

Restricción[44] de dependencia en existencia

"Se dice que el tipo T1 depende en existencia del tipo T2, cuando la existencia de todo ejemplar de T1 está condicionada a la existencia de un determinado ejemplar de T2, de forma que si desaparece el ejemplar de T2, el ejemplar (o ejemplares) de T1 asociado(s) desaparece(n) con él".

Esta especie de restricción es implícita en los TI nivelada con respecto a los TO que interrelacionan. En las meronimias puede tener dos significados: que un determinado merónimo dependa en existencia del holónimo o que el holónimo dependa en existencia de alguno, o de la totalidad (es decir, de la interrelación), de los merónimos.

En nuestra opinión, este doble sentido que puede tener la dependencia en existencia en una meronimia, es igualmente aplicable a las generalizaciones. Sin embargo, es necesario reflexionar sobre ello ya que, implícitamente, en una generalización la dependencia en existencia es siempre del subtipo respecto del supertipo. Nosotros pensamos que la dependencia en existencia en una generalización es bidireccional. Supongamos la generalización *artículo ES-UN documento*. Es cierto que la eliminación de un documento implica la eliminación de un artículo (en el caso de que el documento a borrar sea un artículo), pero también es igualmente cierto que la eliminación de un artículo implica la desaparición de un documento. En el caso de que la generalización tenga varios subtipos, el supertipo dependerá en existencia de todos sus subtipos simultáneamente.

Restricción de dependencia en identificación[45]

[44] De nuevo, por similitud con la terminología habitual, hablaremos de restricciones en lugar de hablar de TR.

"Se dice que el tipo T1 depende en identificación del tipo T2, cuando los ejemplares de T1 no se identifican por sí mismos, sino que necesitan un ejemplar de T2 para poder identificarse".

Toda dependencia en identificación implica una dependencia en existencia, si bien lo contrario no es cierto.

La dependencia en identificación es aplicable entre dos tipos asociados a través de un TI (sea ésta nivelada o jerárquica), siempre que ambos tipos tengan clave primaria[46]. En las meronimias, si existe dependencia en identificación (ya que si no tienen clave primaria este tipo de restricción no puede darse), es siempre el merónimo el que depende en identificación del holónimo.

En las generalizaciones, cuando el supertipo tiene clave primaria, la dependencia en identificación es implícita[47] y, al igual que ocurre con la dependencia en existencia, bidireccional.

Restricción de dependencia en identidad

"Se dice que el TO1 depende en identidad del TO2, cuando el IDO de todo ejemplar de TO1 es igual que el IDO de algún ejemplar de TO2".

[45] El término *identificación* hace referencia al hecho de reconocer si dos ejemplares son idénticos, por lo que podría pensarse que no es el término más apropiado para esta tipo de restricción. Sin embargo, se mantiene por continuidad con la dependencia en identificación de algunas extensiones del modelo E/R.

[46] La dependencia en identificación podría aplicarse en relación a una clave primaria o alternativa. Hablaríamos entonces de dependencia en identificación fuerte o débil. Sin embargo, no hemos incluido esta distinción porque ningún modelo de los estudiados la soporta y pensamos que sus beneficios no compensan el aumento de complejidad que supone. Queda la idea para posteriores estudios sobre el tema.

[47] Aunque no siempre el supertipo tiene clave primaria, de tenerla, la dependencia en identificación es implícita. Por ello, en la tabla 4.5 hemos preferido mantener implícita (en la dependencia en identificación aplicada a las generalizaciones), ya que no es el usuario quien decide aplicar o no este tipo de restricción.

La dependencia en identidad es implícita en las generalizaciones y, al igual que ocurre con la dependencia en existencia y en identificación, la dependencia es bidireccional.

Restricción de exclusividad

"*Sea T1 un tipo que participa en dos tipos de interrelación, TI1 y TI2; se dice que existe una restricción de exclusividad si cada ejemplar de T1 puede participar en TI1 o en TI2, pero nunca en ambas a la vez*".

La exclusividad es aplicable a todos los TI. Además, MIMO permite especificar, en un TI de generalización (o de meronimia compuesto/componente), los subtipos a los que se aplica la exclusividad[48].

Restricción de exclusión

"Sean dos TI niveladas, TI1 y TI2, establecidas entre TO1 y TO2. Se dice que entre TI1 y TI2 existe una restricción de exclusión si para toda combinación[49] de interrelaciones de TI1 entre objetos de TO1 y TO2, dicha combinación no existe entre las interrelaciones de TI2".

Este tipo de restricción no es aplicable, ni a generalizaciones ni a meronimias.

Restricción de inclusión

"Sean dos TI niveladas, TI1 y TI2, establecidas entre TO1 y TO2. Se dice que TI1 tiene una restricción de inclusión respecto de TI2 si para toda combinación[50] de

[48] Así, por ejemplo, podría existir una restricción de exclusividad entre los subtipos profesor y becario (subtipos de persona), pero no existir tal restricción entre alumno y becario o entre alumno y profesor. En las extensiones del modelo E/R que permiten este tipo de restricción, sólo es posible aplicarla a todos los subtipos simultáneamente.

[49] Nótese que hablamos de combinación de interrelación en lugar de interrelación, ya que un ejemplar de un tipo de interrelación, al ser la interrelación un objeto en sí misma, tiene identidad propia por lo que no podrían existir dos idénticas, aunque fueran iguales.

[50] Al igual que en la restricción de exclusión, y por el mismo motivo, hablamos de combinación de interrelación en lugar de interrelación.

interrelaciones de TI1 entre objetos de TO1 y TO2, dicha combinación debe existir necesariamente entre las interrelaciones de TI2".

Este tipo de restricción no es aplicable, ni a generalizaciones ni a meronimias.

Restricción de orden

"Sea TI1 un TI que se establece entre dos tipos, T1 y T2. Se dice que existe una restricción de orden de T2 respecto de T1 cuando los ejemplares de T1 participan en TI1 en un orden concreto, que se determina en función de uno o varios atributos de T2".

No es aplicable a TI de generalización, ni a TI de meronimia compuesto/componente.

Restricción de cardinalidad

"Se define como el número mínimo y máximo de veces en las que cada ejemplar de un tipo está asociado con un ejemplar del otro tipo en una interrelación".

La restricción de cardinalidad se define para cada tipo participante en la interrelación. Es aplicable a cualquier tipo de interrelación.

En la tabla 4.6 aparecen las posibles combinaciones entre las restricciones y los TI primitivos de MIMO:

Restricciones / Interrelaciones		**D. en Existenc.**	**D. en Identific.**	**D. en Identidad**	**Exclusiv.**	**Exclusión**	**Inclusión**	**Orden**	**Cardin.**
Nivel.	**TO**	Aplicable	Aplicable	No aplicable	Aplicable	Aplicable	Aplicable	Aplicable	Aplicable

Gener.	**TO**	Implícita	Implícita	Implícita	Aplicable	No aplicable	No aplicable	No aplicable	Aplicable
	TV	Implícita	Implícita	No Aplicable	Aplicable	No aplicable	No aplicable	No aplicable	Aplicable
Meron.	**TO**	Aplicable	Aplicable	No aplicable	Aplicable	No aplicable	No aplicable	Aplicable m./colecc.	Aplicable
	TV	Aplicable	Aplicable	No Aplicable	Aplicable	No aplicable	No aplicable	Aplicable m./colecc.	Aplicable

Tabla 4.6: Posibles combinaciones entre TI y tipos de restricción

La restricción de exclusividad entre TI niveladas se convierte, debido a la naturaleza de esta especie de interrelaciones, en una restricción intratipo/clase, ya que si un TI es un TO, la restricción de exclusividad pasa a ser una restricción entre tres TO, dos de los cuales son TI. Lo mismo ocurre con las restricciones de inclusión y exclusión, sólo aplicables a TI niveladas, y por tanto, TO.

Siempre que un TI se vea afectado por alguna restricción, dicha restricción se mantendrá en la clase que implemente al tipo (si se trata de un TI nivelada) o en la meronimia o generalización (si se trata de un TI de meronimia o de generalización).

Es importante destacar que el significado de la aplicación de TR a TI varía según la naturaleza del TI al que se apliquen. Así, por ejemplo, la restricción de exclusividad, aplicada a un TI de generalización, implica que si un ejemplar del supertipo pertenece a un subtipo, no podrá pertenecer a ningún otro; sin embargo, esta misma restricción, aplicada a dos TI nivelada implica que cada ejemplar del TO1 participa en el TI1 o en el TI2, pero no en ambos a la vez.

La aplicabilidad de la restricción de cardinalidad a un TI de generalización, convierte este TI en una múltiple generalización. Implícitamente, si no se especifica restricción, los subtipos participan con cardinalidad mínima cero y máxima uno (generalización); si se especifica la cardinalidad de los subtipos, tanto de mínima como de máxima uno, entonces tenemos una múltiple generalización (coincide con el concepto de and-generalization del MU, tal y como veremos en el epígrafe siguiente).

4.4.4. Interrelaciones en MIMO *vs.* interrelaciones en el MU y MEDEA

Tanto el MU como MEDEA incluyen algunos tipos de interrelación que se soportan en MIMO añadiendo restricciones a los TI primitivos. Así, por ejemplo, la dependencia en identificación de MEDEA, se modela en MIMO mediante una ***restricción de dependencia en identificación***. Esta decisión se debe a que tal tipo de dependencia no es exclusiva de los TI niveladas (como ocurre en MEDEA), sino que puede aparecer también en meronimias, siendo implícita en toda generalización.

Otro tipo de restricción es la de ***exclusividad*** que junto con una interrelación nivelada da lugar a una interrelación-or ("Or-Association") del MU (por ejemplo, un empleado de una empresa puede impartir cursos o recibirlos, pero no ambas cosas). MIMO, a diferencia del MU, no considera la interrelación-or como un TI primitivo ya que puede obtenerse por combinación de un TI y una restricción. Además, la restricción de exclusividad puede igualmente aplicarse a generalizaciones y meronimias.

La restricción de ***exclusión***, de los modelos estudiados, tan solo se soporta en el modelo de objetos de MERISE (sin embargo, MERISE no soporta la restricción de exclusividad entre interrelaciones niveladas). Esta restricción, aplicada al mismo ejemplo de los cursos, implica que un empleado no puede impartir y recibir un mismo curso. La restricción de exclusividad implica la de exclusión, pero lo contrario no es cierto.

El MU soporta, al igual que MIMO, la restricción de ***inclusión*** (Subset constraint).

Respecto a la ***generalización***, es necesario hacer un estudio más riguroso, ya que MEDEA y el MU[51] realizan clasificaciones diferentes aportando una semántica distinta.

[51] La discusión que aquí se presenta está basada en el MU V0.8. Como ya se ha dicho, los tipos de generalización permitidos han variado en UML V0.91 y UML V1.0, sin embargo, cuando estas dos versiones aparecieron, septiembre del 96 y enero del 97 respectivamente, esta tesis ya estaba en sus fase final. Es importante destacar que algunas de las críticas que nosotros planteábamos al MU, y cuya discusión mantenemos en este trabajo, han sido subsanadas en sus últimas versiones.

Así, por ejemplo, en MEDEA se propone una clasificación de la generalización, al igual que otras metodologías como OOM, en dos dimensiones ortogonales:

- *Solapada/exclusiva*.- según que los subtipos se solapen o sean disjuntos.
- *Total/parcial*.- según que la unión de los subtipos recubra o no el supertipo.

Estas dos dimensiones pueden combinarse para obtener cuatro tipos distintos de generalización: Solapada y total (las personas relacionadas con una biblioteca son empleados y socios, pudiendo ser ambas cosas a la vez), solapada y parcial (un estudiante puede ser estudiante de informática y de física), exclusiva y total (un estudiante puede ser becario o no becario), exclusiva y parcial (un documento puede ser un libro o un artículo).

El MU propone tan sólo dos tipos de generalización:

- *Generalización-and* (and-generalization).- especialización simultánea sobre distintas dimensiones ortogonales entre sí. Cada rama de la jerarquía representa una dimensión de la especialización (por ejemplo, los animales se clasifican, según el hábitat en animales de aire de tierra, de mar, etc. y según la dimensión biológica en mamíferos, reptiles, etc). Corresponde al concepto de generalización múltiple de MIMO.
- *Generalización-or* (or-generalization).- Corresponde al concepto de exclusividad de MEDEA.

El MU no distingue entre exclusividad y solapamiento y las generalizaciones son siempre exclusivas. Una generalización solapada puede obtenerse a través de una clase que herede simultáneamente de dos superclases[52]. Así, el ejemplo del estudiante de informática y de física que en MEDEA se modela mediante una generalización solapada, en el MU se modelará como sigue: un supertipo estudiante con dos subtipos, estudiante de informática y estudiante de física, en una generalización-or; y un subtipo, estudiante de informática y física, con dos supertipos (estudiante de informática y estudiante de física). Este tipo de restricción en las generalizaciones del MU se debe a

que, el MU, no soporta múltiple ejemplificación, por lo que un objeto sólo podrá ser ejemplar directo de un TO (e indirecto de sus supertipos).

Algunos modelos, como STEP/EXPRESS, soportan también los tres tipos de generalización de MIMO:

ONEOF.- si los subtipos son mutuamente exclusivos

ANDOR.- si los subtipos no son mutuamente exclusivos

AND.- si las instancias del supertipo se clasifican en múltiples grupos de subtipos que son ortogonales entre sí.

La correspondencia de estos tres tipos de generalización con el MU, MEDEA y MIMO puede verse reflejada en la tabla 4.7. En realidad EXPRESS, al igual que MIMO, soporta únicamente un tipo de interrelación de generalización. Los tres tipos expuestos se obtienen por aplicación de restricciones al tipo básico.

EXPRESS	MU V0.8	UML V0.1	MEDEA	MIMO
ONEOF	or-generalization	disjoint-generalization	generalización de exclusividad	generalización + restricción de exclusividad
ANDOR	or-generalization + herencia múltiple	overlapping-generalization	generalización de solapamiento	generalización
AND	and-generalization	and-generalization	múltiples jerarquías	generalización múltiple

Tabla 4.7: comparación de generalizaciones en los distintos modelos

Permitir la posibilidad de subtipos solapados tiene diversas repercusiones de gran relevancia en un modelo. Soportar solapamiento implica soportar múltiple ejemplificación. Aunque existen varios modelos de múltiple ejemplificación, como Chimera en Ceri (1993), OASIS en Pastor y Ramos (1995), ROSES en Costa et al. (1996), o el modelo del SQL3, existen otros que mantienen el concepto de ejemplificación única, por el que un objeto no puede pertenecer simultánea y directamente a más de una clase; este es el caso del ODMG-93 o del MU[53]. La ejemplificación múltiple plantea algunos problemas de implementación, como la

[52] En UML V0.91 se incluye ya la posibilidad de solapamiento o exclusividad en las generalizaciones.
[53] El MU V0.8 y V0.9 no soportaba múltiple ejemplificación, sin embargo en UML V0.91 y V1.0 ya se soporta este concepto (multiple classification).

creación de objetos o la resolución de conflictos derivados del polimorfismo; también deben tenerse en cuenta otros aspectos como las limitaciones respecto a las posibilidades de implementación de IDOs, Wieiringa (1995), así como las restricciones de identidad, López et al. (1994). Sin embargo, también la ejemplificación única plantea problemas, especialmente en sistemas de implementación con muchos tipos, derivados de la necesidad de creación de muchos subtipos con herencia múltiple. Por este motivo, según afirma el propio Melton, el SQL3 ha cambiado su modelo de ejemplificación desde sus primeras versiones en que presentaba ejemplificación simple, respecto al modelo actual en que se permite ejemplificación múltiple. Melton (1994), respecto a la pérdida de tal restricción en el SQL3, afirma: “SQL3 se ha desprendido de esta restricción y ha demostrado que existen pocos problemas, en el caso de haber alguno, asociados con la pérdida de dicha restricción”.

En nuestra opinión un modelo, especialmente a nivel conceptual, debe permitir la mayor riqueza semántica posible, independientemente de los problemas de implementación que ello conlleve. La ejemplificación única, además de que también plantea problemas de implementación, limita las posibilidades semánticas del modelo. Por ejemplo, en el MU la generalización básica es la exclusiva (or-generalization) y el solapamiento se consigue introduciendo un subtipo con herencia múltiple respecto a los dos tipos que se solapan.

El MU[54] no contempla la distinción entre totalidad y parcialidad de MEDEA. Esta distinción si es, sin embargo, contemplada en otros modelos, como el de OMT u OOM. Sin embargo, es interesante destacar la relación existente entre el concepto de clase abstracta definido en MIMO y totalidad. Una clase abstracta no puede ser ejemplificada directamente y, por tanto, existirá siempre totalidad. En el MU se utiliza "clase diferida" para hacer referencia a una "clase abstracta", estableciéndose una equivalencia

[54] Esto es así en las versiones 0.8, 0.9 y 0.91. Sin embargo en la versión 1.0 de UML, de enero de 1997, se ha incluido la distinción entre totalidad (complete) y parcialidad (incomplete).

semántica entre estos dos términos. En MIMO, sin embargo, se mantiene la distinción entre clase[55] abstracta y clase diferida (ver 4.4.2).

Debido a que una clase abstracta no puede ser directamente ejemplificada, habrá de serlo siempre a través de sus subclases, dando así lugar a la generalización total de MEDEA. MIMO distingue entre clases abstractas y diferidas y elimina la totalidad/parcialidad como un tipo especial de generalización. Toda generalización en la que la superclase sea abstracta, coincidiendo con el planteamiento de EXPRESS, es total; cuando la superclase no es abstracta la generalización es parcial.

En realidad, en el MU se puede evitar la necesidad de una clase no ejemplificable debido a la limitación de que todo ejemplar tenga un único tipo más específico, ya que la ejemplificación se realizará sobre dicho tipo más específico[56]. De este modo las generalizaciones totales/parciales se obtendrán de un modo natural. Nosotros no compartimos este enfoque, ya que dicha restricción ("naturalmente realizada") no se reflejará en tiempo de análisis, ni en tiempo de construcción (sí en explotación), lo que, en nuestra opinión, limita de un modo innecesario la capacidad expresiva del modelo.

Todo lo dicho para las generalizaciones entre TO es igualmente aplicable a las generalizaciones entre TI. Esto es debido a que las generalizaciones de TI se establecen entre TI niveladas, y éstos, en MIMO, son TO. En MERISE (1994) se contemplan también los casos de totalidad/parcialidad, exclusividad/solapamiento en generalizaciones de TI.

En la tabla 4.8 se muestra un resumen de la correspondencia entre los sistemas de interrelaciones de MIMO y el MU.

[55] En MIMO se distingue entre tipos abstractos o clases abstractas según la fase de desarrollo. No obstante, en la presente discusión hemos preferido mantener el término clase, a fin de evitar ambigüedades, ya que este es el término empleado por el MU.

[56] Al permitir el solapamiento en UML se ha introducido también la posibilidad de generalizaciones totales o parciales.

MIMO	MU V0.8	UML V1.0
Niveladas	(Multiple-)(Self-)Association	(Multiple-)(Self-)Association
Interrelación nivelada + restricción de exclusividad	Or-association	Or-association
Interrelación nivelada + restricción de exclusión	-	-
Interrelación nivelada + restricción de inclusión	Association + Subset constraint	Association + Subset constraint
Interrelación nivelada + restricción de orden	Association + Ordered constraint	Association + Ordered constraint
Interrelación + restricción de dependencia en existencia	-	-
Interrelación + restricción de dependencia en identificación	-	-
TI entre un TO y un TI	Association class	Association class
Generalización	Or-generalization + herencia múltiple	**Overlapping-generalization**
Generalización + restricción de exclusividad	Or-generalization	**Disjoint-generalization**
Generalización múltiple	And-generalization	And-generalization
Generalización entre TI	Se convierte el TI en una clase	Se convierte el TI en una clase
Generalización con supertipo[57] abstracto (no necesariamente diferido)	Or-generalization con superclase abstracta (diferida)	**Complete-generalization**
Meronimia Compuesto/componente Miembro/colección Miembro/colección + restricción de orden	Aggregation Tree aggregation Aggregation Aggregation + restricción de orden	Aggregation Tree aggregation Aggregation Aggregation + restricción de orden
Física/catálogo	Physical/catalog aggregation	Physical/catalog aggregation

Tabla 4.8: Correspondencia entre el sistema de interrelaciones de MIMO y el MU

Es importante señalar que las actualizaciones realizadas en el modelo de objetos del MU en sus últimas versiones se aproximan a las decisiones tomadas, antes de la aparición de las últimas versiones de UML, en MIMO. Y que, precisamente, estas modificaciones vienen a subsanar algunas de las deficiencias del modelo que nosotros ya habíamos detectado, Marcos et al. (1997b).

La posibilidad de combinar TI primitivos con TR en MIMO da lugar otras posibilidades de TI además de los discutidos anteriormente. Por ejemplo, la agregación disjunta de OASIS (ver 3.2.8.12), puede obtenerse en MIMO añadiendo una restricción de exclusividad al TI meronímica. Además, esta posibilidad soporta mayor semántica que la agregación disjunta de OASIS ya que en MIMO se permite especificar qué merónimos son disjuntos entre sí y cuáles no lo son (no tienen que serlo todos necesariamente). En OASIS, sin embargo, o son disjuntos todos los merónimos o ninguno.

Pero, quizá, la principal ventaja de las posibilidades de combinación de TR con TI, es la facilidad para la extensibilidad del modelo que este enfoque proporciona. Para ilustrar esta facilidad vamos a plantear algunas extensiones a MIMO que permitirían soportar conceptos, como el de la herencia de implementación o el concepto de papel. Veremos con que sencillez, y sin modificar en modo alguno su filosofía, puede extenderse el modelo propuesto.

4.5. Extensibilidad de MIMO: un beneficio de la combinación de TI y TR

Una de las principales ventajas de MIMO es que presenta una gran flexibilidad para adaptarse o extenderse. En realidad, se puede utilizar MIMO de diversas formas: como modelo conceptual o como modelo de diseño e implementación; como un modelo sencillo, utilizando sólo los tipos e interrelaciones básicas (para usuarios poco expertos o para la resolución de problemas poco complejos) o como un modelo, aunque menos simple, con mayor capacidad expresiva (por un diseñador con experiencia o para el modelado de problemas complejos), etc. Pero, además, es posible añadir nuevas capacidades a MIMO de un modo intuitivo y manteniendo su filosofía. Veamos, a título

[57]En el nivel de análisis de MIMO hay TO, no clases, por lo que hablamos de tipo abstracto en lugar de clase abstracta.

de ejemplo, como podría extenderse MIMO a fin de soportar generalizaciones de herencia[58] y de papeles.

4.5.1. Extensiones de MIMO para soportar la generalización de herencia

Una generalización de herencia es aquella que se define con el único fin de reutilizar características. Sin embargo, las clases (este tipo de generalización sólo tiene sentido en construcción) que la componen no responden al concepto de ES-UN.

Este tipo de generalización se podría soportar en MIMO haciendo que la restricción de dependencia en existencia entre el subtipo y el supertipo, así como la restricción de dependencia en identidad (si se trata de una generalización de clases de objetos), pasen a ser aplicables en lugar de implícitas.

De este modo, el tipo de generalización básica sería la de herencia. Aplicando una restricción de dependencia en existencia (bidireccional, es decir entre el subtipo y el supertipo y entre el supertipo y el subtipo), así como una restricción de dependencia en identidad, a una generalización de herencia, convertimos ésta en una generalización del tipo ES-UN.

4.5.2. Extensiones de MIMO para soportar el concepto de papel

Añadimos un nuevo tipo de generalización: la generalización de papeles. Una generalización de papeles permite reflejar los distintos cometidos que un TO puede desempeñar (por ejemplo, una *persona* puede desempeñar la función de *estudiante* o la de *profesor*).

Una generalización de papeles es siempre una generalización ES-UN; sin embargo, lo contrario no es cierto (por ejemplo, un *documento* puede ser un *libro* o un *artículo*, pero *libro* y *artículo* no son distintos papeles de *documento*).

[58] Es importante destacar que MIMO no incluirá, de momento, estas dos extensiones. Tan sólo tratamos de mostrar su facilidad de extensibilidad. Además, en principio, no está previsto añadir en MIMO generalizaciones de herencia, aunque sí lo está incluir el concepto de papel.

La generalización de papeles la obtendríamos aplicando a la generalización de herencia (sin dependencias implícitas, ni en existencia ni en identidad) una restricción de dependencia en identidad. Sin embargo, a diferencia de la generalización ES-UN, no aplicamos una restricción de dependencia en existencia del supertipo respecto del subtipo (si se aplica la dependencia en existencia del subtipo respecto del supertipo).

Así, en el ejemplo que habíamos planteado anteriormente, una *persona* podría dejar de desempeñar el papel de *estudiante* y pasar a ser *profesor* (movilidad de objetos en generalizaciones de roles), o sencillamente ser *persona* sin ningún papel asignado. En este caso, se eliminaría el *estudiante*, pero no la *persona* (el supertipo no depende en existencia del subtipo). Si se elimina una persona, se eliminan todos los papeles que tuviera asociada (los subtipos sí dependen en existencia del supertipo).

Sin embargo, en la generalización ES-UN, no tiene sentido que un *libro* pase a ser un *artículo* (imposibilidad de movilidad de objetos en generalizaciones ES-UN), ni que desaparezca como *libro* y permanezca como *documento* (el supertipo depende en existencia del subtipo). Además, en una generalización ES-UN, el subtipo también depende en existencia del supertipo.

La tabla 4.9 muestra como la aplicación de restricciones a la generalización de herencia para obtener las generalizaciones de papeles y ES-UN.

	D. en identidad	**D. en existencia Sub. de SUP.**	**D. en existencia SUP. de Sub.**
G. Herencia	No	No	No
G. Papeles	Si	Si	No
G. ES-UN	Si	Si	Si

Tabla 4.9: aplicaciones de restricciones a la generalización

No obstante, esto es sólo una primera aproximación a la modelación de papeles y somos conscientes de que quedan algunos aspectos importantes sin tratar (nuestro objetivo aquí es, tan solo, ilustrar a cerca de las posibilidades de extensibilidad de MIMO). Por ejemplo, existen algunos casos en los que no es tan claro si se trata de una generalización de herencia o de papeles. Supongamos una generalización donde niño y adulto son subtipos de persona. El estado de una persona variará a lo largo de su vida y

dejará de ser niño para ser adulto. Sin embargo, el hecho de que el estado de algo varíe, no implica que deje de ser lo que es para convertirse en otro algo (varía su estado, no su ser[59] o naturaleza). Por ello, es dudoso si trataría de una generalización ES-UN o una generalización de papeles, donde una persona desempeña un papel u otro dependiendo de su estado. En nuestra opinión, es posible que hubiera que pensar en otro tipo de generalización del que la literatura no habla: la generalización de estados. En el ejemplo de persona, niño y adulto, una persona cambiaría de clase en función de su estado, pero no en función del papel que desempeñara, ni porque su naturaleza como "ser" hubiese cambiado.

Por este y otros aspectos, en los que consideramos que es necesaria una reflexión más profunda, en MIMO se ha decidido no incluir por el momento el concepto de papel.

Con estos dos ejemplos hemos ilustrado con que sencillez pueden modificarse o ampliarse las funcionalidades de MIMO. El concepto de papel, como ya hemos dicho, es uno de los aspectos que está previsto estudiar en la primera revisión que se haga del modelo, ya que lo consideramos un tema de gran interés y actualidad, Wieiringa et. al (1995), Wold y Lehne (1996), Kristensen y Østerbye (1996). Por ello, el modelo se ha pensado y preparado para incluir éste y otros conceptos, con muy pocas modificaciones.

[59] En este sentido, es muy ilustrativo pensar en el significado del nombre de la generalización (ES-UN). Posemos observar que se trata del verbo ser y no del verbo estar, cuyos significados en español son claramente distintos. Es importante observar que el inglés no proporciona dos verbos diferentes para hacer referencia a estos dos conceptos. Esta distinción podría sugerir una conexión con la distinción Aristotélica entre cambio accidental y cambio sustancial, Candel (1982).

5 Validación y verificación

"La máquina funciona muy bien, podrá resolver todos los problemas que se quiera, pero jamás sabrá plantearse ni uno sólo".

Einstein

En el capítulo 1 se definía una hipótesis de trabajo, así como los objetivos que nos proponíamos en la definición de MIMO. En el presente capítulo se pretende validar que MIMO cumple los objetivos previstos, para concluir que se ha trabajado bajo una hipótesis válida. Debemos recordar que la etapa de validación de nuestro método de trabajo (2.2.2), se incluía la validación y la verificación. Por tanto, en este capítulo se trata, no sólo de validar que MIMO cumple la hipótesis planteada, sino también de verificar, en la medida de lo posible, la consistencia de MIMO en si mismo.

La verificación y validación de MIMO se ha llevado dentro de un proyecto denominado ENEAS/BD, por lo que comenzaremos explicando cómo se integra MIMO dentro de dicho proyecto (5.1). Posteriormente hablaremos de las dos tareas que nos han permitido realizar la verificación y validación parcial de el trabajo: la implementación de MIMO (5.2 y 5.3) y la correspondencia de éste con los modelos subsumidos por él (5.4).

El proceso de validación y verificación de MIMO lleva consigo las siguientes tareas:

a) Verificación de la compleción y consistencia de MIMO en sí mismo.

b) Validar que MIMO soporta los niveles de análisis, construcción y explotación y verificar MIMO como modelo para cada una de estas fases.

c) Validar de la hipótesis de que MIMO integra los modelos del MU, SQL3 y ODMG-93.

Es importante señalar que la validación de alguna de las hipótesis, aunque apoyada en comparaciones, casos de prueba, etc., es subjetiva y que debe ser el usuario quien decida si se cumplen o no. En este caso el usuario es el propio doctorando (y sus directores) ya que son ellos quienes deciden lo que quieren o no, y en que medida sus objetivos se han cumplido. En cuanto a la verificación, tal y como se discutió en 3.2.1, la formalización (más o menos estricta) sólo la garantiza en un cierto grado.

El proceso de verificación y validación, como ya hemos dicho, se ha llevado a cabo dentro del proyecto ENEAS/BD, De Miguel et al. (1996a), por lo que, en primer lugar, pasamos a exponer cómo se integra MIMO dentro de este proyecto.

5.1. Integración de MIMO en ENEAS/BD

Decíamos en el capítulo 1 (epígrafe 1.3) que el trabajo de la presente tesis doctoral se encuadra dentro de un proyecto de investigación más amplio denominado ENEAS/BD. ENEAS/BD consta de un conjunto de módulos que se han de integrar por medio de una metabase (ver figura 1.2, en el capítulo 1). La verificación y validación de MIMO se va a realizar precisamente dentro de este entorno. Por ello, en primer lugar, vamos a explicar las funcionalidades de MIMO en ENEAS/BD y cómo cada una de ellas contribuye a verificar MIMO y a validar las distintas hipótesis planteadas en el capítulo 1.

En la figura 5.1 se muestran los módulos de ENEAS/BD en los que se enmarca MIMO y la comunicación que se establece entre ellos.

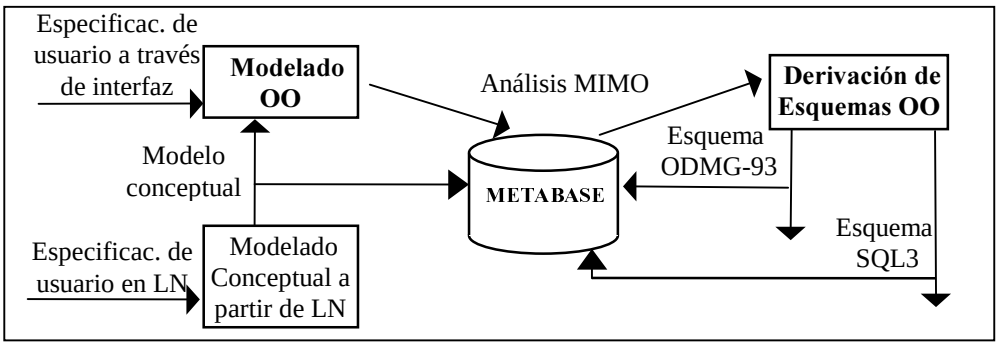


Figura V.1: comunicación entre los módulos de **Modelado OO** y **Derivación de Esquemas OO** por medio de la metabase

El módulo de ***Modelado OO*** obtiene un análisis en MIMO; este análisis puede ser realizado directamente por el usuario a través de un interfaz gráfico, o bien puede obtenerse semiautomáticamente a partir de especificaciones introducidas en lenguaje natural (procesadas por otro módulo denominado *Modelado Conceptual a partir de Lenguaje Natural -LN-)*, De Miguel et al. (1996b).

El módulo de *Derivación de* ***Esquemas OO*** convierte un análisis MIMO a una construcción[1] MIMO, a partir de la cual generará el correspondiente código ODMG-93 o SQL3. El objetivo final de este módulo es acercarse, en la medida de lo posible, a la automatización del proceso de traducción de un análisis MIMO a una implementación SQL3 u ODMG-93. Dicha traducción proporcionará un primer esquema en uno de éstos dos lenguajes que podrá ser refinado mediante cambios introducidos por el usuario.

La salida de cada uno de estos módulos puede dirigirse, bien directamente a un interfaz de usuario, bien a la metabase con el fin de que la salida de un módulo pueda constituir la entrada de otro. Igualmente, las entradas de cada módulo podrán obtenerse, bien directamente a través de un interfaz de usuario, bien a partir de información almacenada en la metabase.

[1] Recordemos que MIMO soporta los distintos niveles en el ciclo de desarrollo de una base de datos: análisis, construcción (que incluye diseño e implementación) y explotación.

Por tanto, podemos decir que MIMO tiene tres funcionalidades en ENEAS/BD:

a) Una vez recogidas las especificaciones de usuario, a través de un interfaz o por medio del módulo de lenguaje natural, el módulo de ***Modelado OO*** se encarga de obtener el análisis conceptual en MIMO. Esta funcionalidad lleva consigo la implementación de una herramienta CASE que permita el modelado conceptual en MIMO y permite verificar, parcialmente, y validar la capacidad de modelado conceptual en MIMO. Como ejemplo de modelado conceptual en MIMO se presentará el modelado de MIMO propiamente dicho (que además servirá de metaesquema para la implementación de la metabase de ENEAS/BD, ver b).

b) Es el modelo que soportará la ***metabase***. Esta funcionalidad implica la necesidad de implementación de MIMO, lo cual implica un cierto grado de formalización (en 3.1.2 se discuten los conceptos de formalización y automatización). La implementación parcial de MIMO, junto con la aplicación de casos de prueba, permite realizar una verificación parcial de la compleción y consistencia de MIMO.

c) A partir de un análisis en MIMO, el módulo de ***generación de esquemas OO*** obtiene una construcción MIMO y el correspondiente código en SQL3 y ODMG-93. Esta funcionalidad permite validar la hipótesis de que MIMO integra los modelos de los estándares para bases de datos orientadas al objeto más importantes. También permite validar la hipótesis de que MIMO soporta los niveles de construcción y explotación y verificar, parcialmente, su capacidad de modelado en dichas fases (respecto a la verificación del nivel de análisis ver a).

Resumiendo, diremos que la validación y verificación de MIMO se realiza en dos etapas:

a**)** *Verificación de la compleción y consistencia de MIMO*:

a.1) implementación de MIMO como modelo de análisis del módulo de Modelado OO de ENEAS/BD. Permite, también, validar la capacidad de MIMO como modelo conceptual.

a.2) implementación de MIMO como modelo de la metabase de ENEAS/BD. Como tarea previa a la implementación será necesario describir el esquema conceptual de MIMO en algún modelo que soporte el nivel de análisis. El modelo elegido es el propio MIMO, lo que permitirá *verificar, y validar, su capacidad para el modelado conceptual* al definir MIMO recursivamente en términos de sí mismo.

b) *Verificación, y validación, de MIMO como modelo que integra los modelos del MU, SQL3 y ODMG-93*: para ello se presenta **una aproximación a la correspondencia de estructuras** de MIMO con cada uno de los modelos citados lo que permite, a su vez, *verificar MIMO como modelo de que soporta los niveles de análisis, construcción y explotación* (ya que SQL3 y ODMG-93 son modelos de diseño e implementación y el MU de análisis y diseño).

5.2. MIMO-CASE: Implementación de MIMO como modelo conceptual del módulo de Modelado OO

Se ha implementado un primer prototipo del módulo de Modelado OO, MIMO-CASE, que se encuentra ya operativo. MIMO-CASE, Vela y Serrano (1997), se ha implementado en Delphi V2.0 y en Object-Pascal. Delphi es un entorno de desarrollo rápido de aplicaciones Cliente/Servidor, que genera código Pascal Orientado al Objeto. Hay que destacar que, a fin de mantener un desarrollo homogénea con el paradigma en el que nos movemos, toda la implementación se ha llevado a cabo en programación orientada al objeto.

MIMO-CASE, en la actualidad, proporciona las siguientes funcionalidades:

- Soporta el modelado conceptual en MIMO, utilizando notación gráfica.
- Realiza una verificación parcial de esquemas, interactuando con el usuario.
- Esta preparado para incorporar la facilidad de modelado conceptual en notación sintáctica MIMO[2], y para convertir esquemas conceptuales en MIMO de notación gráfica a sintáctica y viceversa.

MIMO-CASE se ha validado con diversos casos de prueba. Aunque no se ha hecho un estudio sistemático de pruebas, se han introducido los casos más significativos, verificando así su operatividad en los escenarios más relevantes. En el disquete 1, se entrega el código, un ejecutable, así como varios ejemplos de modelado conceptual en MIMO. En la figura 5.2 se muestra la pantalla principal de MIMO-CASE con uno de los casos de prueba seleccionados.

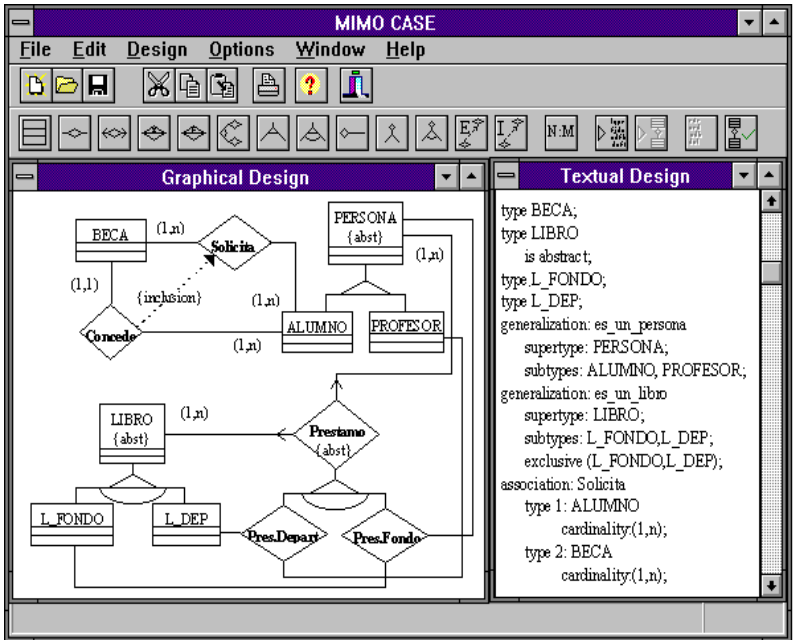


Figura 5.2: MIMO-CASE: entorno para el modelado conceptual en MIMO

[2] Aunque en la actualidad MIMO sólo propone una notación gráfica, se está trabajando en la elaboración de un lenguaje que permita el modelado en MIMO, tanto en análisis como en construcción.

5.3. Implementación de MIMO como modelo de la metabase de ENEAS/BD

Esta tarea se lleva a cabo en dos etapas: análisis y construcción (que incluye diseño e implementación propiamente dicha):

a) Análisis

En esta etapa se describe el esquema conceptual de MIMO (que será el metaesquema de ENEAS/BD). A fin de verificar la capacidad de modelado conceptual en MIMO, dicha descripción, soportada por MIMO-CASE, se realiza recursivamente en MIMO (figura 5.3). La notación gráfica propuesta en MIMO (ver apéndice III) se basa en las notaciones propuestas en De Miguel y Piattini (1993), Rumbaugh (1991) y Booch y Rumbaugh (1995).

esquema conceptual: METAESQU.DOC

b) Construcción

En esta etapa se realizan el diseño (ver apéndice IV) y la implementación de MIMO a partir del esquema conceptual definido en el apartado a).

Como ya hemos visto, la implementación de MIMO permite verificar, parcialmente, su consistencia ya que lleva consigo la automatización del modelo. Debido a la complejidad de MIMO (integra varios modelos de objetos y soporta la definición de esquemas a nivel conceptual y a nivel de construcción) su implementación completa en un período de tiempo razonable no es factible. Por ello, y tomando como base el esquema conceptual propuesto (ver figura 5.3), se ha limitado su implementación a aquella parte de MIMO no soportada por otros modelos y que, por tanto, constituye la parte más novedosa y distintiva del nuestro. Se propone el diseño del nivel de análisis de MIMO y la implementación de los TO y de los TI[3]. No se implementan los TV ya que éstos son soportados en todos los sistemas por lo que las aportaciones derivadas de su implementación no son tan relevantes.

La implementación, Bermúdez (1997), se ha llevado a cabo en POET V4.1 (Persistent Object and Extended Technology), un SGBO que comenzó siendo de libre distribución y para el que ya existen versiones comerciales en distintas plataformas[4] (MIMO se ha implementado en Windows NT). Tiene un modelo de objetos basado en el modelo del ODMG-93 pero no totalmente equivalente, por lo que nos ha permitido reutilizar alguna parte de su modelo. Su lenguaje de definición de datos está basado en C++ (con extensiones de persistencia) y su lenguaje de consulta en el OQL propuesto por el ODMG.

La implementación se ha probado con los mismos casos que se emplearon para MIMO-CASE, lo que ha permitido verificar su operatividad en los escenarios más significativos. En el disquete 2 se entrega el código fuente, el ejecutable y la base de

[3] La parte de MIMO diseñada, así como la parte implementada, se muestran en la figura 5.3 (zona 2 y zona 3 respectivamente) rodeadas en trazo discontinuo.

[4] Distintas versiones de Windows y UNIX, OS/2, Macintosh, NetWare and IntranetWare.

datos con los casos de prueba. El ejecutable y los casos de prueba sólo podrán utilizarse junto con POET.

5.4. Una aproximación a la correspondencia de estructuras

La correspondencia entre las estructuras de MIMO y las del MU, SQL3 y ODMG-93 permite validar la hipótesis de que MIMO integra tales modelos a la vez que verifica, parcialmente, MIMO como modelo que soporta los niveles de análisis y construcción.. Hablamos de *una* aproximación, ya que la correspondencia entre modelos no es única y aquí se presenta *una* de las posibles.

La correspondencia MIMO/MU se ha realizado ya en el capítulo 4 (epígrafe 4.4.4) y se centraba en el sistema de interrelaciones lo que constituye el núcleo del modelo del MU. En dicho capítulo se exponía MIMO y se presentaban sus posibilidades de extensión mediante la aplicación de Tipos de Restricción a Tipos de Interrelación. La comparación del sistema de interrelaciones de MIMO con el del MU nos permitió ilustrar, en parte, tal posibilidad de extensión, así como comparar el Sistema de Interrelaciones con los de otros modelos como el de MEDEA o STEP/EXPRESS. Es por este motivo por lo que la correspondencia MIMO/MU no se presenta en este apartado.

La correspondencia MIMO/SQL3 y MIMO/ODMG-93 se realiza a partir de las estructuras del nivel de construcción de MIMO ya que los citados estándares son modelos construcción. La correspondencia que presentamos se integra en ENEAS/BD dentro del módulo de *Derivación de Esquemas OO*, como paso previo y necesario a la implementación de éste.

5.4.1. Correspondencia MIMO/SQL3 y MIMO/ODMG-93

Uno de los principales objetivos planteados al comienzo de la realización del presente trabajo (ver capítulo 1), era el de definir un modelo que, además de cubrir todas las fases del desarrollo de una base de objetos, integrara los principales modelos de

objetos existentes en la actualidad. En el capítulo 4 se ha abordado dicho objetivo con la definición de MIMO. Se planteaba además, como otro objetivo importante, la posibilidad de definir una primera aproximación a la correspondencia entre las diferentes estructuras de la fase de construcción de MIMO y las de los modelos de SQL3 y ODMG-93, los dos estándares de bases de objetos más importantes en la actualidad. De este modo se podrá, a partir de una construcción en MIMO, obtener el código correspondiente, bien en SQL3 bien en ODMG-93. En este apartado se define una primera aproximación a la correspondencia de estructuras.

Para poder definir una correspondencia de estructuras rigurosa sería preciso elaborar un método que permitiera decidir sobre las posibles transformaciones de un análisis a una construcción MIMO y a sus posibles implementaciones en SQL3 u ODMG-93. Sin embargo, la elaboración de un método no estaba entre los objetivos de este trabajo. Además, el modelo de objetos del SQL3, aunque ya estable, todavía está sujeto a cambios y no define la sintaxis completa de sus estructuras por lo que, por el momento, no es posible elaborar una versión definitiva de la traducción a este lenguaje[5].

El objetivo final de este apartado es acercarse, en la medida de lo posible, a la automatización del proceso de traducción de un análisis MIMO a una implementación SQL3 u ODMG-93. Dicha traducción proporcionará un primer esquema en uno de éstos dos lenguajes que podrá ser refinado mediante cambios introducidos por el usuario.

La traducción entre esquemas SQL3/ODMG-93 debería realizarse a través de MIMO y para ello, la correspondencia de estructuras tendría que ser bidireccional. Sin embargo, la transcripción de todo esquema SQL3 a ODMG-93 y viceversa, queda fuera del alcance de esta Tesis, por lo que la compleción sólo se verifica en un sentido: MIMO ---> SQL3, MIMO ---> ODMG-93). Lo que aquí pretendemos es, tan solo, mostrar la viabilidad de este enfoque. No obstante, dado el interés de la propuesta, la compleción de correspondencias se plantea como una futura extensión al presente trabajo. Así mismo, también se plantea como posterior línea de trabajo el refinamiento de la primera

[5] En la última reunión del comité (Madrid, enero-febrero del 97), se aprobó la parte de objetos del SQL3 como borrador de comité (CD). Podemos hablar ya, por tanto, de una propuesta estable. Sin embargo, en ese momento la presente tesis estaba ya en su fase final, por lo que el trabajo que aquí se presenta esta basado en las anteriores versiones del modelo y éstas han sufrido ya algunas modificaciones importantes.

aproximación que ahora se propone (lo que podría, incluso, implicar algunos cambios en MIMO), así como la elaboración de un método que cubra las fases de análisis y construcción de una base de objetos en MIMO.

El nivel de construcción de MIMO se compone de un conjunto de características (atributos, restricciones y operaciones), un sistema de tipos de datos[6] y clases. Las clases pueden ser: clases de implementación de TO o clases de interrelación (implementan TI niveladas). Los TI de generalización y de meronimia se mantienen en construcción como generalizaciones y meronimias entre clases (ver tabla 4.5 en capítulo 4, epígrafe 4.2.1).

En este apartado veremos, en primer lugar, la correspondencia de características básicas (estáticas y dinámicas), Marcos et al (1997a). A continuación entraremos en la correspondencia del sistema de tipos[7]; incluiremos también en este apartado la correspondencia de las clases correspondientes a TO, Marcos et al. (1997a). Para finalizar estudiaremos la correspondencia entre el sistema de interrelaciones de MIMO, en su nivel de construcción (clases de interrelación, generalizaciones y meronimias), con los sistemas de interrelaciones de los estándares tratados.

5.4.1.1. Correspondencia de características básicas

Antes de analizar las equivalencias entre el ST, es necesario definir la correspondencia entre las estructuras básicas en el nivel de construcción de MIMO, características estáticas (atributos y restricciones) y características dinámicas (servicios), con las de los modelos de los estándares tratados.

Características básicas

[6] Recuérdese que todo tipo (valor u objeto) en análisis, pasa a ser una clase en construcción. Sin embargo, y a fin de evitar ambigüedades terminológicas, se decidió hablar de *clase* para hacer referencia a la implementación de un TO y de *tipo* para hacer referencia, tanto a un tipo valor, como a su implementación.

[7] Que en MIMO son realmente clases.

Incluimos entre las características básicas los atributos, las restricciones aplicables a éstos y los servicios. Su correspondencia con ODMG-93 y SQL3 es la siguiente:

- ***Atributos***.- Son atributos en ambos modelos. Tanto en SQL3 como en ODMG-93 se soportan atributos base (que son siempre reales) y derivados (que son siempre virtuales). Estos últimos se implementan a través de funciones en SQL3 y operaciones en ODMG-93. Ninguno de los dos modelos soporta atributos derivados reales (siempre que un atributo es derivado, es virtual), por lo que la transformación de todo atributo derivado en MIMO a cualquiera de los otros dos modelos dará lugar a un atributo derivado virtual (definido como una función o una operación, según el modelo objetivo), con independencia de que en MIMO fuese real o virtual. La implementación de un atributo derivado real de MIMO en SQL3 u ODMG-93 se hará mediante un atributo y una función u operación que calcule y almacene el valor de dicho atributo. Aunque en SQL3 un atributo derivado real se podría implementar también mediante un disparador, hemos mantenido su implementación a través de funciones por homogeneidad con la transformación al ODMG-93, ya que este estándar carece de disparadores. Consideramos, de todos modos, que esto constituye una limitación de ambos estándares, en especial de ODMG-93 que ni siquiera proporciona la alternativa de los disparadores.

- ***Restricciones***.- ODMG-93 no soporta restricciones y, aunque es una de las ampliaciones previstas en futuras revisiones del estándar, en la versión actual, Cattell (1995), aún no se han incluido, por lo que su semántica deberá recogerse mediante operaciones. SQL3 soporta todas las restricciones sobre atributos y clases de MIMO: no nulidad (NOT NULL), unicidad (UNIQUE), validación de atributo o de clase (CHECK) y aserción (ASSERTION).

- ***Servicios***.- Son operaciones en ODMG-93 y rutinas (funciones o procedimientos) en SQL3.

MIMO soporta, además, el concepto de **características de clase y de ejemplar**. En SQL3, aunque no se recoge de modo explícito la distinción entre característica de

clase y de ejemplar, sí se hace de un modo implícito, ya que las generalizaciones se implementan mediante una especificación de los supertipos directos, lo que podría considerarse un características de clase. ODMG-93 considera los supertipos como características de tipo e incluye, además, otras dos: la extensión y las claves.

Sin embargo, en SQL3 las **claves** se consideran restricciones y no características de clase. SQL3 soporta claves primarias mediante una restricción de PRIMARY KEY y alternativas mediante la restricción UNIQUE, incluyéndose opcionalmente NOT NULL. MIMO soporta el concepto de clave primaria mediante un atributo (o conjunto de atributos) especial. Las claves alternativas se soportan mediante una restricción de unicidad y una restricción de no nulidad.

ODMG-93 permite definir claves, pero con una semántica distinta a la de las claves en MIMO y en SQL3, ya que sólo implican una restricción de unicidad, pudiendo ser nula en parte o en la totalidad de sus atributos. Además, se pueden definir varias claves por lo que se trata, en realidad, del concepto de claves alternativas en SQL3, y corresponden a la restricción de unicidad de MIMO.

En la tabla 5.1 se muestra un resumen de las correspondencias descritas. La letra cursiva indica la implementación propuesta para aquellas estructuras que no tienen una transformación directa.

MIMO	SQL3	ODMG-93
Servicios	Funciones	Operaciones
Atributo base real	Atributo	Atributo
Atributo derivado real	*Atributo+Función*	*Atributo+Operación*
Atributo derivado virtual	Función	Operación
Restricciones	Restricciones	*Operaciones*
no nulidad	NOT NULL	*Operación*
unicidad	UNIQUE	KEY
validación	CHECK	*Operación*
aserción	ASSERTION	*Operación*
Clave primaria	PRIMARY KEY	*KEY+operación*

Tabla 5.1: correspondencia de las estructuras básicas

5.4.1.2. Correspondencia del sistema de tipos

La clasificación del sistema de tipos de MIMO difiere de la propuesta en DBL: MAD-004 (1996) para el SQL3 y de la propuesta por Cattell (1994a) para el ODMG-93. No se pretende aquí realizar una comparación exhaustiva de los tres ST tratados, sino establecer los criterios que permitan realizar una primera aproximación a la correspondencia entre los sistemas de tipos de los tres modelos.

En la tabla 5.2 se establece la equivalencia entre los tipos soportados por MIMO y los soportados por los estándares SQL3 y ODMG-93 pasando posteriormente a su discusión. Al igual que en la tabla 5.1., se muestran en cursiva las transformaciones propuestas para aquellas estructuras que no tienen una correspondencia directa.

MIMO	SQL3	ODMG-93
Básico		
Carácter	Character	Character
Numérico	Numeric	*Float*
Entero	Integer	Integer
Numérico Flotante	Float	Float
Bit	Bit	*Enumeration*
Blob	Blob	*Bit_String*
DíaMes, Año, Hora, Minuto	*Domain (Integer)*	*Enumeration*
DíaSem, Mes	*Enumerated*	*Unsigned Short*
Segundo	*Numeric*	*Float*
Intervalo	Interval	Interval
IDO	OID	OID
Dominio	**Domain**	-----
por extensión	Domain CHECK (value list)	*Enumeration*
por intensión	Domain [CHECK (search_condition)]	*Literal*
Enumerado	Enumerated	Enumeration
Boolean	Boolean	Boolean
Estructura		
Primitiva	**Predefined Type**	**Immutable_Structure**
Fecha	Date	Date
Tiempo	Time	Time
Fecha-tiempo	Timestamp	Timestamp
Intervalo	Interval	Interval
Específica		------
Comport. def. por el usuario	ADT	*Interface*
Comport. def. por el sistema	--------	Structure
Colección	**Collection**	**Collection**
Conjunto	Set	Set
Multiconjunto	Multiset	Bag
Lista	List	List
Tira de caracteres	String	String
Tira de bits	Bit varying	Bit_String
Vector	*List*[8]	Array
Multiconjunto de estructuras	Table	Table
Clase (implementación de TO)	Named Row Type	Interface defin.+Implementation

Tabla 5.2: Correspondencia de los ST

Se puede observar, coincidiendo con Melton (1995), que el conjunto de tipos de SQL y ODMG-93 es diferente. Aunque algunos tipos tienen correspondencia directa, en otros casos es necesario establecer algún tipo de transformación. En Melton (1994) se hace especial hincapié en la necesidad de hacer corresponder la semántica de los tipos colección. En este sentido, una primera propuesta del SQL/ODMG merger group, DBL:

[8] Tal y como ya se ha dicho, debido a la aprobación de la propuesta DBL: MAD-175 (1996), la próxima versión del SQL3, aún no disponible, incluirá ya vectores unidimensionales.

MCI-079 (1996), plantea la convergencia del SQL3 respecto a las consultas sobre colecciones soportadas por ODMG-93. Sin embargo, la diferencia semántica existente entre las colecciones de estos dos estándares no afecta a la correspondencia con MIMO ya que se debe a las diferencias en los tipos de datos en resultados de consultas. En SQL3 el resultado de una consulta es siempre una tabla y además, sólo es posible realizar consultas sobre tablas[9]. Por el contrario, en ODMG-93 es posible efectuar consultas sobre cualquier tipo colección y, de igual modo, el resultado de una consulta puede ser un conjunto, un multiconjunto o una lista. Entre las propuestas actuales para el SQL3 se incluye la posibilidad de incluir la tabla como un tipo colección (una tabla es, en realidad, un multiconjunto); si esta propuesta se aprueba, será necesario revisar la aproximación a la correspondencia entre colecciones presentada.

El SQL3 tiene un sistema de tipos primitivos más rico que el del ODMG-93. Sin embargo, este último proporciona un soporte mejor en lo que se refiere a tipos colección. MIMO, al integrar ambos modelos, proporciona las ventajas de ambos.

Tipos básicos

La mayor parte de los tipos básicos de MIMO tienen correspondencia directa tanto con SQL3 como con ODMG-93. Algunas excepciones son los tipos numérico, bit y blob de MIMO que no existen en ODMG-93 y que, en nuestra opinión, deberán traducirse como float, enumerado y bit_string respectivamente, y los tipos díames, diasem, mes, año, hora, minuto y segundo no soportados, directamente como tipos, en ninguno de los dos estándares. En SQL3 se traducirán en dominios definidos sobre el tipo integer y con las correspondientes restricciones de rango, excepto los tipos mes y diasem que se definirán como tipos enumerados a fin de poder acceder a los meses y a los días de la semana por su nombre o por su posición y el tipo segundo que se traducirá en un numérico por compatibilidad con el estándar[10]; en ODMG-93, los tipos mes y díasem serán igualmente tipos enumerados, el tipo segundo se convertirá en float (por compatibilidad con la definición del propio estándar) y el resto, al no haber dominios, serán tipos unsigned short.

[9] Téngase en cuenta que las vistas son tablas.

Tanto MIMO como SQL3 tienen un tipo de dato identificador de objeto (IDO en MIMO y OID en SQL3). En ODMG-93 aunque no se especifica explícitamente, sí se puede hablar de un tipo de datos OID (el OID es una característica que toma valores asignados por el sistema, los cuales deben pertenecer a un tipo, aunque éste no sea visible al usuario).

Dominio

El dominio existe tanto en SQL3 como en MIMO, pero no en ODMG-93 y su correspondencia se realizará del siguiente modo: todo dominio definido por extensión en MIMO se traducirá al ODMG-93 en un tipo enumerado; los dominios definidos por intensión se traducirán al tipo básico sobre el que se define el dominio en MIMO (que será un literal del ODMG-93), pero sin la restricción asociada.

Enumerado

Un enumerado se traducirá siempre en un enumerado ya que los tres modelos soportan este tipo.

Estructuras

Las estructuras **primitivas** son equivalentes en los tres modelos, por lo que existe una correspondencia directa en los dos sentidos.

Las estructuras **específicas** de MIMO se traducen en Abstract Data Type (ADT) del SQL3 y en Structure en ODMG-93. Estructuras, ADTs y Structures tienen comportamiento, sin embargo en ODMG-93 no está claro como se podría definir una Structure cuyo comportamiento no sea el que implementa el sistema. Por tanto, en aquellos casos en que la estructura tenga comportamiento definido por el usuario, en ODMG-93 deberá definirse como interfaz, por lo que se tendrán estructuras con IDO. Aunque su semántica no es equivalente a la de las estructuras valor de MIMO, ni a la de los ADTs del SQL3, esta es, en nuestra opinión, la traducción más ajustada que se puede hacer.

[10] El SQL3 no define directamente los tipos Día, Mes, Año, Hora, Minuto y Segundo, pero sí especifica los valores que deben tomar cada uno de los campos de los tipos Date y Time.

Colecciones

En cuanto a las colecciones, los tres modelos soportan tres generadores de colección predefinidos (lista, conjunto y multiconjunto). Sin embargo, SQL3 no tiene vectores, por lo que un vector de MIMO se implementará como una lista en SQL3[11].

La tabla del SQL3 se soporta en MIMO mediante un multiconjunto de estructuras. ODMG-93, aunque en sus primeras versiones no soportaba el tipo tabla, Cattell (1994a), la última revisión del estándar ya lo incluye, Cattell (1995). La tabla en ODMG-93 es equivalente al tipo Bag (Structure). A pesar de que en los dos estándares estudiados se soporta directamente el concepto de tabla, en MIMO hemos decidido no incluirlo, ya que puede obtenerse por combinación de dos tipos básicos.

Clase

Una clase (implementación de un TO) en MIMO puede implementarse en SQL3 mediante un tipo fila con nombre (Named Row Type) a la que se le añade comportamiento. SQL3 no distingue entre el concepto de tipo y clase, ya que ambos se definen simultáneamente, por lo que un tipo en SQL3 tiene una sola implementación. Sin embargo, en la última reunión del grupo (Madrid, enero-febrero del 97), aunque no se planteó la posibilidad de múltiples implementaciones para el comportamiento de los objetos, sí se aprobó la posibilidad de múltiples implementaciones para los ADTs, DBL: MAD-226 (1996), lo que, en nuestra opinión, constituye un paso importante en esta dirección.

El interfaz de ODMG-93 es similar, aunque no equivalente, al tipo de MIMO, por lo que un TO en MIMO se traducirá, a un interfaz (sin implementación) en ODMG-93. Una clase de MIMO se transformará, en ODMG-93, en una definición de interfaz junto con su implementación.

Aunque ODMG-93 distingue entre los conceptos de tipo y clase, permitiendo distintas implementaciones para una misma interfaz, es importante subrayar que el concepto de interfaz en ODMG-93 no es equivalente al concepto de tipo en MIMO. Un

tipo en MIMO es un concepto de análisis, mientras que la interfaz del ODMG-93 es un concepto de construcción (concretamente de diseño, pues aún no se ha implementado).

5.4.1.3. Correspondencia del sistema de interrelaciones

Tanto en SQL3 como en ODMG-93 el sistema de interrelaciones es muy limitado. Las únicas interrelaciones niveladas[12] que soportan directamente son las binarias y se implementan mediante referencias desde una de las clases involucradas en la interrelación a la otra (pueden ser unidireccionales o bidireccionales). Debido a que una interrelación es una referencia, en ninguno de los dos estándares se soportan directamente interrelaciones con características propias (sean estas atributos, operaciones o interrelaciones). Por este motivo, tanto las interrelaciones **n-arias,** como las interrelaciones **binarias con características propias** (sean éstas estáticas o dinámicas), se implementan en SQL3 y en ODMG-93 como clases con atributos de referencia a los TO que interrelacionan[13]. La misma solución se adopta para los **TI entre TI** no soportados tampoco en ninguno de los dos estándares.

En cuanto a los TI generados por aplicación de restricciones a los tipos básicos, esto es igualmente aplicable al SQL3, no así al ODMG-93 que carece de restricciones por lo que habría que implementar éstas mediante operaciones si no se desea perder esta semántica.

Las **cardinalidades** de la interrelación determinan el tipo de los atributos de referencia:

- La cardinalidad mínima 1, se traduce en SQL3 en una restricción de no nulidad (NOT NULL) sobre el atributo de referencia; en ODMG-93 esta restricción se implementa mediante operaciones.

[11] Aunque esta limitación del SQL3 ya ha sido subsanada, esto era así en el momento en que se realizó esté trabajo de Tesis Doctoral.
[12] Clases de interrelación (CI) en el nivel de construcción de MIMO. Sin embargo, hablamos de interrelaciones para referirnos tanto a las CI de MIMO como a las interrelaciones de SQL3 y ODMG-93 para no añadir más complicación a la discusión con aspectos terminológicos.
[13] En V.4.1.2 se discute la implementación de las clases en cada uno de los dos estándares.

- La cardinalidad máxima mayor que uno se traduce, en ambos casos, como un atributo de referencia de tipo colección.

La **meronimia** también se soporta en los dos modelos de los estándares vía atributos. Sí se trata de meronimia entre TO, estos atributos serán referencias al objeto componente (en SQL3 se admiten referencias en un sólo nivel de anidamiento, aunque se prevé eliminar esta restricción en el SQL4). La meronimia entre tipos valor, se implementará con un atributo que contenga directamente el propio valor (las referencias entre valores no se soportan en ninguno de los dos modelos, si bien el SQL tiene previsto incluirlas en el SQL4, DBL:LHR-077 (1996)). Dependiendo del tipo de meronimia de que se trate tendremos las siguientes equivalencias:

- ***Miembro/colección***.- El atributo que la implementan es de tipo colección.

- ***Compuesto/componente***.- El atributo que la implementa es una estructura. Si algún componente es un grupo repetitivo, este será un vector en ODMG-93 y una lista en SQL3.

En MIMO la meronimia puede combinarse con restricciones. Estas restricciones pueden recogerse en SQL3 mediante la definición de restricciones, aserciones o disparadores. En ODMG-93, esta semántica sólo pude implementarse mediante operaciones.

La **generalización** se soporta directamente en los dos estándares, permitiéndose herencia simple y múltiple. En ODMG-93 cada ejemplar tiene un tipo más específico; en SQL3 no. Esto implica que la generalización soportada por el primero es la del MU V0.8 (la exclusiva) y la soportada por el segundo es la misma que soporta MIMO (la solapada). En SQL3, en un principio, se mantenía también el concepto de tipo más específico sin embargo, en la actualidad tal restricción se ha eliminado debido a que los problemas que plantea, Melton (1994) y que ya han sido expuestos en el epígrafe 4.4.4. Hay que señalar, que aunque la generalización soportada por el MU V0.8 era la exclusiva, en la actualidad, UML V1.0 soporta ya generalización solapada.

En SQL3 es posible modificar el tipo de generalización básica añadiendo restricciones al igual que se hace en MIMO. Sin embargo, esto no es posible en ODMG-93 debido a que este estándar no define restricciones. ODMG-93 soporta directamente la generalización exclusiva, y la solapada, al igual que en el MU V0.8, combinando una generalización con herencia múltiple.

ODMG-93 incluye los tipos abstractos cuyo concepto es equivalente al de clase diferida de MIMO. Es decir, un tipo abstracto en ODMG-93 es aquel que no puede ser ejemplificado directamente. En SQL3 no es posible definir clases abstractas ni diferidas. Por este motivo, en SQL3, una generalización total (aquella cuya superclase es abstracta) deberá implementarse mediante restricciones. En ODMG-93 la generalización total se implementaría definiendo el supertipo como abstracto; sin embargo, el ODMG-93 no especifica la sintaxis que permita esta definición. En nuestra opinión, deberá hacerse a través de su vinculación con el C++, definiendo una clase con operaciones diferidas.

Para finalizar se muestra, tabla 5.4, un resumen de la correspondencia entre las interrelaciones del nivel de construcción de MIMO con las de los modelos de los estándares:

MIMO	SQL3	ODMG-93
Clase de interrelación binaria sin características propias	Atributos de referencia (REF)	Atributos de referencia (relationship)
Todas las demás clases de interrelación	Un nuevo TO con atributos de referencia	Un nuevo TO con atributos de referencia
Meronimia miembro/colección	Atributo de tipo colección	Atributo de tipo colección
Meronimia compuesto/componente	Atributo de tipo estructura	Atributo de tipo estructura
Meronimia compuesto/componente + restricción de exclusividad	Atributo de tipo estructura+ aserción	Atributo de tipo estructura
Generalización	Generalización	Generalización con herencia múltiple
Generalización + restricción de exclusividad	Generalización + aserción	Generalización
Generalización con superclase abstracta (no necesariamente diferida)	Generalización + cláusula CHECK en el supertipo	Generalización con supertipo abstracto[14]

Tabla 5.4: correspondencia de interrelaciones

[14] Recuérdese que el concepto de clase abstracta en ODMG-93 coincide con el concepto de clase diferida en MIMO.

6 Conclusiones

"El último paso de la razón es conocer que hay infinitas cosas que la superan"

Pascal

En este capítulo se analizan los resultados obtenidos contrastándolos con los objetivos fijados en su comienzo (6.1) y se resumen las principales aportaciones de la investigación realizada (6.2). El análisis de objetivos y aportaciones anteriormente realizado nos permitirá extraer conclusiones, delimitando hasta donde se ha llegado en la investigación y qué aspectos se han dejado sin solucionar. Los problemas no resueltos, junto con otros que se derivan de la propia investigación, dan lugar a puntos de partida para nuevas investigaciones y trabajos futuros (6.3).

6.1. Análisis de la consecución de objetivos

Al comienzo del presente trabajo (epígrafe 1.2) se plantearon una serie de objetivos parciales cuyo fin era llegar a alcanzar el objetivo fundamental de esta investigación: la definición de MIMO. Se analiza ahora el resultado de cada uno de los objetivos parciales:

Objetivo 1: Analizar los distintos modelos de objetos existentes en la actualidad seleccionando aquellos que habrían de ser integrados en MIMO

De los modelos estudiados se han seleccionado diecinueve (epígrafe 3.2) de acuerdo a los siguientes criterios: *importancia* del modelo (determinada por su ámbito de aplicación), *adecuación* de éste a nuestros objetivos (fases del ciclo de vida que soporta, tipo de aplicaciones a las que está orientado, así como su capacidad semántica) y *bondad* (compleción, sencillez y consistencia). Para su exposición, los diecinueve modelos seleccionados se han dividido en dos grupos según su influencia en MIMO. Aunque de todos ellos se han destacado sus principales aportaciones y limitaciones, se han expuesto con mayor profundidad los modelos que más influencia han tenido en MIMO. El estudio se ha centrado en la parte estática de los modelos.

La selección de los modelos que deberían ser integrados en MIMO se ha realizado de acuerdo a los criterios de importancia y adecuación expuestos anteriormente. Se han seleccionado los modelos de implementación más importantes (los estándares SQL3 y ODMG-93) y el modelo de un método de análisis y diseño, el MU, que integra las dos metodologías de orientación al objeto más extendidas en la actualidad (Booch y Rumbaugh). Se ha seleccionado también el modelo de MEDEA, con el fin de incorporar aspectos de bases de datos que el resto de los modelos no tratan.

Una de las principales dificultades encontradas en la realización de esta tarea ha sido las continuas actualizaciones de los modelos seleccionados. ODMG-93, desde que

se comenzó este trabajo, Cattell (1994a), tan sólo ha sufrido una revisión, Cattell (1995), y con muy pocas modificaciones en el modelo. Sin embargo, del MU hemos conocido cuatro revisiones, MU V0.8, UML V0.9, UML V0.91 y UML V1.0. Peor ha sido del caso del SQL3 donde ya no hay que hablar de cuatro revisiones, sino de cuatro modelos totalmente distintos con un número incontable de sucesivas revisiones. Además, gran parte de la documentación del SQL3 es confidencial, por lo que su seguimiento nos ha obligado a participar en diversas reuniones del grupo, Southampton (julio de 1994), Londres (enero de 1996), Kansas-City (mayo de 1996) y Madrid (enero-febrero de 1997).

Objetivo 2: Definición de un modelo de objetos (MIMO) que soportara los niveles de abstracción de cada una de las fases del desarrollo de una base de datos integrando los modelos de objetos previamente seleccionados y aumentando la capacidad semántica proporcionada por ellos sin penalizar excesivamente su complejidad

MIMO (capítulo 4) integra, el MU en su fase de análisis; el MU, SQL3 y ODMG-93 en construcción; SQL3 y ODMG-93 en explotación. Su capacidad semántica es mayor que la de cualquiera de estos tres modelos ya que MIMO subsume a los tres, además de incluir facilidades de modelado importantes, algunas tomadas de diferentes modelos y otras propias, entre las que cabe destacar la posibilidad de interrelacionar TI. A fin de no obtener un modelo excesivamente complicado, MIMO proporciona una serie de constructores primitivos que pueden ser ampliados mediante la aplicación de restricciones a los TI o mediante la generación de nuevos TD.

Aunque sí se consideran algunos aspectos de la dinámica, el modelo se centra en la parte estática, por lo que no puede considerarse un modelo completo. Existen otros aspectos semánticos importantes que no han sido estudiados en profundidad, como la posibilidad de definición de papeles o la migración de objetos, pero que sí son tenidos en cuenta en la definición de modelo con el fin de poder incorporarlos posteriormente.

Para integrar los conceptos de los tres modelos que MIMO aglutina se han tenido en cuenta, ordenados según prioridad, los siguientes criterios:

1. En primer lugar se ha priorizado MIMO como modelo, es decir, MIMO debería:

   - proporcionar una gran capacidad expresiva en todos sus niveles, equilibrando esta capacidad semántica con el grado de complejidad
   - mantener la continuidad en el modelado entre cada uno de los niveles de abstracción que soporta

2. Consistencia de MIMO en sí mismo

3.- Integración de los modelos del MU, SQL3 y ODMG-93

Objetivo 3: Definición de una notación gráfica que soportara todos los constructores del modelo

Dicha notación (apéndice I), se basa en las propuestas para el MU, OMT y MEDEA. En general, se ha tratado de mantener la compatibilidad con los modelos existentes, por lo que los conceptos de MIMO que tienen una representación en el MU, o en su defecto en OMT, han mantenido su notación. Para aquellos conceptos sin representación en el MU ni en OMT, se ha mantenido la notación de MEDEA. Y, por último, para los conceptos característicos de MIMO se ha definido una notación nueva procurando mantener la homogeneidad de representación.

Objetivo 4: Validación y verificación del modelo

Para la validación y verificación del modelo se propusieron los siguientes objetivos parciales:

Objetivo 4.1: Definición de una primera aproximación a la correspondencia entre las estructuras de la fase de construcción de MIMO y las estructuras del ODMG-93 y el SQL3

Se ha realizado (capítulo 5) la aproximación a la correspondencia de estructuras entre MIMO y los modelos del ODMG-93 y el SQL3, unidireccional en el sentido indicado (MIMO ----> SQL3, MIMO ----> ODMG-93), lo que ha permitido validar, y verificar parcialmente, una de las hipótesis: que MIMO soporta el nivel de construcción y que efectivamente integra los modelos de los estándares mencionados. Queda abierta la posibilidad de completar la correspondencia de estructuras, de tal forma que ésta pueda realizarse desde la fase de análisis de MIMO (lo que incluye la definición de una metodología de desarrollo en MIMO) y de modo bidireccional. Esto permitirá (ver capítulo 5) la traducción de código SQL3 a ODMG-93 y viceversa, siempre a través de la fase de construcción de MIMO.

Según la hipótesis planteada, MIMO debería incluir el modelo del MU. Su validación se realiza en el capítulo 4 (epígrafe 4.4.4) donde se estudia la correspondencia de MIMO con la del MU, lo que permite, además, verificar parcialmente la capacidad de MIMO como modelo conceptual.

Objetivo 4.2: Implementación de un prototipo de MIMO como modelo de una herramienta de modelado conceptual OO

Se han implementado las funcionalidades básicas de dicha herramienta (ver 5.2). Dicha herramienta, MIMO-CASE, ha servido para verificar la parcialmente la capacidad de MIMO como modelo conceptual.

Objetivo 4.3: Descripción recursiva de MIMO en términos de si mismo

Apoyándonos en MIMO-CASE, se ha descrito MIMO recursivamente, en términos de MIMO (5.3) lo que permite verificar en parte la capacidad de MIMO para el modelado conceptual. Dicha descripción es, a su vez, el esquema de la metabase de ENEAS/BD. en MIMO.

Objetivo 4.4: Implementación de un prototipo de MIMO como modelo de la metabase de una herramienta de diseño de bases de datos avanzadas

La implementación permite verificar parcialmente la consistencia de MIMO y ha supuesto refinamientos en el modelo derivados de inconsistencias encontradas durante su implementación. Se ha presentado el metaesquema completo de MIMO, el diseño de la parte correspondiente al nivel de análisis y se ha implementado la parte más distintiva del nivel de análisis (TO y TI).

De todo lo dicho podemos concluir que se ha alcanzado el **objetivo principal** del presente trabajo: definir un modelo de objetos que soportara todos los niveles de abstracción del desarrollo de una base de datos, integrando los principales modelos de objetos de la actualidad y aumentando la capacidad semántica respecto a los modelos existentes sin penalizar excesivamente su complejidad. Sin embargo, somos conscientes de que MIMO es mejorable y ampliable.

6.2. Principales aportaciones de la investigación

Creemos que el trabajo realizado ha supuesto una serie de aportaciones relativas, no sólo al tema central de la investigación, modelos de objetos, sino también a la labor de investigación en sí misma. Algunas de estas aportaciones eran objetivos previos a la elaboración del trabajo; otras se han obtenido a partir de los estudios y reflexiones necesarios para la consecución de las primeras. Resumimos aquí las aportaciones que consideramos más relevantes:

1. No cabe duda que la aportación principal es la elaboración de un **modelo de objetos que soporta el modelado en cada una de las fases del desarrollo de una base de datos**, algo que ninguno de los modelos estudiados permite (se han analizado todo tipo de modelos, pero no se ha encontrado ninguno que soporte los conceptos de las diferentes fases en el desarrollo de una base de datos). Tal característica permite que un desarrollo en MIMO pueda ser realizado de un modo uniforme y sin transiciones traumáticas entre una fase y otra.

2. MIMO **aumenta la capacidad semántica** de los modelos estudiados, no sólo en el modelado conceptual, soportando el concepto de **interrelaciones entre tipos de interrelación**, sino también en **construcción** donde se mantienen los conceptos de interrelación, generalización y meronimia.

En MIMO **un TI nivelada es un subtipo de un TO**. Esta clasificación, característica de MIMO, hace que cualquier concepto aplicable a un TO (v.g. restricciones) es igualmente aplicable a los TI niveladas, lo que cambia substancialmente la semántica del modelo con respecto a los demás modelos de objetos.

3. Se ha logrado un **equilibrio entre potencia expresiva y complejidad**. Para ello, MIMO soporta:

   - Un conjunto de Tipos de Datos básicos, ampliables mediante generadores de tipos de datos

   - Un conjunto de Tipos de Interrelación primitivos, ampliables mediante la aplicación de una serie de Tipos de Restricción

4. Se ha contribuido a **esclarecer los conceptos de *clase abstracta* y *clase diferida*,** Marcos et al. (1997b), utilizados confusamente en la literatura. En MIMO una clase diferida y una clase abstracta son clases no ejemplificables pero, mientras la clase diferida es una noción de construcción, la clase abstracta se introduce en análisis (por lo que, en MIMO, se habla de tipo abstracto). Ambas tienen en común que sólo puede ejemplificarse a través de sus subtipos.

5. La definición de la correspondencia entre las estructuras de la fase de construcción de MIMO y las de los estándares, Marcos et al. (1997a), permite realizar la **traducción de una construcción en MIMO a código SQL3/ODMG-93**, así como sentar las bases para una futura conversión de código SQL3 a código ODMG-93 y viceversa, siempre a través de MIMO.

6. Se ha definido un **método de investigación**, adaptada al tema concreto del trabajo, basada en el método experimental (hipotético-deductivo) de investigación científica y el método general de trabajo en la ingeniería del software. El trabajo realizado ha seguido dicho método. Aunque esta aportación no es relativa al objetivo de esta Tesis, consideramos importante destacarla, porque creemos que sí constituye una aportación en cuanto al proceso de investigación en sí misma, dentro del ámbito de la ingeniería del software. No hay que olvidar, que el objetivo de una Tesis Doctoral no es sólo el de presentar aportaciones en un área de estudio, sino también el de realizar un trabajo de investigación que, si es riguroso, deberá seguir un método.

Además de las aportaciones resaltadas, se podrían señalar otros trabajos importantes, como la discusión a cerca del término de modelo de datos, precisando su significado y planteando la necesidad de una reflexión más profunda a cerca del concepto de modelo dentro del ámbito de la ingeniería del software; la clasificación rigurosa de los TI, ya que si bien la mayor parte de los modelos presentan clasificaciones para el Sistema de Tipos, el Sistema de Interrelaciones se presenta siempre de un modo menos formal y riguroso; y, por último la comparación realizada entre los diferentes tipos colección, que tampoco presenta ninguno de los modelos estudiados, nos ha permitido establecer la distinción conceptual entre los tipos vector y lista, argumentando porqué, en contra del planteamiento de todos los modelos estudiados, de un vector no es una colección.

6.3. Nuevas Líneas de investigación

A pesar de todas las aportaciones que creemos que supone esta investigación, existen varios aspectos en los que es necesario seguir trabajando. Algunos de ellos, debido a que su realización no se encontraba desde el inicio entre los objetivos previstos; otros aspectos sin resolver se deben a que el propio avance en la investigación va dando lugar a nuevos problemas y a nuevas necesidades. Resumimos aquí las principales líneas de investigación abiertas durante la realización del presente trabajo:

1. Completar el modelo con aspectos de **dinámica** que no han sido tenidos en cuenta en esta primera versión. Así, deberán contemplarse aspectos como el paso de mensajes, polimorfismo, vinculación con algún lenguaje que permita la implementación completa de los servicios, etc.

2. Ampliar el modelo para **soportar** los siguientes conceptos:

   - **Migración de objetos**. Aunque, como un primer paso, el modelo soporta un tipo especial de servicio que permite la movilidad de objetos de una clase a otra, no se han estudiado a fondo las repercusiones que de ello podrían derivarse.

   - **Tratamiento de papeles**. MIMO ha sido pensado para que ésta, y otras extensiones, puedan realizarse de un modo sencillo y sin variar la filosofía del modelo, tal y como explicamos en el apartado 3.5.

   - Estudiar en profundidad las repercusiones que pudieran derivarse de un modelo de **múltiple ejemplificación** ya que, aunque esta posibilidad está contemplada en MIMO, es también uno de los aspectos que requieren un mayor estudio.

   - **Actividad**, incluyendo en un primer momento el soporte de reglas activas (disparadores) que no se han considerado en esta primera versión.

3. Extender el modelo de objetos incluyendo aspectos relativos al tratamiento **temporal** de los datos, así como a la **seguridad** de éstos.

4. Realizar la **implementación completa del modelo** y analizar las ventajas, y desventajas si las hubiere, de un modelo que soporta uniformemente los conceptos de las distintas etapas del desarrollo de una base de datos.

5. **Refinar la correspondencia de estructuras** definida entre la fase de construcción de MIMO y los modelos de los estándares, hacerla bidireccional e

implementarla. De este modo se llegaría a la automatización de una construcción MIMO a un esquema en SQL3 o en ODMG-93 y, lo que es más importante, se conseguiría la automatización de la traducción de esquemas SQL3/ODMG-93 (siempre a través de la fase de construcción de MIMO).

6. Definir un **método para desarrollo** de bases de datos soportada por MIMO, que permita determinar los pasos a seguir para llegar de un análisis a una implementación MIMO.

7. **Implementar las transformaciones**, definidas en dicho método, que permitan obtener una construcción MIMO a partir de un análisis MIMO. La consecución de los puntos 5, 6 y 7 permitirá obtener a partir de un análisis en MIMO su correspondiente código en ODMG-93 o SQL3.

8. Definir e implementar la **correspondencia entre las estructuras de análisis MIMO y las del MU**. De este modo, un análisis MIMO, podrá integrarse en futuras herramientas CASE que soporten el MU (en el caso de que este método se impusiera como estándar[1]). En el capítulo 3 se realiza un estudio comparativo del sistema de interrelaciones de MIMO con el del MU (lo que constituye el núcleo del modelo de objetos del MU) como un primer paso en esta dirección.

9. Definir e implementar un **lenguaje de definición de datos** que permita describir esquemas MIMO (esquemas conceptuales, así como de construcción). Se ha definido ya una primera versión de dicho lenguaje (dentro del módulo de *modelado OO* de ENEAS/BD) que contempla algunos de los constructores básicos de MIMO.

10. Definir e implementar un **lenguaje de manipulación de datos**

[1] Como ya se ha dicho, la última versión del MU, UML V1.0, considerada ya estable, está siendo estudiada por el OMG para ser aprobada como futuro estándar.

Para finalizar, queremos resaltar la importancia de la definición de un nuevo modelo que **soporta de un modo uniforme los distintos niveles de abstracción** del desarrollo de una base de datos; si la orientación al objeto difumina las fronteras entre cada una de las fases, no cabe duda que un modelo de tales características constituyen un avance en esta misma dirección ya que permite que la transición de una fase a otra en el desarrollo de una base de datos se realice de un modo uniforme. Además, aunque la integración de modelos es una de las principales propuestas para solucionar el problema de heterogeneidad existente en la actualidad, está propuesta está enfocada a la convergencia de modelos que corresponden a una misma fase del ciclo de vida, por lo que la integración de modelos de análisis con modelos de implementación es un paso más allá en las tendencias actuales. Por otra parte, como ya se ha explicado, los trabajos de integración de modelos de construcción están más orientados a la convergencia o entendimiento de éstos que a su unificación.

La posibilidad de traducción de esquemas SQL3 a esquemas ODMG-93 y viceversa que se deriva del presente trabajo, constituye un avance importante en la línea de la portabilidad. Los resultados obtenidos en este área, junto con los del Object Merger Group (más en la línea de la convergencia de los lenguajes de manipulación de estos dos estándares), contribuirán de modo importante a **mejorar las posibilidades de interoperabilidad y portabilidad** de las bases de datos de las próximas décadas.

Queremos también hacer hincapié en la importancia de las líneas de trabajo abiertas, ya que permiten determinar los problemas que aún no se han resuelto y de este modo marcar la dirección que puedan tomar nuevas investigaciones.

Apéndice I: Glosario de términos

Aserción: restricción de verificación que se define sobre dos o más tipos/clases.

Atributo: característica de un tipo/clase que, junto con sus otros atributos, definen la estructura de los ejemplares de dicho tipo/clase.

Atributo base: atributo cuyo valor se asigna, y obtiene, de modo directo.

Atributo derivado: atributo cuyo valor se calcula o infiere a partir de otro u otros atributos.

Atributo real: atributo que almacena su valor.

Atributo virtual: atributo cuyo valor se obtiene en el momento de la recuperación.

Característica: cada una de las propiedades de un tipo/clase que, en conjunto, permiten diferenciarlo de los demás tipos/clases.

Características estáticas: son característica estructurales, de relacionamiento, de integridad.

Características estructurales: características de un tipo, o clase, que definen su estructura y su identidad.

Características de ejemplar: características que afecta de forma distinta a cada ejemplar.

Características de integridad: características de un tipo, o clase, que definen las restricciones semánticas de sus ejemplares.

Características de relacionamiento: características de un tipo/clase que definen los ejemplares de los tipos/clases con los que pueden asociarse sus ejemplares.

Características de tipo/clase: características que afectan por igual a todos los ejemplares del tipo/clase.

Características dinámicas: características de un tipo, o clase, que definen el comportamiento de cada uno de sus ejemplares.

Características privadas: características que sólo son accesible por los servicios del propio tipo.

Características protegidas: características accesibles por los servicios del tipo y por los de sus subtipos.

Características públicas: características accesibles por todo objeto que tenga privilegios para ello.

Clase: implementación de un tipo (valor u objeto).

Clase abstracta: aquella que no tine una correspondencia directa con la definición de ningún ente identificable del universo del discurso y que se introduce en el esquema con el fin de aumentar su nivel de abstracción. Una consecuencia directa de esta definición es que una clase abstracta no podrá ser nunca ejemplificada directamente.

Clase de Interrelación (CI): construcción (diseño e implementación) de un TI nivelada.

Clase diferida: *aquella que delega la implementación de alguno de sus servicios a sus subclases y que se introduce a fines de reutilización*, Meyer (1988). Una consecuencia directa de esta definición es que una clase diferida no podrá ser nunca ejemplificada directamente.

Clave ajena: atributo o conjunto de atributos que son clave primaria, o alternativa, en otro tipo de objetos. La clave ajena debe verificar siempre la *restricción de integridad referencial*.

Clave primaria: atributo o conjunto de atributos que identifican unívoca y mínimamente los ejemplares de un tipo de objetos o de la clase que lo implementa.

Condición de excepción: es una condición que se asocia a un servicio de tal modo que, cuando se verifique dicha condición, se realizará la acción o acciones definidas en la excepción.

Constructor: servicio que crea objetos.

Copiador: servicio que reproduce objetos. El objeto reproducido es idéntico al original.

Destructor: servicio que destruye objetos.

Dominio: grupo nominado y homogéneo de valores de un cierto tipo de datos a los que se puede aplicar una restricción; esta restricción puede ser, bien la enumeración de un subconjunto de los valores del tipo sobre el que se define, bien un predicado que deben verificar todos los valores del tipo sobre el que se define.

Ejemplar de una clase: cada uno de los objetos o valores de una clase.

Ejemplar de una CI: cada una de las interrelaciones de una CI.

Ejemplar de un TI: cada uno de las interrelaciones que satisface un TI.

Ejemplar de un TO: cada uno de los objetos que satisfacen un TO.

Ejemplar de un TV: cada uno de los valores que satisfacen un TV.

Ejemplificable: con capacidad para ser ejemplificado.

Ejemplificar: acción de crear un ejemplar. Si el ejemplar es un objeto la ejemplificación se realizará a través de un constructor.

Estado de un objeto: conjunto de valores que toman los atributos de un objeto en un momento dado y conjunto de interrelaciones que mantiene en ese mismo momento con otro u otros objetos.

Esquema: representación formal de un universo del discurso, generalmente, en un sistema informático. Según Dittrich (1994), *descripción específica de un mini-mundo en términos de un modelo de datos*. En 3.1.1 se discute el significado de esquema en relación con el de modelo de datos y formalización.

Extensión de clase: conjunto de ejemplares de una clase en un momento dado. La extensión de una clase es un subconjunto de la extensión del tipo al que implementa.

Extensión de tipo de datos: conjunto de ejemplares que satisfacen dicho tipo de datos.

Extensión real (de tipo o de clase): extensión definida por enumeración.

Extensión virtual (de tipo o de clase): extensión definida a través de algún predicado, o rango que deben satisfacer los ejemplares de dicha clase.

Formalizar: según la Real Academia Española: *"concretar, precisar. Dar carácter de seriedad a lo que no la tenía"*. ***Formalizar un modelo***, ver discusión en 3.1.2.

Generalización: construcción de un TI de generalización que representa una clasificación en la que las subclases son una especialización de cada una de sus superclases (herencia simple y múltiple). En una generalización cada subclase hereda todas las características de sus superclases, además de poder añadir características propias (herencia de especialización) o redefinir las heredadas.

Generalización de herencia: es aquella en la que los ejemplares de la subclase no son ejemplares de la superclase y que se define con el único fin de reutilizar características. No existe el concepto de TI de generalización, ya que la generalización de herencia es siempre una noción de construcción.

Generalización de papeles: es aquella en la que las subclases especifican los posibles cometidos que pueden desempeñar los objetos de la superclase durante su período de vida.

IDentificador de Objeto (IDO): característica estructural de un objeto, asignada por el sistema, que se vincula con éste mediante una conexión fija uno-a-uno y que es independiente del resto de sus características.

Interrelación nivelada: representación, en un sistema de información, de un ente del mundo real que es susceptible de conocimiento en sí mismo y cuya existencia depende de la existencia de otros entes a los que relaciona.

Metabase: es una base de datos cuyos datos, metadatos, constituyen la descripción de una base de datos.

Metamodelo: modelo que describe modelos. ***Metamodelo de datos***, modelo de datos que describe modelos de datos.

Migración: acto por el cual un objeto deja de ser ejemplar de una clase para pasar a ser ejemplar de otra.

Migrador: servicio que mueve un objeto de una clase a otra.

Mini-mundo: parcela del mundo real susceptible de conocimiento. En bases de datos, es aquella parcela del mundo real que debe ser representada en un sistema informático. Ver *universo del discurso*.

Modelar: según la Real Academia Española: *"(de modelo) configurar o conformar algo no material. Ajustarse a un modelo*". En el caso de las bases de datos, modelar es utilizar un modelo de datos para definir un esquema. Ver 3.1.1

Modelo: según la Real Academia de la Lengua Española: "*en las obras de ingeniero y en acciones morales, ejemplar que por su perfección se debe seguir e imitar"* (acepción 1).// "*Esquema teórico, generalmente en forma matemática, de un sistema o de una realidad compleja (por ejemplo la evolución económica de un país), que se elabora para facilitar su comprensión y el estudio de su comportamiento"* (acepción 2). Estas dos acepciones, trasladadas al ámbito de las bases de datos, se discuten en 3.1.1

Modelo de datos: según Dittrich (1994), "*es un conjunto de herramientas conceptuales para describir la representación de la información en términos de datos. Comprende aspectos relacionados con tipos y estructuras de datos, operaciones y restricciones"*. En 3.1.1 se discute el significado de este término en relación con el de modelo y esquema.

Múltiple ejemplificación: se dice que un modelo soporta múltiple ejemplificación, cuando un objeto puede pertenecer simultáneamente a dos o más clases (y a sus tipos correspondientes).

Modificadores: servicio que cambia el estado de un objeto. Ver *Mutador*.

Mutador: servicio que cambia el estado de un objeto. Ver *Modificador*.

Objeto: representación, en un sistema de información, de un ente del mundo real que es susceptible de conocimiento en sí mismo.

Observador: servicio que accede a un objeto sin alterar su estado. Ver *Selector*.

Postcondición: predicado asociado a un servicio que tendrá que verificarse a la finalización de éste.

Precondición: predicado asociado a un servicio que tendrá que verificarse antes de iniciarse la ejecución de éste.

Reconstructor: servicio que recupera objetos previamente destruidos. Los objetos se recuperan con el estado que tenían en el momento de su destrucción.

Restricción: limitación impuesta a la estructura o a los datos.

Restricción de cardinalidad: se define como el número mínimo y máximo de veces en las que cada ejemplar de un tipo está asociado con un ejemplar del otro tipo en una interrelación.

Restricción de dependencia en existencia: se dice que el tipo T1 depende en existencia del tipo T2, cuando la existencia de todo ejemplar de T1 está condicionada a la existencia de un determinado ejemplar de T2, de forma que si desaparece el ejemplar de T2, el ejemplar (o ejemplares) de T1 asociado(s) desaparece(n) con él.

Restricción de dependencia en identidad: se dice que el tipo de objeto TO1 depende en identidad del tipo de objeto TO2, cuando el IDO de todo ejemplar de TO1 es igual que el IDO de algún ejemplar de TO2.

Restricción de dependencia en identificación: se dice que el tipo T1 depende en identificación del tipo T2, cuando los ejemplares de T1 no se identifican por sí mismos, sino que necesitan un ejemplar de T2 para poder identificarse.

Restricción de exclusión: sean dos TI niveladas, TI1 y TI2, establecidas entre TO1 y TO2. Se dice que entre TI1 y TI2 existe una restricción de exclusión si para toda combinación de interrelaciones de TI1 entre objetos de TO1 y TO2, dicha combinación no existe entre las interrelaciones de TI2.

Restricción de exclusividad: sea T1 un tipo que participa en dos tipos de interrelación, TI1 y TI2; se dice que existe una restricción de exclusividad si cada ejemplar de T1 puede participar en TI1 o en TI2, pero nunca en ambas a la vez.

Restricción de inclusión: sean dos TI niveladas, TI1 y TI2, establecidas entre TO1 y TO2. Se dice que TI1 tiene una restricción de inclusión respecto de TI2 si para toda combinación de interrelaciones de TI1 entre objetos de TO1 y TO2, dicha combinación debe existir necesariamente entre las interrelaciones de TI2.

Restricción de integridad: es un predicado o proposición que toma los valores verdadero o falso para uno o varios ejemplares.

Restricción de integridad referencial: implica que todo valor de una clave ajena debe tener correspondencia con un valor de la clave primaria, o alternativa, a la que dicha clave ajena hace referencia, o tener valor nulo.

Restricción de no nulidad: restricción de integridad que fuerza a que el atributo al que se aplique tenga que tener un valor asignado. Si la restricción de no nulidad se define

sobre un conjunto de atributos, esto significa que, al menos uno de ellos, deberá tener un valor asignado.

Restricción de orden: sea TI1 un TI que se establece entre dos tipos, T1 y T2. Se dice que existe una restricción de orden de T2 respecto de T1 cuando los ejemplares de T2 participan en TI1 en un orden concreto, que se determina en función de uno o varios atributos de T2.

Restricción de unicidad: restricción de integridad que impide que dos ejemplares pueden tener el mismo valor en el atributo, o los atributos, a los que se aplique.

Restricción de verificación: restricción de integridad que se define sobre uno o varios atributos del tipo/clase.

Selector: servicio que accede a un objeto sin alterar su estado. Ver *Observador*.

Servicio: características de un tipo/clase que, junto con sus otros servicios, definen el comportamiento de los ejemplares de dicho tipo/clase.

Tipo abstracto: aquel que no tiene una correspondencia directa con ningún ente identificable del universo del discurso y que se introduce en el esquema con el fin de aumentar su nivel de abstracción. Una consecuencia directa de esta definición es que un tipo abstracto no podrá ser nunca ejemplificado directamente.

Tipo agregado: se llama tipo agregado a un tipo estructura, colección o vector.

Tipo básico: aquel que no se puede construir a partir de ningún otro.

Tipo colección: tipo de datos con un único elemento el cual puede tomar un número variable de valores todos ellos del mismo tipo.

Tipo compuesto: tipo que describe elementos no atómicos.

Tipo conjunto: tipo colección en el que los valores que lo componen no tienen orden, ni pueden estar duplicados.

Tipo de datos: es, bien un tipo de objeto, bien un tipo valor.

Tipo de Interrelación (TI): asociación que se establece, bien entre tipos (objeto o valor) bien entre clases.

TI de generalización: TI jerárquica entre tipos (valor u objeto) que representa una clasificación en la que los subtipos son una especialización de cada uno de sus supertipos (herencia simple y múltiple). En una jerarquía de generalización cada subtipo hereda todas las características de sus supertipos, además de poder añadir características propias (herencia de especialización) o redefinir las heredadas.

TI de generalización de papeles: es un TI de generalización en el que los subtipos especifican los posibles cometidos que pueden desempeñar los objetos de los supertipos durante su período de vida.

TI de meronimia: es un TI que corresponden al concepto PARTE-DE y que se establecen entre tipos (valor u objeto), llamándose holónimo (todo) al tipo compuesto y merónimo (parte) a cada uno de los tipos componentes.

TI de meronimia compuesto/componente: TI de meronimia en la que los componentes realizan una función concreta en relación a otros componentes o al todo (por ejemplo, la rueda y el coche).

TI de meronimia Física: TI de meronimia en la que una "parte" no puede pertenecer a más de un "todo".

TI de meronimia Lógica: TI de meronimia en la que una "parte" puede encontrarse en diferentes "todos"; la parte funciona como un modelo de un catálogo.

TI de meronimia miembro/colección: TI de meronimia en la que no existe relación funcional entre el todo y las partes (por ejemplo, un árbol y el bosque al que pertenece).

TI de múltiple generalización: es una agregación de TI de generalización de un mismo tipo que se especializa simultáneamente por diversos criterios.

TI jerarquizadas: es un TI en el que los participantes mantienen algún tipo de subordinación.

TI nivelada: (1) es un TI de igual a igual que tienen lugar entre TO. **(2):** es un TO que asocia a dos o más TO y en el que la existencia de cada ejemplar depende de la existencia de los ejemplares de los TO que asocia.

Tipo de Restricción (TR): plantilla de predicado o proposición para uno o varios ejemplares genéricos de un tipo (es decir, que aún no se han especificado). Cuando la plantilla del predicado se completa y se especifica para qué ejemplar o conjunto de ejemplares debe verificarse, se obtiene una restricción.

Tipo enumerado: grupo de valores simples que no se toman de ningún tipo básico, sino de una lista definida por el usuario.

Tipo estructura: tipo de datos compuesto por un número fijo de elementos que no tienen por qué ser del mismo tipo.

Tipo específico: tipo de datos definido por el usuario.

Tipo generador: es un tipo, no ejemplificable directamente, que permite construir distintos tipos específicos de datos.

Tipo lista: tipo colección en el que los valores que la componen tienen un orden y pueden estar duplicados.

Tipo multiconjunto: tipo colección en el que los valores que la componen no tienen orden y pueden estar duplicados.

Tipo Objeto (TO): descripción de un conjunto de objetos que tienen las mismas características .

Tipo plantilla: es un tipo generador que permite construir tipos de datos por ejemplificación.

Tipo primitivo: tipo de datos que pueden ser suministrados bien por el sistema, bien por el administrador.

Tipo simple: tipo que describe elementos -siempre valores- atómicos.

Tipo Valor (TV): descripción de un conjunto de valores que tienen las mismas características .

Tipo Vector: tipo de datos compuesto por un número fijo de elementos todos ellos del mismo tipo.

Universo del discurso: parcela del mundo real susceptible de conocimiento. En bases de datos, es aquella parcela del mundo real que debe ser representada en un sistema informático. Ver *mini-mundo.*

Valor: representación de una característica estructural de un objeto. El valor pueden describirse en función de otros valores dando así lugar a un **valor compuesto**.

Apéndice II:
Glosario de siglas

ADT	Abstrac Data Type. Los ADTs son especialmente importantes en el modelo de datos del SQL3.
ANSI	American National Standard Institute. ANSI/X3H2 es el grupo de trabajo de ANSI que se ocupa de la estandarización del SQL3.
BD	Base de Datos
BOD	Comité Técnico del OMG que se encarga de tomar las decisiones finales respecto a las propuestas de estándares. *Ver TC*.
B.O.E.	Boletín Oficial del Estado.
COMMA	Common Object Methodology Architecture. Es un proyecto para la definición de un metamodelo válido para cualquier metodología, Henderson-Sellers (1996), (3.2.8.3).
CORBA	Common Object Request Architecture. Arquitectura de ORB propuesta por el OMG, (3.2.1).
CD	Committee Draft. Es uno de los estados por los que tiene que pasar un borrador de estándar de ISO antes de su aprobación como norma internacional.
ENEAS/BD	ENtorno para la Enseñanza Avanzada de Bases de Datos, (1.3).
E/R	Entidad/Relación.
EXPRESS	Lenguage para el modelado de información en sistemas de automatización industrial, desarrollado por ISO TC184/SC4/WG5. El WG5 recibe el nombre de STEP Development Methods. *Ver STEP*. (3.3.6).
IDEA	Intelligent Database Environment for Advanced Applications. Proyecto ESPRIT desarrollado por el Politécnico de Milano, dentro de cuyo marco de define el modelo de Chimera. (3.2.8.5).
IDL	Interface Definition Language. Es el lenguaje de definición de interfaces propuesto por el OMG para CORBA.
IDO	IDentificador de Objetos, (4.2, Definición 10).
IGES	Initial Graphics Exchange Specification. *Ver IPO.*

IPO Organización IGES/PDES encargada de gestionar la contribución de los estándares de intercambio de modelos de datos de productos (Standard for the Exchange of Product data model, -STEP-). *Ver IGES y PDES.*

ISO International Standarization Organization.
ISO/IEC JTC1/SC21/WG3 es el grupo de trabajo de ISO que se ocupa de la estándarización del SQL3.

MEDEA Metodología de Desarrollo de bAses de datos orientada al objeto, Piattini (1994), (3.2.7).

MIMO Metamodelo para la Integración de Modelos de Objetos. Metamodelo objeto de la presente Tesis Doctoral, cuyas características principales son permite el modelado tanto a nivel de análisis como a nivel de construcción (diseño e implementación); integra los modelos del SQL3 y el ODMG-93 (en construcción) y el del MU en análisis y construcción ; aumenta la capacidad semántica de cada uno de los modelos que integra, (4).

MOOSE Major Object-Oriented SQL Extension. En DBL ARL-078 (1991) recoge las primeras extensiones de objetos del SQL.

MU Método Unificado (Unified Method, -UM-), Booch and Rumbaugh (1995), (3.2.4). *Ver UML*.

MU/OM Es el modelo de objetos del método unificado (MU).

OADTF Object Analysis & Design Task Force. Es un grupo de trabajo del OMG cuya misión, entre otras, es la de trabajar en metamodelos. *Ver TF*.

ODL Object Language Definition. Lenguaje de definición del ODMG-93. Su sintaxis está basada en la del IDL de CORBA .

ODMG Object Database Management Group.

ODMG-93 Estándar para sistemas de bases de objetos definido por el ODMG, (3.2.2).

ODMG/OM Object Data Management Group/Object Model. Es el modelo de objetos del ODMG-93, (3.2.2).

OID Object IDentifier. *Ver IDO.*

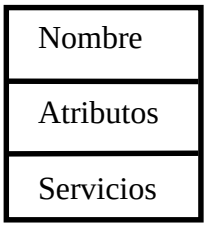
OMA Object Management Arquitecture. Es un arquitectura para

	entornos distribuidos definida por el OMG, (3.2.1).
OMFT	Object Model Task Force, (3.2.1). *Ver OMG y TF*.
OMG	Object Management Group. Se estructura en Comités Técnicos, Grupos de Trabajo y Grupos de Interés Especial, (3.2.1). *Ver TC, TF, SIG*.
OMG/OM	Object Management Group/Object Model. Es el modelo de objetos propuesto por el OMG y que se denomina *Core Object Model*, (3.2.1).
OMT	Object Modelling technique. Metodología de análisis y diseño orientado al objeto, Rumbaugh et al. (1991).
OO	Orientación al Objeto.
OOD	Object-Oriented Design. Metodología de diseño orientado al objeto, Booch (1994).
OOM	Object Oriented MERISE. Metodología MERISE orientada al objeto, Rochfeld (1992), (3.2.5).
OOram	Object Oriented Role Analysis and Modelling. Metodología de modelado y análisis orientado al objeto basada en roles, Wold y Lehne (1996), (3.2.8.2).
OQL	Object Query Language. Lenguaje de Consulta de Objetos del ODMG-93.
ORB	Object Request Broker. Es el manejador de objetos de OMA.
OS	Object Service. Es el servicio de objetos de OMA.
PDES	Product Data Exange using STEP. *Ver IPO.*
PDM	PROBE Data Model. Modelo de datos de un SGBD orientado al conocimiento denominado PROBE, Manola y Dayal (1994), (3.2.8.11).
RAE	Real Academia Española.
ROSES	Rules Objects and Events. Es un Sistema de Gestión de Bases de Conocimientos, Costa et al. (1996), (3.2.8.7).
SGBD	Sistema de Gestión de Base de Datos.

SGBO	Sistema de Gestión de Base de Objetos.
SGBDOO	Sistema de Gestión de Base de Datos Orientada al Objeto.
SIG	Special Interest Group. Grupo de Interés Especial del OMG.
SQL3	Structured Query Language. Futuro estándar para bases de datos relacionales, extendidas con capacidades de orientación al objeto. Desarrollado por ISO/IEC JTC1/SC21 WG3, (3.2.3).
STEP	Standar for the Exchange of Product model data. *Ver IPO.*
SUMM	Semantic Unification Mete-Model. Es un metamodelo que integra diferentes modelos desarrollado por el Dictionary/Methodology/Committee del IGES/PDES de IPO, (3.3.1). *Ver IPO.*
TC	Technical Committee. Comité Técnico del OMG.
TF	Task Force. Grupo de Trabajo del OMG. *Ver OMTF*.
UDT	User Defined Types. Son tipos de datos definidos por el usuario. Su incorporación al SQL3 (entre los años 1988-89) constituye el primer acercamiento del SQL3 hacia la Orientación al Objeto.
UML	Unified Modelling Language. Lenguaje para el modelado de información cuyo modelo se basa en el propuesto para el MU, Booch et al. (1997) (3.2.4).

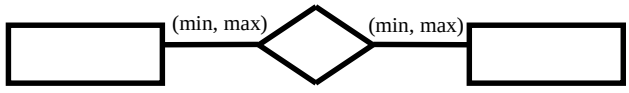
Apéndice III:
Notación gráfica de MIMO

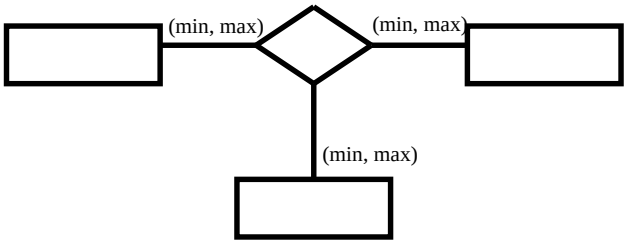
a) TIPO DE OBJETO



La especificación de los tipos de los atributos (TIPOS VALOR), de la cabecera de los servicios y de las restricciones se efectúa explotando el tipo de objeto.

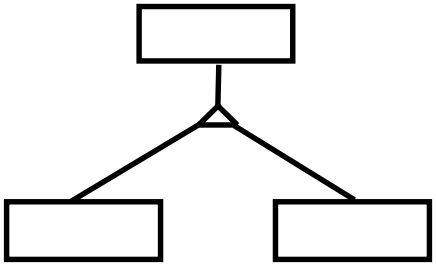
b) TIPO DE INTERRELACIÓN NIVELADA



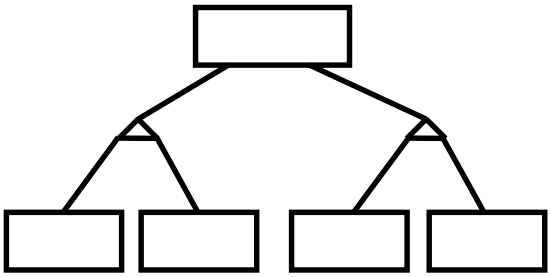


c) TIPO DE INTERRELACIÓN JERÁRQUICA

c.1) Generalización

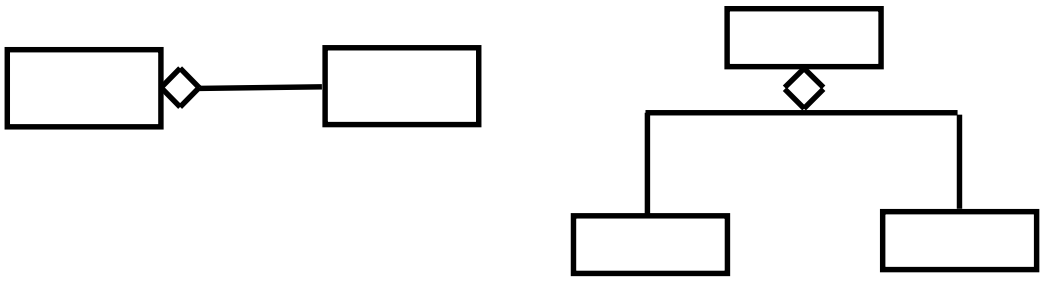


Múltiples generalizaciones

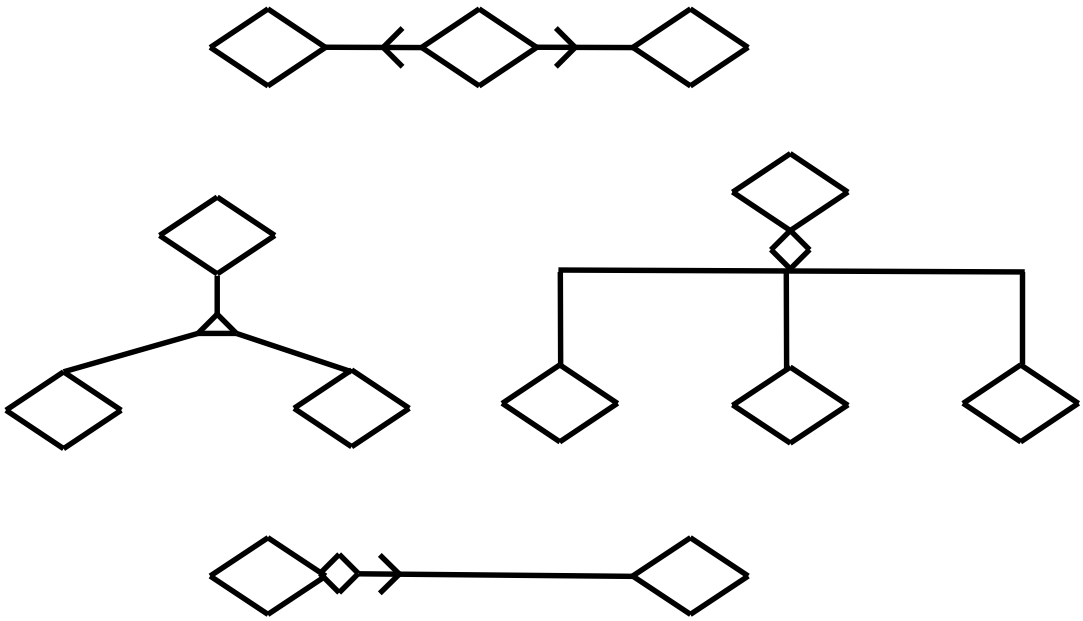


c.2) Meronimia miembro/colección

Meronimia compuesto/componente

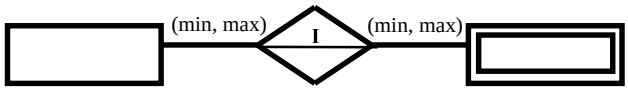


d) TIPOS DE INTERRELACIÓN ENTRE TIPOS DE INTERRELACIÓN



e) RESTRICCIONES APLICABLES A TI

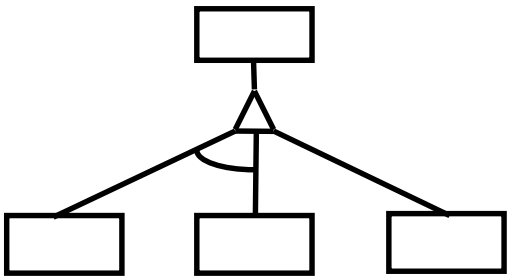
e.1) Dependencia en identificación



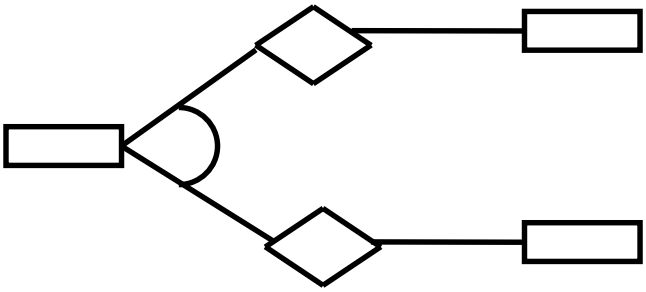
e.2) Dependencia en existencia

e.3) Exclusividad

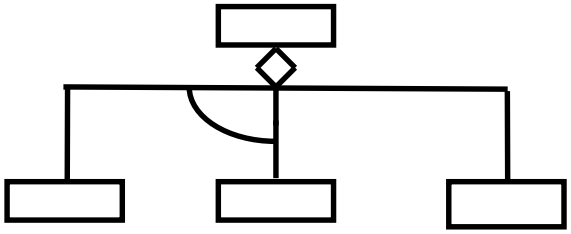
- **Exclusividad en generalizaciones**



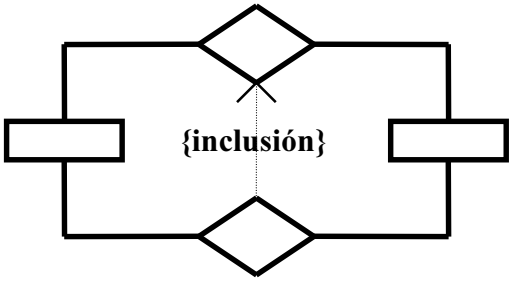
- **Exclusividad en interrelaciones niveladas**



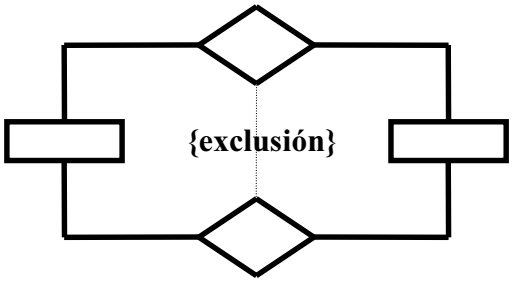
- **Exclusividad en meronimias compuesto/componente**



e.4) Restricción de inclusión

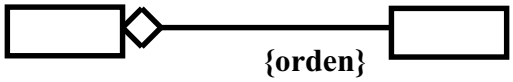


e.5) Restricción de exclusión

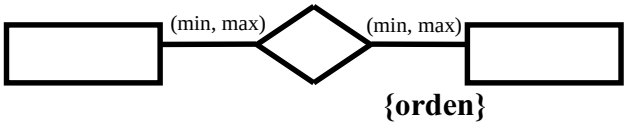


e.6) Restricción de orden

- **En meronimia miembro/colección**

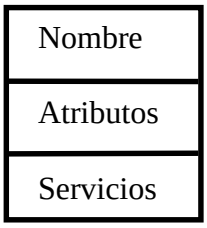


- **En interrelaciones niveladas binarias**



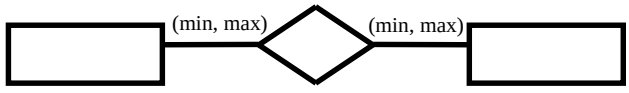
Apéndice III:
Notación gráfica de MIMO

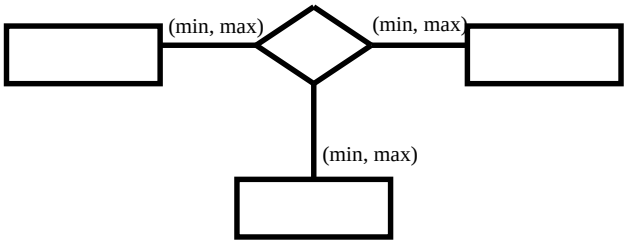
a) TIPO DE OBJETO



La especificación de los tipos de los atributos (TIPOS VALOR), de la cabecera de los servicios y de las restricciones se efectúa explotando el tipo de objeto.

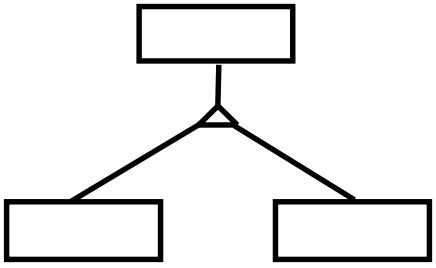
b) TIPO DE INTERRELACIÓN NIVELADA



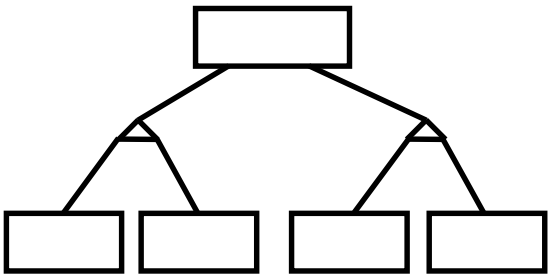


c) TIPO DE INTERRELACIÓN JERÁRQUICA

c.1) Generalización

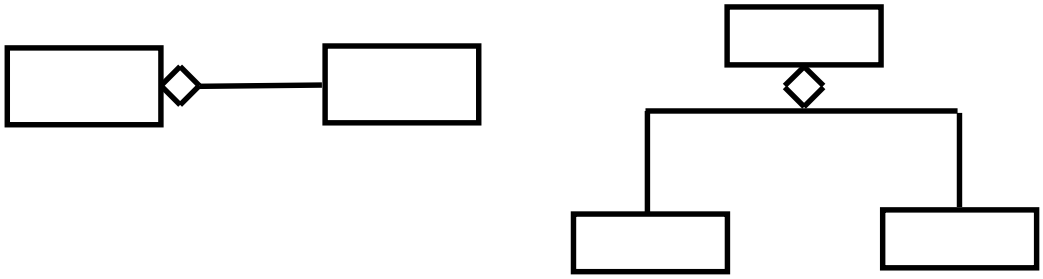


Múltiples generalizaciones

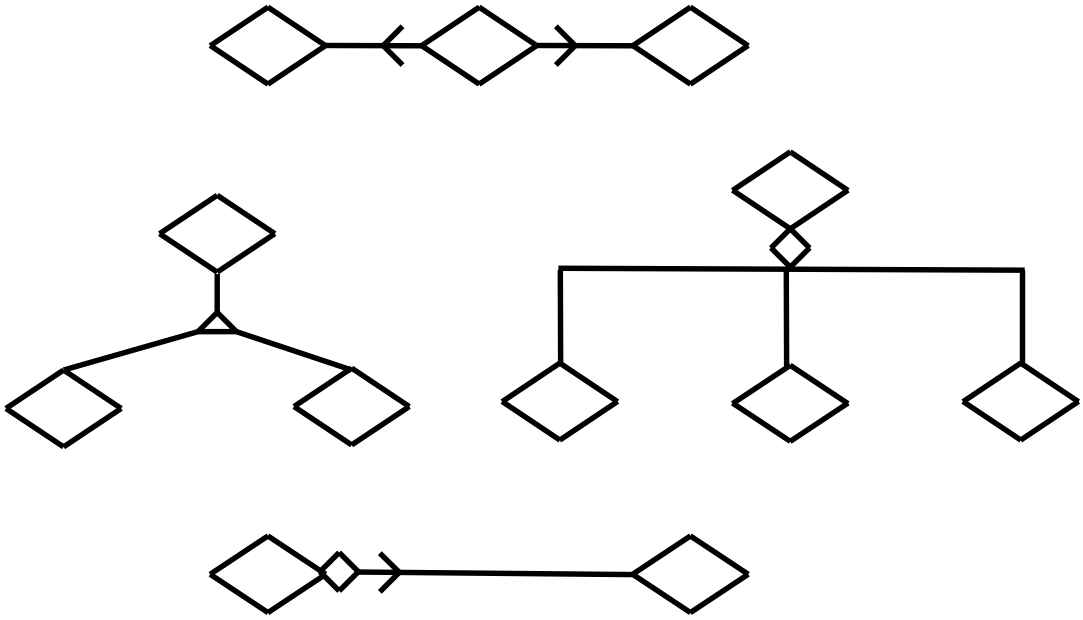


c.2) Meronimia miembro/colección

Meronimia compuesto/componente

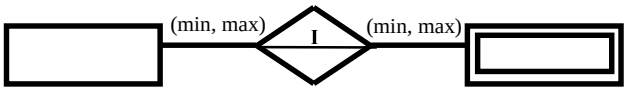


d) TIPOS DE INTERRELACIÓN ENTRE TIPOS DE INTERRELACIÓN



e) RESTRICCIONES APLICABLES A TI

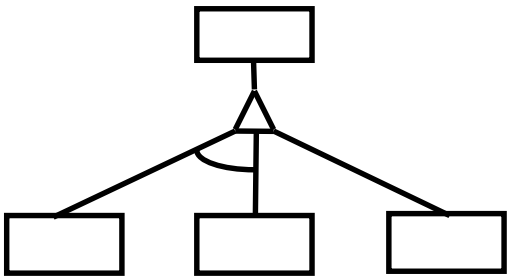
e.1) Dependencia en identificación



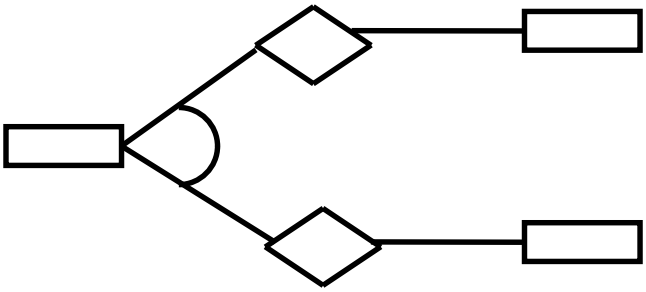
e.2) Dependencia en existencia

e.3) Exclusividad

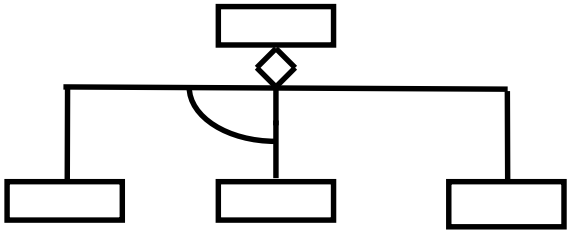
- **Exclusividad en generalizaciones**



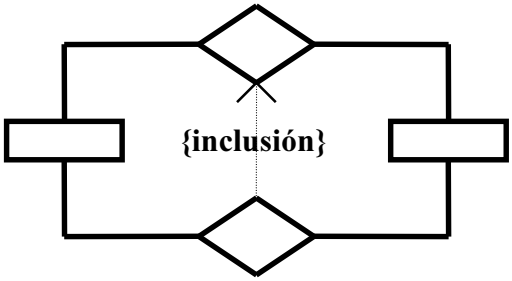
- **Exclusividad en interrelaciones niveladas**



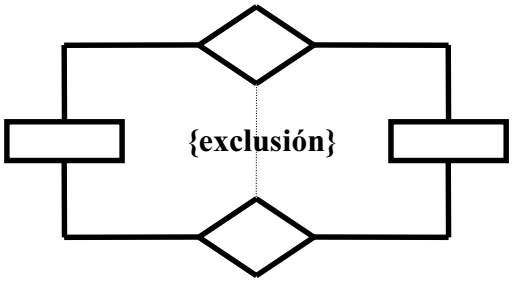
- **Exclusividad en meronimias compuesto/componente**



e.4) Restricción de inclusión

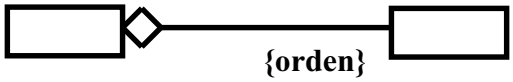


e.5) Restricción de exclusión

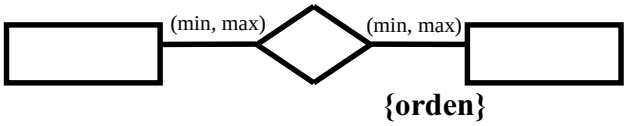


e.6) Restricción de orden

- **En meronimia miembro/colección**



- **En interrelaciones niveladas binarias**



Apéndice IV:
Diseño de MIMO

A continuación se presenta el diseño simplificado del esquema de MIMO. Tal y como se indicó en el capítulo 5, se presenta el diseño del nivel de análisis de MIMO, que corresponde a la zona 2 de la figura 5.3 (capítulo 5).

Sólo se detallan aquellos aspectos que son relevantes del modelo y no de la implementación. Así, por ejemplo, no aparecen los constructores y destructores necesarios ni otro tipo de operaciones o estructuras derivadas de necesidades de implementación.

Además, algunos tipos de datos y restricciones definidas se han modificado debido a limitaciones del producto. Por ejemplo, POET no soporta estructuras, por lo que todas las estructuras definidas en el diseño se han implementado en clases. POET tampoco soporta restricciones como la unicidad, no nulidad, etc., por lo que todas ellas se han implementado mediante operaciones que tampoco se reflejan en el diseño que presentamos.

Diseño simplificado del esquema conceptual de MIMO

CLASE: TIPO DE DATOS
{abstracta}
Atributos
Nombre: char(15) único, no_nulo
Servicios: conjunto (SERVICIO)
Tipo_abstracto: Booleano
Servicios
Generalizaciones: lista (GENERALIZACIÓN)
Restricciones

CLASE: ATRIBUTO
Atributos
Nombre: char(15) único, no_nulo
Tipo_dato: TIPO_VÁLIDO no_nulo
Nulidad: Booleano
Unicidad: Booleano
Rest_verif: conjunto (string)
Servicios
Aserciones (TD): conjunto (ASERCIONES)
/* devuelve las aserciones en la que participa el TD */
Restricciones

CLASE: SERVICIO
Atributos
Nombre: char(15) no_nulo /* no es único porque se soporta polimorfismo */
Parámetros: conjunto (PARÁMETRO)
Tipo_devuelto: TIPO_VÁLIDO
Diferido: Booleano
Precondiciones: conjunto (string)
Postcondiciones: conjunto (string)
Excepciones: conjunto (string)
Servicios
Restricciones

ESTRUCTURA: PARÁMETRO
Nombre: string único, no nulo
Tipo_datos: string no nulo

CLASE: ASERCIÓN

Atributos
Nombre: char(15) único, no_nulo
Tipo_dato: conjunto (TIPOS DE DATOS) no_nulo
Restricción: string
Servicios
Restricciones
Verificar (Tipo_dato <> ∅)

CLASE: TIPO ENUMERADO
{subtipo de: TIPO DE DATOS}
Atributos
Valores: lista (string)
Servicios
Restricciones
Verificar (numero_elementos (lista) > 0)

CLASE: TIPO DOMINIO
{subtipo de: TIPO DE DATOS}
{abstracta}
Atributos
Tipo_básico: TIPO_VALIDO no nulo
Valor_por_defecto: string
Servicios
Restricciones

CLASE: TIPO DOMINIO EXTENSIÓN
{subtipo de: TIPO DOMINIO}
Atributos
Valores: lista (string) no nulo
Servicios
Restricciones
Verificar (numero_elementos (lista) > 0)
Verificar (cada elemento de la lista es de TIPO_BÁSICO)
Verificar (valor_por_defecto pertenece a Valores)
CLASE: TIPO DOMINIO INTENSIÓN
{subtipo de: TIPO DOMINIO}
Atributos
Valores: string no nulo
Servicios
Restricciones
Verificar (la condición del string se define sobre el tipo básico)
Verificar (valor_por_defecto verifica la condición del string)

CLASE: TIPO COMPUESTO

{subtipo de :TIPO DE DATO}
{abstracta}

Atributos
Atributos: conjunto (ATRIBUTO), no_nulo
No Nulidad: conjunto (conjunto (ATRIBUTO))
Unicidad: conjunto (conjunto (ATRIBUTO))
Rest_verif: conjunto (string)

Servicios
Meronimias: lista (MERONIMIA)
/* devuelve los TI de meronimia en las que participa el TO */

Restricciones
Verificar (Atributos <> ∅)
Verificar que la clave se define sobre atributos de este tipo
Verificar que la no nulidad se define sobre atributos de este tipo
Verificar que la unicidad se define sobre atributos de este tipo

CLASE: TIPO OBJETO
{subtipo de :TIPO COMPUESTO}
{abstracta}

Atributos
OID: string no nulo, único
Clave: conjunto (ATRIBUTO)
Exclusividad: lista (lista(TI_NIVELADA))

Servicios
Interrelaciones_niveladas (TO): lista(TI_NIVELADA)
/* devuelve los TI niveladas en las que participa el TO */

Restricciones
Verificar que el TO participa en los TI niveladas definidas en el atributo exclusividad.

CLASE: TIPO VALOR COMPUESTO
{subtipo de :TIPO DE DATO COMPUESTO}
{abstracta}

Atributos
Servicios
Restricciones

ESTRUCTURA: ESTR_TIPO_DEPEN
TipoDependencia: enumerado {Existencia , Identificación}
Débil: TIPO_DATO
FIN_ESTRUCTURA

CLASE: TIPO DE INTERRELACIÓN NIVELADA
{subtipo de: TIPO OBJETO}

Atributos
Participantes: lista (OBJ_PART) no_nulo
Dependencia: (ESTR_TIPO_DEPEN)
Excluye: lista (TI_NIVELADA)
Incluido_en: lista (TI_NIVELADA)
Servicios
Grado (TI_NINELADA): Entero
/* devuelve el número de TO participantes en la interrelación nivelada */
Incluir una restricción de exclusión en el TI_NIVELADA de excluye.
/* este servicio se encarga de asegurar que la exclusividad es bidireccional */
Restricciones
Verificar (grado>=2)
Verificar que el TIPO_DATO del objeto débil del atributo dependencia participa en este TI nivelada.
Verificar que los TI_NIVELADA de excluye participan en esta TI nivelada.
Verificar que los TI_NIVELADA de incluido_en participan en esta TI nivelada.
Verificar que los TI_NIVELADA de incluido_en no están incluidos en este TI nivelada.

ESTRUCTURA: OBJ_PART
Objeto: TIPO_OBJETO
Papel: Char (15)
Cardinalidad: Estructura (min: Char (3), max: Char (3))
FIN_ESTRUCTURA

CLASE: GENERALIZACIÓN
Atributos
Nombre: Char(15)
Supertipos: TIPO_DATO
Subtipos: lista (SUBTIPO)
Exclusividad: lista (lista(TIPO_DATO))
Servicios
Restricciones
Verificar que cada TIPO_DATO del atributo exclusividad son subtipos de esta generalización.

ESTRUCTURA: SUBTIPO
Nombre: TIPO_DATO
Cardinalidad: Estructura (min Char(3), max char(3))
FIN_ESTRUCTURA

CLASE: MERONIMIA
{abstracta}
Atributos

Nombre: Char(15)
Holónimo: lista (TIPO_COMPUESTO)
Dependencia: lista(ESTR_TIPO_DEPEN)

Servicios

Restricciones

CLASE: MERONIMIA COMPUESTO COMPONENTE
{subtipo de: MERONIMIA}

Atributos
Merónimos: lista (SUBTIPOS)
Exclusividad: lista (lista(TIPO_COMPUESTO))

Servicios

Restricciones
Verificar (en la exclusividad sólo participan merónimos)

CLASE: MERONIMIA MIEMBRO COLECCIÓN
{subtipo de: MERONIMIA}

Atributos
Merónimos: SUBTIPO
Orden: lista (ATRIBUTO)

Servicios

Restricciones
Verificar que en la lista del atributo dependencia (heredado) hay como máximo un elemento.

Bibliografía

"Quién no añade nada a sus conocimientos, los disminuye"

Talmud

La bibliografía se ha dividido en dos secciones:

1. La primera contiene la **bibliografía tradicional**, entre la que se encuentran los libros, artículos y comunicaciones (citados, estudiados o simplemente consultados) así como los estándares (normas, borradores de normas, así como propuestas para la elaboración de éstos).

2. La segunda contiene una serie de direcciones interesantes de la **red de Internet**. Este tipo de búsqueda de documentación plantea una nueva problemática en cuanto al modo de referenciarla ya que, debido a su reciente aparición, aún no existe un criterio comúnmente aceptado. Algunos documentos que en estas direcciones aparecen, pueden ser documentos publicados en medios tradicionales (libros, revistas, etc.) pero otros no. Además, estás direcciones, consituyen una conexión de interés, no sólo por los documentos que en ellas puedan encontrarse, sino también porque constituyen un punto de búsqueda para cualquier persona interesada en el tema tratado.

Por ello hemos decidido incluir en la bibliografía la sección ***lugares de interés***. En ella aparecen direcciones que han sido de gran utilidad para la realización de este trabajo. Junto con cada dirección, se da una breve explicación de la información contenida en dicho lugar. Es importante resaltar que la dirección citada podría variar.

B.1. Bibliografía tradicional

Alcina (1994), "Aprender a Investigar. Métodos de trabajo para la redacción de tesis doctorales", J. Alcina Franch. Compañía Literaria, S. L. Madrid 1994.

Anderson et al. (1991), "Manageable Object-Oriented Development: Abstraction, Descomposition, and Modeling". J.A Anderson, J.D. Sheffler y E.S. Ward. TRI-Ada ´91 Proceedings. ACM, New York 1991, pp. 199-212.

Andonoff et al. (1995), "Modelling inheritance, composition and relationship links between objects, object versions and class versions". E. Andonoff, G. Hubert, A. Le Parc, G. Zurfluh. CAiSE'95, 7th International Conference on Advanced Information Systems Engineering. Eds. Juhani Iivari, Kalle Lyytinen y Matti Rossi. Serie: Lecture Notes in Computer Science 932, Jyväskylä, Finland, junio 1995.

ANSI (1986), "Reference Model for DBMS Standarization". Report of the DAFTG of the ANSI/X3/SPARC Database Study Group, en SIGMOD RECORD, Vol. 17, Nº 2, junio 1986.

ANSI (1990a), "Reference Model for Information Resource Dictionary Systems". Draft Technical, Report X3H4/90 -195R-, 1990.

ANSI (1990b), "Revised draft of Reference Model for Object Data Management". DBSSG/OODBTG OODB 89-01-R5, 1990.

Aracil (1986a), "Máquinas, sistemas y modelos. Un ensayo sebre sistemática". Javier Aracil. TECNOS, S. A. Madrid, 1986.

Aracil (1986b), "Introducción a la dinámica de sistemas". Javier Aracil. Alianza Editorial, S.A., Madrid 1986.

Armout et al. (1995), "Tailoring OO analysis and design methods". Frank Armour, Todd Cotton, Geoff Hambrick, Barbara Moo and Dennis Mancl. Panel presentado en OOPSLA'95. 10th. Conference on Object-Oriented Programing Systems, Languages ans Applications, Austin, Texas, 1995, pp. 185-186.

Arpinar et al. (1993), "MoodView: An Advanced Graphical User Interface for OODBMSs". I.B. Arpinar, A. Dogaç y C. Evrendilek. SIGMOD RECORD, Vol. 22, No. 4, diciembre 1993. pp. 11-18.

Atkinson et al. (1989), "The object-oriented database system manifesto". M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier y S. Zdonik. Proc. First International Conference on Deductive and Object Oriented Databases, Kyoto (Japan), 1989.

Balsters et al. (1993), "Typed Sets as a Basis for Object-Oriented Database Schemas". Herman Balsters, Rolf A. de By, Roberto Zicari. ECOOP'93, 7th European Conference on Object Oriented Programing. Ed. O. Nierstrasz. Serie: Lecture Notes in Computer Science 707, Kaiserslautern, Germany, julio 1993, pp.161-184.

Bancilhon y Ferran (1994), "ODMG-93: The Object Database Standard". François Bancillhon y Guy Ferran. Bulletin of Technical Committee on DATA ENGINEERING IEEE Computer Society. Vol. 17, Nº 4, diciembre 1994, pp.3-14.

Banchilhon y Kim (1990), "Object-Oriented Database Systems: In Transition". François Bancillhon y Won Kim.SIGMOD RECORD Vol. 19, No. 4, diciembre 1990, pp. 49-53.

Bandinelli et al. (1995), "Experiences in the Implementation of a Process-centered Software Engineering Environment Using Object-Oriented Technology". S. Bandinelli, L. Baresi, A. Fuggetta, L. Lavazza. Theory And Practice of Object Systems, Vol. 1(2), 1995, pp. 115-131.

Banker et al. (1993), "Repository Evaluation of Software Reuse". Rajiv D. Banker, Robert J. Kauffman y Dani Zweig. IEEE Transactions on Software Engineering, Vol. 19, No 4, abril 1993, pp.379-389.

Batori y Vasavada (1993), "Software Components for Object-Oriented Database Systems". Don Batori y Devang Vasavada. World Scientific Publishing Company, Vol. 3, No. 2 1993, pp. 165-192.

Beech (1993), "Collections of Objects in SQL3". David Beech. Proceedings of the 19th VLDB Conference, Dublin, Ireland, 1993, pp.244-255.

Beeri (1994), "Query Languages for Models with Object-Oriented Features". En: Advances in Object-Oriented Database Systems. Proceedings of the NATO Advanced Study Institute on Object-Oriented Database Systems, 1993. Eds.Asuman Dogac, M. Tamer Özsu, Alexandros Biliris y Timos Sellis. Springer-Verlag, 1994, pp.47-71.

Bermudez (1997), "Diseño e implementación de un modelo de objetos como prototipo de la metabase de una herramienta de diseño de bases de datos". Santiago Bermudez. Proyecto fin de carrera (dirigido por Esperanza Marcos). Escuela Politécnica Superior. Universidad Carlos III de Madrid. Leganés, 1997.

Berre y Oldevik (1995), "EXPRESS-OODL - Object-oriented extensions to EXPRESS as a basis for Information Interchange, Information sharing and System Interoperability". Arne Jorgen Berre y Jon Oldevik. ISO/TC184-10303, WG5 N250, 1995.

Bertino (1991), "Object-Oriented Database Management-Systems: Concepts and Issues". E. Bertino. IEEE Computer, abril 1991.

Bertino et al. (1993), "Specification of CHIMERA. The conceptual interface of IDEA". Elisa Bertino, Stefano Ceri y Rainer Manthey. Esprit Project 6333, junio 1993. Conferencia Internacional, Sistemas de Bases de Datos: en la frontera del 2000. CIEMAT, Ministerio de Industria y Energía, Madrid, 7-9 de junio, 1994.

Bertino et al. (1994), "Deductive Object Databases". Elisa Bertino, Giovanna Guerrini y Danilo Montesi. Proceedings of the 8th. European Conference on Object Oriented Programming (ECOOP'94) in Lecture Notes of Computer Science (LNCS). Ed. by Mario Tokoro y Remo Pareschi. Springer Verlag, Berlin 1994, pp. 213-235.

Blaster et al. (1993), “Typed Sets as a Basis for Object-Oriented Database Schemas”. Herman Balsters, Rolf A. de By y Roberto Zicari. ECOOP’93 Object.Oriented Programming. En proceedings de la 7th. European Conference., Kaiserlautern, Germany, julio de 1993. De. Oscar M. Nierstrazs. Serie: Lecture Notes in Computer Science (LNCS 707). Springer-Verlag, 1993, pp. 161-184.

Blum (1992), “Software Engineering. *A Holistic View*”. Bruce I. Blum. Oxford University Press, 1992.

Blum (1996), “Beyond Programming: To a New Era of Design”. Bruce I. Blum. Oxford University Press, 1996.

Bock y Odell (1996), “A User-Level Model of Composition”. Conrad Bock y James Odell. Report on Object Analysis & Design, SIGS Publications, Inc. NY, mayo-junio 1996, Vol.2, nº 7, pp.5-8.

Booch (1994), “Object-Oriented Analysis and Design”. The Benjamin/Cummings Publishing Company, Inc. 1994.

Booch (1996a), “Object-Oriented Development with Java”. Grady Booch. Report on Object Analysis & Design, SIGS Publications, Inc. NY, marzo-abril 1996, pp.2-4.

Booch (1996b), “Patterns and Protocols”. Grady Booch. Report on Object Analysis & Design, SIGS Publications, Inc. NY, mayo-junio 1996, Vol.2, nº 7, pp.2-4.

Booch y Rumbaugh (1995), "Unified Method for Object-Oriented Development". Grady Booch y James Rumbaugh. Documentation Set V0.8. Rational Software Corporation, 1995.

Booch et al. (1996a), “The Unified Modelling Language for Object-Oriented Development”. Grady Booch, James Rumbaugh y Ivar Jacobson. Documentation Set V0.9 Addemdum. Rational Software Corporation, julio 1996.

Booch et al. (1996b), “The Unified Modelling Language for Object-Oriented Development”. Grady Booch, James Rumbaugh y Ivar Jacobson. Documentation Set V0.91 Addemdum. Rational Software Corporation, septiembre 1996.

Booch et al. (1997), “Unified Modelling Language, V1.0”. Grady Booch, James Rumbaugh y Ivar Jacobson. Rational Software Corporation, enero 1997.

Bowen y Hinchey (1995), “Seven More Myths of formal Methods”. Jonatha P. Bowen y Michael G. Hinchey. IEEE Software, julio 1995, pp.34-41.

Brachman y Schmolze (1985), “An Overview of the KL-ONE Knowledge Representation System”. Ronald J. Brachman y James G. Schmolze. Cognitive Science 9, 1985, pp. 171-216.

Breuker y Van de Velde (1994), “The CommonKADS Library for Expertise Modelling”. J.A. Breuker y W. Van de Velde. IOS Press, Amsterdam, The Netherlands, 1994.

Brodie et al. (1984), “On Conceptual Modelling. Perpectives from Artificial Intelligence, Databases and Programming Languages”. Michael L. Brodie, John Mylopoulos, Joachim W. Schmidt. Springer-Verlag, 1984.

Bruce (1992), “Designing Quality Databases with IDF1X Information

Models”, Thomas A. Bruce. Dorset House Publishing, NY 1992.

Bryant (1991), “A Comparison of Object-Oriented and Relational Databases”. David H. Bryant. Software Concepts & Trends. Datapro Reports on Software. McGraw-Hill, Inc. USA, diciembre 1991, pp.101-108.

Bunge (1976), “La Investigación Científica”. Mario Bunge. Ariel, S.A. Barcelona, 1976.

Bunge (1985), “Epistemología”. Mario Bunge. Ariel S.A. Barcelona, 1985.

Byung (1995), “OODB design with EER”. SIGS Publications, Inc., N. Y. USA, 1995

Calvanese y Lenzerini (1994), “Making Object-Oriented Schemas More Expressive”. Diego Calvanese y Maurizio Lenzerini. Proceedings of the Thirteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (SIGMOD/PODS 94), Minneapolis, Minnesota USA, mayo 24-26, 1994, pp. 243-254.

Candel (1982), “Categorías”. En tratados de lógica (Organon). (Traduccón de *Categorías* de Aristóteles). Ed. Gredos. Madrid, Vol. I. 1982.

Cattell (1994a), "The Object Database Standard: ODMG-93", release 1.1. Edited by R.G.G. Cattell. Morgan Kaufmann Publishers, San Francisco 1994.

Cattell (1994b), "Object Data Management. Object-Oriented and Extended Relational Database Systems". Revised Edition. Addison-Wesley Publishing Company, 1994

Cattell (1995), "The Object Database Standard: ODMG-93", release 1.2. Edited by R.G.G. Cattell. Morgan Kaufmann Publishers, San Francisco 1995.

Cattell et al. (1995), “Object and Database Standards”. Rick Cattell, Frank Manola, Richard Soley, Jeff Sutherland y Mary Loomis. Panel presentado en OOPSLA’95. 10th. Conference on Object-Oriented Programing Systems, Languages ans Applications, Austin, Texas, 1995, pp. 331-332.

Ceri (1993), “Consolidated Specification for Chimera”. S. Ceri. Technical Report, IDEA.DE.2P.006.01, ESPRIT Proyect 6333, noviembre 1993.

Chalmers (1984), “¿Qué es esa cosa llamada ciencia?”. Alan Chalmers. Siglo XXI de España Editores, S.A. Madrid, 2ª edición, 1984.

Chalmers (1992), “La ciencia y como se elabora”. Alan Chalmers. Siglo XXI de España Editores, S.A. Madrid, 1992.

Champeaux y Faure (1992), “A comparative study of object-oriented analysis methods”. D. Champeaux y P. Faure. Journal on Object-Oriented Programing, marzo-abril 1992, pp.21-33.

Chen (1976), “The Entity/Relationship Model: Toward a Unified View of Data”. P. Chen. ACM Transactions on Database Systems, Vol. 1, No. 1, marzo 1976.

Chen (1983), “English Sentence Structure and Entity-Relationship Diagrams”. P. Chen. Information Sciences 29, 1983, pp.127-147.

Chen (1990), “Entity-Relationship Approach to Data Modeling”. P. Chen. IEEE, 1990, pp. 238-243.

Chonoles et al. (1996), “Speaking of Methods”. Michael J. Chonoles, James A. Schardt y Phil J. Magrogan. Report on Object Analysis & Design, SIGS Publications, Inc. NY, marzo-abril 1996, pp. 8-12.

Chou y Chen (1996), “An Object-Oriented Analysis Method Based on Parallel Descomposition of Function and Data”. Shih-ChienChou y Jen-Yen Chen. Report on Object Analysis & Design, SIGS Publications, Inc. NY, marzo-abril 1996, pp. 22-35.

Clancey (1985), “Heuristic Classification”. William J. Clancey. Artificial Intelligence, Vol.27, 1985, pp. 289-350.

Coad y Yourdon (1991a), “Object-Oriented Design”. Peter Coad y Edward Yourdon. Prentice-hall, 1991.

Coad y Yourdon (1991b), “Object-Oriented Analysis, 2nd Ed.”. Peter Coad y Edward Yourdon. Prentice-hall, 1991.

Coleman (1993), “FUSION: An Object-Oriented Analysis and Design Method”. Proc. Object Expo Europe, Londrés, julio 1993.

Coleman et al. (1994), “Object-Oriented Development: The Fusion Tethod”. P. Coad, D. North y M. Mayfield, Prentice Hall, 1994.

Costa et al. (1996), “Las clases de objetos en ROSES”. P. Costa, M. Barceló, A. Olivé, C. Quer, A. Roselló, M. R. Sancho. En Actas de las I Jornadas de Investigación y Docencia en Bases de Datos. Ed. N. Brisaboa et al. A Coruña, julio 1996, pp.98-108.

Costilla et al. (1995), "The Object Data Model and its Graphical User Interface for an Object Database System". C.R. Costilla, M.J. Bass y J.L. Villamor. Basque International Workshop on Information Technology, San Sebastián julio 1995. Proceedings, IEEE Computer Society Press, pp.161-172.

Cook y Daniels (1994), “Designing Object Systems-Object Oriented Modelling with Syntropy”. S. Cook y J. Daniels. Prentice-Hall 1994.

Darwen y Date (1995), “The Third Manifesto”. Hugh Darwen y C.J. Date. SIGMOD Record, Vol. 24, No.1, marzo 1995, pp.39-49.

DBL: ARL-078 (1991), "Major Object-Oriented SQL Extension". Jim Melton. ISO/IEC JTC1/SC21/WG3 DBL: ARL-078, 1991.

DBL: CBR-003 (1992), "(ISO Working Draft)". Jim Melton. ISO/IEC JTC1/SC21 WG3 DBL: CBR-004, 1992

DBL: YOW-003 (1995), "(ISO Working Draft) Part 1: Framework for SQL (SQL/Framework)". Jim Melton. ISO/IEC JTC1/SC21 WG3 DBL: YOW-003, marzo 1995.

DBL: YOW-004 (1995), "(ISO Working Draft) Part 2: Foundation (SQL/Foundation)". Jim Melton. ISO/IEC JTC1/SC21 WG3 DBL: YOW-004, marzo 1995.

DBL: YOW-031 (1995), "Accomodating SQL3 and ODMG". Jim Melton. ISO/IEC JTC1/SC21 WG3 DBL: YOW-032, mayo 1995.

DBL: LHR-003 (1995), "(ISO Working Draft) Part 1: Framework for SQL (SQL/Framework)". Jim Melton. ISO/IEC JTC1/SC21 WG3 DBL: LHR-003, 1995.

DBL: LHR-004 (1995), "(ISO Working Draft) Part 2: Foundation (SQL/Foundation)". Jim Melton. ISO/IEC JTC1/SC21 WG3 DBL: LHR-004, 1995.

DBL: LHR-029 (1995), “UK position on “object” support in SQL3”. UK Experts. ISO/IEC JTC1/SC21 WG3 DBL: LHR-029, Augost 1995.

DBL: LHR-030 (1995), “Response to UK-Position OO”. DIN-NI21.3. ISO/IEC JTC1/SC21 WG3 DBL: LHR-030, octubre 1995.

DBL: LHR-031 (1995), “Reprioritising Object ADTs”. UK DBL. ISO/IEC JTC1/SC21 WG3 DBL: LHR-031, noviembre 1995.

DBL: LHR-032 (1995), “The SQL Object Model”. UK Experts. ISO/IEC JTC1/SC21 WG3 DBL: LHR-032, noviembre 1995.

DBL: LHR-034 (1995), “Unnesting nested tables”. Hugh Darwen (UK). ISO/IEC JTC1/SC21 WG3 DBL: LHR-034, noviembre 1995.

DBL: LHR-036 (1995), “Object extents refined”. J. Melton, A. Eisenberg, P. Shaw y R. Monajjemi. ISO/IEC JTC1/SC21 WG3 DBL: LHR-036, noviembre 1995.

DBL: LHR-037 (1995), “Making extent table use optional”. J. Melton. ISO/IEC JTC1/SC21 WG3 DBL: LHR-037, noviembre 1995.

DBL: LHR-057 (1995), “SQL/CORBA”. Kazuhiro Oiso. ISO/IEC JTC1/SC21 WG3 DBL: LHR-057, noviembre 1995.

DBL: LHR-061 (1995), “UK response to lhr-36 (and lhr-37)”. UK Experts. ISO/IEC JTC1/SC21 WG3 DBL: LHR-061, diciembre 1995.

DBL: LHR-077 (1995), “Introducing Reference Types and Cleaning up SQL3’s Object Model”. K. Kulkarni, M. Carey, L. DeMichiel, N. Mattos, W. Hong, M. Ubell, A. Nori, V. Krishnamurthy, D. Beech. ISO/IEC JTC1/SC21 WG3 DBL: LHR-077, diciembre 1995.

DBL: LHR-079 (1995), "The ODMG Object Query Languaje (OQL 1.2)". Franç̧ois Bancillhon ISO/IEC JTC1/SC21 WG3 DBL: LHR-079, diciembre 1995.

DBL: MCI-001 (1996), "Minutes of the Database Languages Rapporteur Group Meeting". Stephen Cannan. ISO/IEC JTC1/SC21 WG3 DBL: MCI-001, 1996.

DBL: MCI-003 (1996), "(ISO Working Draft) Part 1: Framework for SQL (SQL/Framework)". Jim Melton. ISO/IEC JTC1/SC21 WG3 DBL: MCI-003, marzo 1996.

DBL: MCI-004 (1996), "(ISO Working Draft) Part 2: Foundation (SQL/Foundation)". Jim Melton. ISO/IEC JTC1/SC21 WG3 DBL: MCI-004, marzo 1996.

DBL: MCI-010 (1996), "(ISO Working Draft) Part 8: Extended Objects (SQL/Object)". Jim Melton. ISO/IEC JTC1/SC21 WG3 DBL: MCI-010, marzo 1996.

DBL: MCI-043r1 (1996), “LHR-032 Follow-on”. UK Experts. ISO/IEC JTC1/SC21 WG3 DBL: MCI-043r1, abril 1996.

DBL: MCI-075 (1996), “Further Restriction on the use of REF types”. K. Kulkarni, M. Carey, L. DiMichiel, N. Mattos, A. Nori, V. Krishnamurthy, A. Calrson. ISO/IEC JTC1/SC21 WG3 DBL: MCI-075, febrero 1996.

DBL: MCI-076 (1996), “Field Names and Column Names”. K. Kulkarni, M. Carey, L. DiMichiel, N. Mattos, A. Nori, V. Krishnamurthy, A. Calrson. ISO/IEC JTC1/SC21 WG3 DBL: MCI-076, febrero 1996.

DBL: MCI-078 (1996), “Use of REF types in UPDATEs”. M. Carey, A. Calrson, L. DiMichiel, B. Kelley, K. Kulkarni, N. Mattos, A. Nori, V. Krishnamurthy. ISO/IEC JTC1/SC21 WG3 DBL: MCI-078, febrero 1996.

DBL: MCI-079 (1996), "Convergence with ODMG collection queries". SQL3/ODMG merger group. ISO/IEC JTC1/SC21 WG3 DBL: MCI-079, marzo 1996.

DBL: MCI-100 (1996), "Extended Object Support for SQL- Proposal for new SQL subproject". ISO/IEC JTC1/SC21 WG3 Database Languages Rapporteur Group. ISO/IEC JTC1/SC21 WG3 DBL: MCI-079, marzo 1996.

DBL: MAD-001 (1996), "Minutes of the Database Languages Rapporteur Group Meeting". Paul Cotton. ISO/IEC JTC1/SC21 WG3 DBL: MAD-001, mayo 1996.

DBL: MAD-003 (1995), "(ISO Working Draft) Part 1: Framework for SQL (SQL/Framework)". Jim Melton. ISO/IEC JTC1/SC21 WG3 DBL: MAD-003, 1996.

DBL: MAD-004 (1996), "(ISO Working Draft) Part 2: Foundation (SQL/Foundation)". Jim Melton. ISO/IEC JTC1/SC21 WG3 DBL: MAD-004, 1996.

DBL: MAD-010 (1996), "(ISO Working Draft) Part 8: Extended Objects (SQL/Object)". Jim Melton. ISO/IEC JTC1/SC21 WG3 DBL: MCI-010, 1996.

DBL: MAD-033 (1996), "Unsing named row types in <table type> definitions". György Kovács. ISO/IEC JTC1/SC21 WG3 DBL: MCI-033, abril 1996.

DBL: MAD-144 (1996), "Example od REF type requirement". Paul Scarponcini. ISO/IEC JTC1/SC21 WG3 DBL: MCI-144, diciembre 1996.

DBL: MAD-174 (1996), "Proposed Language Opportunity for virtual types". Fred Zemke. ISO/IEC JTC1/SC21 WG3 DBL: MCI-174, 1996.

DBL: MAD-175 (1996), "Minimal Array support for SQL3". Nelson Mattos. ISO/IEC JTC1/SC21 WG3 DBL: MCI-175, diciembre 1996.

DBL: MAD-187 (1996), "Response to SQL3 CD Ballot Comments". Krishna Kulkarni, Mike Carey. ISO/IEC JTC1/SC21 WG3 DBL: MCI-187, noviembre 1996.

DBL: MAD-217 (1996), "SQL/Object: A restement of the model". J. M. Sykes. ISO/IEC JTC1/SC21 WG3 DBL: MCI-217, diciembre 1996.

DBL: MAD-218 (1996), "Extending scope, in SQL/Object". J. M. Sykes. ISO/IEC JTC1/SC21 WG3 DBL: MCI-218, diciembre 1996.

DBL: MAD-223 (1997), "Resolution of Comment FRANCE-007: Fixing the Site Identity Model". Mihnea Andrei. ISO/IEC JTC1/SC21 WG3 DBL: MCI-223, enero 1997.

DBL: MAD-226 (1996), "Removal of <ADT Routine>". Hugh Darwen. ISO/IEC JTC1/SC21 WG3 DBL: MCI-226, diciembre 1996.

DBL: MAD-230 (1997), "Deferring Multiple Inheritance of ADTs". Hugh Darwen. ISO/IEC JTC1/SC21 WG3 DBL: MCI-230, enero 1997.

DBL: MAD-231 (1997), "Deprecation of domains". Hugh Darwen. ISO/IEC JTC1/SC21 WG3 DBL: MCI-231, enero 1997.

DBL: MAD-234 (1997), "Moving Sites". Mihnea Andrei. ISO/IEC JTC1/SC21 WG3 DBL: MCI-234, enero 1996.

DBL: MAD-235 (1997), "The ADT Implementation and Representation Concepts". Mihnea Andrei. ISO/IEC JTC1/SC21 WG3 DBL: MCI-235, enero 1997.

De Marco (1982), “Controlling Software Projects”. T. De Marco. Yourdon Press, New York 1982.

De Miguel y Piattini (1992), “SQL Lenguaje de Interrogación de Bases de Datos”, Adoración de Miguel y Mario Piattini. Cuadernos técnicos de INICIATIVAS. Normas y estándares internacionales. ICM, revista iniciativas, Madrid 1992.

De Miguel y Piattini (1993), "Concepción y diseño de bases de datos relacionales. Del modelo E/R al modelo relacional". A. de Miguel y M. Piattini. Rama, España 1992.

De Miguel y Piattini (1995), "Types in object bases development". A. de Miguel y M. Piattini. OOPS Messenger, Vol. 6, No. 2, abril 1995, pp. 12-17.

De Miguel et al. (1996a), "ENEAS/BD: Un ENtorno para la Enseñanza Avanzada de Sistemas de Bases de Datos". A. de Miguel, M. Piattini, P. Martínez y E. Marcos. En actas de las I Jornadas de Investigación y Docencia en Bases de Datos. Ed. N. Brisaboa et al. A Coruña, julio 1996, pp. 42-54.

De Miguel et al (1996b), “Object Oriented Modelling from Natural Language”. A. de Miguel, P. Martínez, E. Marcos y M. Piattini. En actas de INFONOR’96, Antofagasta, Chile, noviembre 1996.

Delobel (1994) “Convergence and/or divergence between SQL3 standard evolution and ODMG-93 standard for object-oriented databases systems”. C. Delobel. En actas de BIWIT´94. First Basque International Workshop on Information Technology, 1995, pp. 1-18.

Dittrich (1994), “Object-Oriented Database Systems: The Notion and the Isuues”. Klaus R. Dittrich. En: On Object-Oriented Database Systems. Proceedings of the NATO Advanced Study Institute on Object-Oriented Database Systems, Kusadasi, Turkey, agosto 1993. Eds. K.R.Dittrich, U.Dayal y A.P.Buchmann. Springer-Verlag, 1994, pp-3-10.

Dittrich (1994), "Object-Oriented Data Model Concepts". En: Advances in Object-Oriented Database Systems. Proceedings of the NATO Advanced Study Institute on Object-Oriented Database Systems, 1993. Eds.Asuman Dogac, M. Tamer Özsu, Alexandros Biliris y Timos Sellis. Springer-Verlag, 1994, pp.30-45.

Dori (1996), "Expressing Structural Relations Among Dimension-Set Componentes Using the Object-Process Methodology". Dov Dori. Report on Object Analysis & Design, SIGS Publications, Inc. NY, mayo-junio 1996, Vol.2, nº 7, pp.20-24.

D'Souza (1996), "Interfaces, subtypes, and framewoks". Desmond D'Souza. Journal of Object-Oriented Programing. SIGS Publications, noviembre-diciembre 1996, Vol.9, nº 7, pp.19-22.

Eick y Raupp (1991), "Towards a formal semantics and inference rules for conceptual data models". Christopt F. Eick y Thomas Raupp. Data & Knowledge Engineering 6, 1991, pp. 297-317.

Ellis y Stroustrup (1990), "The C++ Annotated Reference Manual". Margaret Ellis y Bejarne Stroustrup. Addisson-Wesley Publishing Company, Massachusets 1990.

Estany (1993), "Introducción a la Filosofía de la Ciencia". Anna Estany. CRÍTICA, Barcelona 1993.

Fetzer (1993), "Philosophy of Science", James H. Fetzer. Paragon House. Uneted States, 1993.

Firesmith (1996), "The Inheritance of State Models". Donald G. Firesmith. Report on Object Analysis & Design, SIGS Publications, Inc. NY, marzo-abril 1996, pp. 13-15.

Fisher y Mitchell (1995), "The Development of Type Systems for Object-Oriented Languages". K. Fisher y Johm C. Mitchell. Theory And Practice of Object Systems, Vol. 1(3), 1995, pp. 189-220.

Fong (1995), "Mapping Extended Entity Relationship Model to Object Modeling Technique". Joseph Fong. SIGMOD Record, Vol. 24, No. 3, septiembre 1995, pp.18-22.

Gabbay et al. (1993), "Handbook of Logic in Artificial Intelligence and Logic Programming". Ed. by Dov M. Gabbay, C.J. Hogger, J.A. Robinson. Oxford University Press Inc. New York, 1993

Gallagher (1994), "Influencing Database Language Standards". Leonard Gallagher. SIGMOD Record, Vol. 23, No. 1, marzo 1994, pp. 122-127.

Gallego (1987), "Ser Doctor. Cómo redactar una Tesis Doctoral", A. Gallego. Fundación Universidad-Empresa. Monografías Profesionales:107. Madrid, 1987.

García Molina (1996), "Dimensiones en el diseño de un modelo de vistas orientadas a objetos". J. García Molina, G. Guerrini, B. Catania. En actas de las I Jornadas de Investigación y Docencia en Bases de Datos. Ed. N. Brisaboa et al. A Coruña, julio 1996, pp. 119-129.

García Pelayo (1975), “La teoría General de Sistemas”, M. García Pelayo. Revista de Occiodente, 3ª época. 2. Madrid 1975.

Gardarin et al. (1989), “Managing Complex Objects in an Extensible Relational DBMS”. G. Gardarin, J.P. Cheiney, G. Kiernan, D. Pastre, H. Stora. 15th. Very Large Data Bases International Conference, Amsterdam, Holanda, agosto 1989, pp.55-65.

Gardarin y Valduriez (1992), “ESQL2: An Object-Oriented SQL with F-Logic Semantics”. G. Gardarin y P. Valduriez. 8th. International Conference on Data Engineering, Arizona, California 1992, pp. 320-327.

Gargouri et al. (1995a), “Towards a Generic Model for OO information system Modeling”. F. Gargouri, C.F. Ducateau, F. Boufarès. International Conference on Industrial Engineering and Production Management. Marruecos, abril 1995.

Gargouri et al. (1995b), “Relational Implementation of Object Oriented Information System Design Using a Generic Model”. F. Gargouri, F. Boufarès, C.F. Ducateau, 1995. pp. 114-129.

Gibbs et al. (1990), “Class Management for Software Communities”. S. Gibbs, D. Tsichritzis, E. Cosais, O. Nierstrasz y X. Pintado. Communications of the ACM, Vol. 33, No. 9, septiembre 1990, pp. 90-103.

Gossain (1996), “Object-Oriented Methods: Do they Matter?”. Sanjiv Gossain. Report on Object Analysis & Design, SIGS Publications, Inc. NY, mayo-junio 1996, Vol.2, nº 7, pp.9-11.

Graham (1994), “Object-Oriented Methods, 2nd De.”. I. Graham. Addison-Wesley 1994.

Grene (1985), “About the Division of Sciences” M. Grenge. En “Aristotle on Nature and Living Things”, Ed. Gotthelf. Mathesis Publications y Bristol Classical Press. Pittsburgh, 1985.

Guerraoui (1995), “Modular Atomic Objects”. Rachid Guerraoui. Theory And Practice of Object Systems, Vol. 1(2), 1995, pp. 89-99.

Gurevich y Huggins (1993), "The Semantics of the C Programming Language". Yuri Gurevich and James K. Huggins. CSL'92 (Computer Science Logic), Springer Lecture Notes in Computer Science 702, 1993, pp. 274-308.

Helga (1993), “An Abstraction-Based Rule Approach to Large-Scale Information Systems Development”. Anne Helga Seltveit. CAiSE’93, 5th International Conference on Advanced Information Systems Engineering. Eds. Colette Rolland, François Bodart y Corine Cauvet. Serie: Lecture Notes in Computer Science 685, París, Francia, junio 1993, pp.328-351.

Henderson-Sellers (1992), “A Book of Object-Oriented Knowledge”. Prentice Hall, 1992.

Henderson-Sellers (1994), “COMMA: An architecture for method interoperability”. Brian Henderson-Sellers. Report on Object Analysis & Design, SIGS Publications, Inc. NY, Vol. 1, Nº 3, 1994, pp.25-28.

Henderson-Sellers et al. (1995), "Merging OOAD methodologies: The COMA project and its first results". B.Henderson-Sellers, I.M.Graham, J.J.Odell, B.Unhelkar. Tutorial 51, presentado en OOPSLA'95. 10th. Conference on Object-Oriented Programing Systems, Languages ans Applications, Austin, Texas, 1995.

Henderson-Sellers y Edwars (1995), "Boock Two of Object-Oriented Knowledge: The Workind Object". Prentice Hall 1995.

Henderson-Sellers (1996a), "Convergence Is in the Air".Brian Henderson-Sellers. Report on Object Analysis & Design, SIGS Publications, Inc. NY, marzo-abril 1996, pp.47-49.

Henderson-Sellers (1996b), "The OPEN-MeNtOR Methodology". Brian Henderson-Sellers. Object Magazin. SIGS Publications, noviembre 1996, pp.56-59.

Henderson-Sellers y Bulthuis (1996a), "The COMMA Proyect: First Step". Brian Henderson-Sellers y A. Bulthuis. Report on Object Analysis & Design, SIGS Publications, Inc. NY, mayo-junio 1996, Vol.2, nº 7, pp.49-52.

Henderson-Sellers y Bulthuis (1996b), "COMMA: Sample metamodels". Brian Henderson-Sellers y A. Bulthuis. Journal of Object-Oriented Programing. Sección ROAD (Report on Object Analysis & Design). SIGS Publications, noviembre-diciembre 1996, Vol.9, nº 7, pp.44-48.

Henderson-Sellers y Dué (1997), "OPEN Project Management". Brian Henderson-Sellers y Richard Dué. Object Expert. SIGS Publications, enero-febrero 1997, pp.30-35.

Henderson-Sellers y Firesmith (1997), "COMMA: Proposed core model". Brian Henderson-Sellers y Donald G. Firesmith. Journal of Object-Oriented Programing. Sección ROAD (Report on Object Analysis & Design). SIGS Publications, enero 1997, Vol.9, nº 8, pp.48-53.

Hidalgo (1991), "Extensiones semánticas en el diseño de bases de datos: aplicación a un caso concreto". Trabajo fin de carrera, Facultad de Informática, U.P.M, 1991.

Hürsch (1994), "Should Superclasses be Abstract?". Walter L. Hürsch. Proceedings of the 8th. European Conference on Object Oriented Programming (ECOOP'94) in Lecture Notes of Computer Science (LNCS). Ed. by Mario Tokoro y Remo Pareschi. Springer Verlag, Berlin 1994, pp. 12-31.

IDEF1X (1993), "Integration Definition For Information Modelling (IDF1X)". Federal Information Processing Standards Publications 184. National Institute of Standars and Technology, diciembre 1993.

INTERLIS, "ISO/TC211 conceptual schema language evaluation, INTERLIS candidature-imput documents". Swiss Federal Institute of Technology, documento sin fecha.

Jaakkola (1992), "Data Models from Theory to Practice: Modelling the Adoption of New Data Models". Hannu Jaakkola. Information Modelling and Knowledge Bases III. S. Ohsuga et al., Eds. IOS Press, 1992, pp.675-690.

Jacobson et al. (1992), “Object-Oriented Software Engineering. A Use Case Driven Aproach”. I. Jacobson et al. ACM Press, Addison-Wesley, Wokingham, Inglaterra, 1992

Jacobson (1993), “Object-Oriented Software Engineering. A Use Case Driven Aproach, Revised”. I. Jacobson. ACM Press/Addison-Wesley 1993

Jensen et. al (1992), “A Glosary of Temporal Databases”. En SIGMOD RECORD, junio 1992.

Jesse et al. (1996), “Speaking of Methods”. M. Jesse Chonoles, J. A. Schardt, P.J. Magrogan. Report on Object and Analysis Design; SIGS Publications; marzo-abril, 1996.

Jordan y Van De Vanter (1995), “Software Configuration Management in an Object Oriented Database”. Mick Jordan y Michael L. Van De Vanter. Proceedings of the USENIX Conference on Object-Oriented Technologies (COOTS), USENIX Association, junio 26-29, Monterey, California, USA, 1995, pp. 55-67.

KADS-II (1993), Esprit Project P5248. Disponible en:
http://swi.psy.uva.nl/projects/CommonKADS/description/root.htm (subsecciones 3.1 y 3.2), 1993.

Karam y Casselman (1993), “A Cataloging Framework for Software Development Methods”. Gerald M. Karam y Ronald S. Casselman. IEEE Computer, febrero 1993, pp.34-45.

Kathuria (1997), “Improved modeling and design usign assimilation and property modeling”. Ravi Kathuria. Journal of Object-Oriented Programing. SIGS Publications, enero 1997, Vol.9, nº 8, pp.15-24.

Keene (1993), “Bridging the Gap Between Relational Data And Object Oriented Development”. C. Keene. Object Expo Europe Conference Procedings, SIGS Publications, NY 1993, pp. 171-174.

Kim et al. (1989), “Composite Object Revisted”. Won Kim. Proc. of the SIGMOD Int. Conf. on the Management of Data, 1989, pp. 337-347.

Kim (1990), “Object-Oriented Databases: Definition and Research Directions”. Won Kim. IEEE Transactions on Knowledge and Data Engineering, Vol. 2, No 3, septiembre 1990, pp. 327-341.

Kim (1994), "Observations on the ODMG-93 Proposal for an Object-Oriented Database Language". Won Kim. SIGMOD RECORD, Vol.23, No.1, marzo 1994, pp.4-9.

Korson y McGregor (1990), “Understanding Object-Oriented: A Unifying Paradigm”. Tim Korson y John D. McGregor. Communications of the ACM, Vol. 33, No. 9, septiembre 1990, pp 40-60.

Kosso (1992), “Reading the Book of Nature. An Introduction to the Philosophy of Science”. Peter Kosso. Cambridge University Press, 1992.

Kristen (1996), “Total Quality Management with Objec Orientation”. Gerald Kristen. Report on Object Analysis & Design, SIGS Publications, Inc. NY, marzo-abril 1996, pp. 40-46.

Kristensen y Østerbye (1996), “Roles: Conceptual Abstraction Theory and Practical Language Issues”. Bent Bruun y Kasper østerbye. Theory And Practice of Object Systems, Vol. 2(3), 1996, pp. 143-160.

Kulkarni (1993), “Object-orientation and the SQL standard”. Krishna G. Kulkarni. Computer Standards & Interfaces 15, 1993, pp.287-300.

Learmont y Cattell (1994), “An Object-Oriented Interface to a Relational Database”. T. Learmont y R.G.G.Cattell. En: On Object-Oriented Database Systems. Proceedings of the NATO Advanced Study Institute on Object-Oriented Database Systems, Kusadasi, Turkey, agosto 1993. Eds. K.R.Dittrich, U.Dayal y A.P.Buchmann. Springer-Verlag, 1994, pp.157-167.

Lee (1995), “Normalization in OODB Design”. Byung S. Lee. SIGMOD Record, Vol. 25, No. 3, septiembre 1995, pp. 23-27.

Lee (1995), “OODB design with EER”. Byung S. Lee. JOOP (Journal on Object Oriented Programming), vol. 8, nº 9. SIGS Publications, Inc., New York, febrero 1996. También disponible en:
http://www.sigs.com/publications/docs/joop/9602/joop9602.toc.html

Liskov y Wing (1993), “A New Definition of the Subtype Relation”. Barbara Liskov y Jannette M. Wing. ECOOP’93, 7th European Conference on Object Oriented Programing. Ed. O. Nierstrasz. Serie: Lecture Notes in Computer Science 707, Kaiserslautern, Germany, julio 1993, pp.118-141.

Loomis (1993), “Object and relational technologies. Can they cooperate?”. Mary E.S. Loomis. Object Magazine, enero-febrero 1993, pp.35-40.

López et al. (1994), “Constraints and Object Identity”. Gus López, Bjorn Freeman-Benson y Alan Borning. Proceedings of the 8th. European Conference on Object Oriented Programming (ECOOP’94) in Lecture Notes of Computer Science (LNCS). Ed. by Mario Tokoro y Remo Pareschi. Springer Verlag, Berlin 1994, pp.260-279.

Loy (1989), “A comparison of object-oriented and structured methods”. Patrick H. Loy. Pacific Nortwest Software Quality Conference, 1989, pp.290-303.

Luqi y Goguen (1997), “Formal Methods: Promises and Problems”. Luqi y Joseph A. Goguen. IEEE Software, enero 1997, pp. 73-85.

Maiden y Sutcliffe (1992), “Exploiting Reusable Specifications Trough Analogy”. Neil A. Maiden y Alistair G. Sutcliffe. Communications of the ACM, Vol. 35, No. 4, abril 1992, pp.55-64.

Manola y Dayal (1994), “An Overview of PDM: An Object-Oriented Data Model”. Frank Manola y Umeshwar Dayal. En: On Object-Oriented Database Systems. Proceedings of the NATO Advanced Study Institute on Object-Oriented Database Systems, Kusadasi, Turkey, agosto 1993. Eds. K.R.Dittrich, U.Dayal y A.P.Buchmann. Springer-Verlag, 1994, pp.13-27.

Manola y Mitchell (1994), "A Comparison of Object Models in ODBMS-Related Standars". Frank Manola y Gail Mitchell. Bulletin of Technical Committee on DATA ENGINEERING IEEE Computer Society. Vol. 17, Nº 4, diciembre 1994, pp. 27-35.

Manzano (1989), “Teoría de Modelos”. María Manzano. Alianza Editorial, Barcelona 1989.

Marcos (1993), “Estudio del paradigma de la orientación al objeto en los SGBD relacionales con especial énfasis en el borrador del éstandar del SQL, conocido como SQL3”. E. Marcos. Proyecto fin de carrera (dirigido por Adoración de Miguel), Facultad de Informática, U.P.M. Madrid, junio 1993.

Marcos et al. (1996a), “Caracterización del sistema de interrelaciones pra una integración de modelos de objetos”. E. Marcos, A. de Miguel, M. Piattini, P. Martínez, B. Voyer. En Actas de las II Jornadas de Informática. Ed. B. Clares. Almuñecar, julio 1996, pp. 113-122.

Marcos et al. (1996b), “Modelado conceptual en MIMO”. E. Marcos, A. de Miguel, M. Piattini y P. Martínez. “II Jornadas sobre Tecnología de Objetos”. ATI, Madrid 6-7 noviembre de 1996.

Marcos et al. (1997a), “SQL3/ODMG-93 integration through MIMO”. E. Marcos, A. de Miguel, M. Piattini, P. Martínez. Aceptado en el BIWIT’97 (Third Basque International Workshop on Information Technology). Biarritz (France), 2-4 de julio, 1997.

Marcos et al. (1997b), “About Abstract Classes”. E. Marcos, A. de Miguel, M. Piattini. Journal of Object-Oriented Programing. Sección ROAD (Report on Object Analysis & Design. SIGS Publications (aceptado para publicación en 1997).

Martin y Odell (1992), “Object-Oriented Analysis and Design”. Prentice-Hall, Englewood Cliffs, New Jersey, 1992.

Martín y Odell (1995), “Object-Oriented Methods: A Foundation”. J. Martín y J.J. Odell. Prentice Hall 1995.

Martín y Odell (1996), “Object-Oriented Methods: Pragmatic Considerations”. J. Martín y J.J. Odell. Prentice Hall 1996.

Martínez Prieto et al. (1996), “Desarrollo de SGBDOO en Oviedo3”. A.B. Martínez Prieto, J.M. Cueva Lovelle, D. Álvarez Gutiérrez. En actas de las I Jornadas de Investigación y Docencia en Bases de Datos. Ed. N. Brisaboa et al. A Coruña, julio 1996, pp. 109-118.

Mattos y DeMichiel (1994), “Recent Design Trade-offs in SQL3”. Nelson Mattos y Linda G. DeMichiel. SIGMOD RECORD, Vol. 23, No. 4, diciembre 1994, pp. 84-89.

Melton (1994), "Object Technology and SQL: Adding Objects to Relational Language". Jim Melton. Bulletin of Technical Committee on DATA ENGINEERING IEEE Computer Society. Vol. 17, Nº 4, diciembre 1994, pp.15-26.

Melton (1995), “A Flurry of Activity in the SQL Standards World”. Jim Melton. Database Programming & Design, noviembre 1995, pp. 61-63.

Mendelzon et al. (1994), "Object Migration", Alberto O. Mendelzon, Tova Milo, Emmanuel Waller. Proceedings of the Thirteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (SIGMOD/PODS 94), Minneapolis, Minnesota USA, mayo 24-26, 1994, pp. 232-254.

Meslati & Ghoul (1997), "Semantic classification: A genetic approach to classification in object-oriented models". Djamel Meslati & Said Ghoul. Journal of Object-Oriented Programing. SIGS Publications, enero 1997, Vol.9, nº 8, pp.25-37.

Meyer (1988), "Object-Oriented Software Construction". Bertrand Meyer. Prentice Hall International (UK), 1988.

Moore (1996), "An Ode to persistence". Dana Moore. Journal of Object-Oriented Programing. SIGS Publications, noviembre-diciembre 1996, Vol.9, nº 7, pp.29-33.

Morejon (1994), "MERISE: Vers une modélisation orientée object". Les Éditions d'Organisation, 1994.

Motschnig-Pitrik (1993), "The Semantic of Parts Versus Aggregates in Data/Knowledge Modelling". Renate Motschnig-Pitrik. CAiSE'93, 5th International Conference on Advanced Information Systems Engineering. Eds. Colette Rolland, François Bodart y Corine Cauvet. Serie: Lecture Notes in Computer Science 685, París, Francia, junio 1993, pp.352-373.

Motschnig-Pitrik y Mylopoulus (1996), "Semantics, Features, and Applacations of the Viewpoint Abstraction". Renate Motschnig-Pitrik y John Mylopoulos. CAiSE'96, 8th International Conference on Advanced Information Systems Engineering. Eds. Panos Constantopoulos, Jon Mylopoulos y Yannis Vassiliou. Serie: Lecture Notes in Computer Science 1080, Heraklion, Crete, Greece, mayo 1996, pp.314-339.

Nagel (1974), "La estructura de la ciencia" Ernest Nagel. Editorial PAIDOS S.A.I.C.F., Argentina. 2ª edición, 1974.

Newell (1992), "The Knowledge Level". Allen Newell. Artificial Intelligence, Vol.18 Nº1, 1982, pp. 87-127.

OMG (1992), "Object Analysis and Design", Volume 1, Reference model (Draft 7.0). OMG Object Analysis and Design Special Interest Group. OMG, Inc. Framingham, MA, USA, octubre 1992.

OMG (1995a), "CORBA: Common Object Request Broker Architecture and Specification". Object Management Group (OMG), Inc. Framingham, MA 01701, USA 1995.

OMG (1995a), "CORBAservices: Common Object Services Specification". Object Management Group (OMG), Framingham, MA, USA 1995.

OMG TC (1995), "Object Analysis & Design RFI". OMG TC Document 95-9-35. OMG, Framingham, MA, USA, diciembre 1995.

O´Neill (1991), "Worlds Without Content. Against formalism". John O´Neill. Routledge, London 1991.

Panet et al. (1991) , ”De MERISE à MERISE/2: Techniques de ‘refinement’ et nouvelles architectures d’applications”. Panet G., Letouche R., Peugeot C. Congrès Autour et à l’entour de MERISE, AFCET-CERAM, Sophia Antipolis, 1991

Pastor y Ramos (1995), “OASIS 2.1.1: A Class -Definition Language to Model Information Systems Using an Object-Oriented Aproach”. Dpto. de Sistemas Informáticos y Computación, SPUPV-95.788. Servicio de Publicaciones de la Universidad Politécnica de Valencia, marzo 1995.

Piattini (1994), "Definición de una metodología de desarrollo para bases de datos orientadas al objeto fundamentadas en extensiones del modelo relacional". Universidad Politécnica de Madrid. Facultad de Informática 1994.

Popper (1985), “Realismo y el objetivo de la ciencia”. K. Popper. TECNOS, Madrid 1985.

Prieto-Díaz (1993), “Status Report: Software Reusability”, Rubén Prieto-Díaz. IEEE Software 1993, pp. 61-66.

Quer y Olivé (1993), “Object Interation in Object-Oriented Deductive Conceptual Models”. Carmen Quer y Antoni Olivé. CAiSE’93, 5th International Conference on Advanced Information Systems Engineering. Eds. Colette Rolland, François Bodart y Corine Cauvet. Serie: Lecture Notes in Computer Science 685, París, Francia, junio 1993, pp.374-393.

RAE (1992), “Diccionaro de la Lengua Española”. Real Academia Española. Vigésima primera edición, 1992.

Ram (1994), “Automated Tools for Database Design: State of the Art”. Sudha Ram. Working Paper, Departament of MIS, University of Arizona, 1994.

Ram (1995), “Deriving Functional Dependencies from the Entity-Relationship model”. Sudha Ram. Communications of the ACM, Vol. 38, No.9, septiembre 1995, pp. 95-107.

Riehle (1996), “Ada and the notion of class”. Richard Riehle. Journal of Object-Oriented Programing. SIGS Publications, noviembre-diciembre 1996, Vol.9, nº 7, pp.66-74.

Richey (1995), “Condition Handling in SQL Persistent Stored Modules”. Jeff Richey. SIGMOD Record, Vol. 24, No. 3, septiembre 1995, pp.98-103.

Robinson y Berrisford (1994), “Object-Oriented SSADM. Keith Robinson y Graham Berrisford. Prentice Hall, 1994.

Rochfeld (1992), “OOM: évolution de MERISE vers une aproche object. Journées AFCET Ingénierie des bases de données, 1992.

Rosenthal y Reiner (1994), “Tools and Transformations -Rigorous and Otherwise- for Practical Database Design”. Arnon Rosenthal y David Reiner. ACM Transactions on Database Systems, Vol. 19, No. 2, junio 1994, pp.167-211.

Rumbaugh et al. (1991), “Object-Oriented Modeling and Design”. Prentice-Hall, Englewood Cliffs, New Jersey, 1991.

Rumbaugh (1996), “Packaging a system: Showing architectural dependencies”. James Rumbaugh. Journal of Object-Oriented Programing. SIGS Publications, noviembre-diciembre 1996, Vol.9, nº 7, pp.11-18.

Russell (1977), “Los Principios de la Matemática”. Russell, Bertrand. Espasa Calpe, Madrid 1977

Saltor et al. (1991), “On Canonical Models for Federated Dbs”. SIGMOD RECORD, Vol. 20, nº 4, diciembre 1991, pp. 44-48.

Sane y Campbell (1995), “Object-Oriented State Machines: Subclassing, Composition, Delegation, and Genericity”. Aamod Sane y Roy Campbell. En Proceedings de OOPSLA’95. 10th. Conference on Object-Oriented Programing Systems, Languages ans Applications, Austin, Texas, 1995, pp. 17-32.

Schenck y Wilson (1994), “Information Modeling: The EXPRESS Way”. Douglas Schenck y Peter Wilson. Oxford University Press, Inc. New York 1994.

Shlaer y Lang (1996a), “Competitive Relationships”. Sally Shlaer y Neil Lang. Report on Object Analysis & Design, SIGS Publications, Inc. NY, marzo-abril 1996, pp. 5-7.

Shlaer y Land (1996b), “Reflexive Relationships”. Sally Shlaer y Neil Lang. Report on Object Analysis & Design, SIGS Publications, Inc. NY, mayo-junio 1996, Vol.2, nº 7, pp.39-40.

Shlaer y Mellor (1990), “Object-Oriented Systems Analysis: Modeling the World in Data”. Prentice-Hall, Englewood Cliffs, New Jersey, 1990.

Soley y Stone (1995), “Object Management Architecture Guide (Revision 3.0)”. Richard Mark Soley y Christopher M. Stone. Object Management Group, Inc. (OMG), Framingham, Massachusetts USA. Third Edition, junio 1995.

Soloviev (1992), "An Overview of Three Commercial Object-Oriented Database Management Systems: ONTOS, ObjectStore, and O2". Valery Soloviev. SIGMOG RECORD, vol.21, nº1, marzo 1992, pp. 93-104.

Sourrouille (1996), “Should Subclasses Inherit All States and Transitions?”. Jean Louis Sourrouille. Report on Object Analysis & Design, SIGS Publications, Inc. NY, marzo-abril 1996, pp.19-21.

Spiby (1991), "EXPRESS Language Reference Manual". Ed. Philip Spiby. ISO TC184/SC4/WG5/P3, abril 1991.

Spiby (1994), "Product Data Representation and Exange: Part 11-EXPRESS Language Reference Manual". Ed. Philip Spiby. ISO TC184/SC4/WG5/ N65 (P2), agosto 1994.

SQL/MM: MAD-024 (1996), “Conceptual Schema Language”. Arne-Jorgen Berre. ISO/TC211 WG1-WI 3, Working Draft 1.0, agosto 1996.

Steels (1996), “Beyond Objects”. Luc Steels. In Object-Oriented Programing, 8th European Conference (ECOOP’94), Bologna, Italy. Ed. by Tokoro y Pareschi. Springer, Lecture Notes in Computer Science 821. New York, 1994, pp. 1-11.

Stonebraker et al. (1990), "Third Generation Database System Manifesto". M. Stonebraker, B. Lindsay, J. Gray, M. Carey, M. Brodie, P. Bernstein, D. Beech. ACM SIGMOD Record vol. 19, nº 3, septiembre 1990.

Storey y Goldstein (1993), "Knowledge-Based Approaches to Database Design". Veda C. Storey y Robert C. Goldstein. MIS Quarterly, marzo 1993, pp. 25-46.

Stroustrup (1991), "The C++ Programming Language". Bjarne Stroustrup. Addison-Wesley Publishing Company, 1991.

SUMM (1992), "SUMM (Semantic Unification Meta-Model)". Dictionary/Methodology Committee of the IGES/PDES Organization. ISO TC184/SC4/WG3 N103 (V.1.0), octubre 1992.

Tardieu et al. (1983), "Méthode MERISE -Tome 1: Principles et outils. Tardieu H., Rochfeld A., Colletti R. Les Editions d'Organisation, Paris 1983

Tardieu (1988), "MERISE: 2ème génération". SEMA-METRA, 1988.

Tsichritzis y Lochovsky (1982), "Data Models". Dionysios C. Tsichritzis and Frederick H. Lochovsky. Prectice-Hall, Inc. Englewood Clifs, New Jersey, 1982.

Vasan (1993), "Techniques for object and relational integration. Handling the interaction between the two". Robin Vasan. Object Magazine Vol. 3, mayo-junio 1993, pp. 52-53, 60.

Vela y Serrano (1997), "MIMO-CASE: diseño e implementación de una herramienta de modelado conceptual orientado al objeto". Belén Vela y José Serrano. Proyecto fin de carrera (dirigido por Esperanza Marcos). Escuela Politécnica Superior. Universidad Carlos III de Madrid. Leganés, 1997.

Vernon (1992), "Relational Database Design Subject Area Working Paper". John Vernon. EIA CDIF Technical Committee. Working Document. Reference: CDIF-JV-N1-V2, abril 1992.

Vossen (1995), "On Formal Models for Object-Oriented Databases". Gottfried Vossen. OOPS Messenger, Vol.6, No. 3, julio 1995, pp. 1-19.

Wade (1996), "Object Query Standards". Andrew E. Wade. SIGMOD Record, Vol. 25, No. 1, marzo 1996, pp.87-92

Wegner (1995), "A perspective on Object-Oriented Research". Peter Wegner. Theory And Practice of Object Systems, Vol. 1(2), 1995, pp. 133-143.

Weir (1996), "Improve Your Sense of Ownership". Charles Weir. Report on Object Analysis & Design, SIGS Publications, Inc. NY, marzo-abril 1996, pp. 16-18.

Wells y Thompson (1994), "Evaluation of the Object Query Service Submission to the Object Management Group". David L. Wells y Craig W. Thompson. Bulletin of Technical Committee on DATA ENGINEERING IEEE Computer Society. Vol. 17, Nº 4, diciembre 1994, pp. 36-45.

Wertz (1989), "The DATA Dictionary: Concepts and Uses". Charles J. Wertz. Second Edition. QED Information Sciences, Inc., Wellesley, MA, USA, 1989.

Wieringa et al. (1994), "Roles and dynamic subclasses: a modal logic approach". R. J. Wieringa, W. de Jonge, P. A. Spruit. Ed. by Tokoro y Pareschi. In Object-Oriented Programing, 8th European Conference (ECOOP'94). Springer, Lecture Notes in Computer Science 821. New York, 1994, pp. 32-59

Winston et. al (1987), "A taxonomy of part-whole relations". Cognitive Science Vol. 11, 1987, pp. 417-444.

Wieringa et al. (1995), "Using Dynamic Classes and Rol Classes to Model Object Migration". R. Wieringa, W. de Jonge y P. Spruit. Theory And Practice of Object Systems, Vol. 1(2), 1995, pp. 61-83.

Wieringa y de Jones (1995), "Object Identifiers, Keys, and Surrogates: Object Identifiers Revisited". R. Wieringa y W. de Jonge. Theory And Practice of Object Systems, Vol. 1(2), 1995, pp. 101-114.

Wirfs-Brock et al. (1990), "Design Object-Oriented Software". R. Wirfs-Brock, B. Wilkerson y L. Wiener. Prentice-Hall 1990.

Wold y Lehne (1996), "Working With Objects. The OOram Software Engineering Method". Manning Publications Co., Greenwich, 1996

Yourdon et al. (1995), "Mainstream Objects: An Analysis And Design Approach for Bussiness". E. Yourdon, K. Whitehead y J. Thumann. Prentice-Hall 1995.

Zaccagnini (1994), "La metodología experimental". En Metodología y teoría de la psicología. Ed. Morales Dominguez, J. F. Universidad Nacional de Educación a Distancia, Madrid 1994.

Zdonik (1994), "What Makes Object-Oriented Database Management Systems Different?". Stanley B. Zdonik. En: Advances in Object-Oriented Database Systems. Proceedings of the NATO Advanced Study Institute on Object-Oriented Database Systems, 1993. Eds.Asuman Dogac, M. Tamer Özsu, Alexandros Biliris y Timos Sellis. Springer-Verlag, 1994, pp. 3-26.

B.2. Lugares de INTERNET

GRUPO DE NEWS

news:comp.databases.object/ es un grupo de interés en bases de datos orientadas al objeto. En él pueden plantearse dudas o cuestiones de interés a cerca de las bases de datos orientadas al objeto. En este grupo participan personas de reconocido prestigio en este ámbito, como por ejemplo el propio Jim Melton (editor del SQL3).

JOOP

En **http://www.sigs.com/publications/docs/joop/** se encuentra el directorio del Journal on Object Oriented Programming (JOOP) on line. JOOP es una revista de SIGS Publications donde escriben regularmente personas de reconocido prestigio en la orientación al objeto como Rumbaugh, Booch, et. JOOP se publica en papel y en internet.

KADS-II

En **http://swi.psy.uva.nl/projects/CommonKADS/descrition/root.htm** se encuentra una descripción del proyecto ESPRIT KADS-II (P5248).

ODMG-93

http://www.odmg.org/ es el lugar del ODMG. En esta página puede encontrarse información relativa a dicho grupo (miembros, funcionamiento, etc.) así como información relativa al estándar ODMG-93 (estado actual, bibliografía, etc.). También puede encontrarse una relación de los niveles de cumplimiento del el estándar de diversos gestores de objectos del mercado (O2, Object Store, POET, etc.)

OMG

http://www.rational.com/pst/tech_papers/ es el lugar del OMG. En esta página pueden encontrarse enlaces a CORBA, así como referencias de publicaciones y libros. Se puede encontrar también información referente a las actividades y miembros del grupo.

POET

En **http://www.poet.com/mainpage.htm** se encuentra la información relativa a POET. En esta dirección pueden encontrarse publicaciones relativas a POET, así como los manuales y una versión gratuita de POET.

RATIONAL

http://www.rational.com es el lugar de Rational Software Corporation. Pueden encontrarse en él otros documentos de interés como el borrador del Método Unificado y de UML.

http://www.rational.com/pst/tech_papers/ es un índice con los documentos disponibles.

En **http://www.rational.com/pst/tech_papers/uml_rt.html#overview** se encuentra la versión 2.0 de *Unified Modelling Language for Real-Time System Design.*

SIGS Publications

En **http://www.sigs.com/publications/** pueden encontrarse enlaces a diversas publicaciones del grupo SIGS (Expert Object, Object Magazin, etc.), así como listados de libros, conferencias, etc.

SQL3

En **speckle.ncsl.nist.gov** pueden encontrarse los documentos de estándares del SC21 de ISO y en concreto los del SQL3 y SQL/MM. Este servidor tiene una estructura de árbol. Los documentos del SQL3 pueden encontrarse en el directorio *dbl/basedocs* (este es de acceso restringido a los miembros del grupo de trabajo y en él se encuentran los estándares y borradores de estándares) y en *dbl/lhr*, *dbl/mci*, *dbl/mad,* etc. para los documentos de acceso público (propuestas para los estándares).

En abril de 1997, este servidor cambió su ubicación (en la actualidad se mantienen los dos servidores activos). La nueva dirección es: **jerry.ece.umassd.edu**. Este servidor mantiene la misma estructura de directorios que el anterior.

http://www.jcc.com/sql_stnd.html#Current Status es el lugar del SQL3. En sus páginas pueden encontrarse enlaces a los últimos borradores del estándar, así como enlaces a otros grupos de trabajo relacionados con el SQL3 (SQL/MM, RDA, etc).

http://epoch.cs.berkeley.edu:8000/sequoia/dba/montage/FAQ/SQL_TOC.html recoge las preguntas más frecuentes rrespecto al SQL junto con sus respuestas,

Agradecimientos

En primer lugar, quiero dar las gracias a mis directores de Tesis, sin cuya guía este trabajo no hubiera sido posible. En especial: a Adoración por confiar en mí; a Mario, porque me impulsó a continuar.

A “mis niños” del laboratorio, especialmente a Belén, Jose, Santi y Roberto: gracias, no sólo por ayudarme con los problemas técnicos, sino también por haber “adoptado” a MIMO. He aprendido mucho con vosotros.

Quiero agradecer a todos los miembros del SC21/WG3 de ISO por darme la posibilidad de *vivir* la elaboración del SQL3. Especialmente, a Hugh Darwen y Jim Melton, cuya sabiduría admiro, porque su participación en las reuniones me ha permitido presenciar discusiones, especialmente interesantes, sobre la tecnología de bases de datos. Al MAP, MINER, SEDISI y a la Universidad Carlos III por el apoyo económico para el seguimiento del SQL3. Sin esta ayuda no hubiera sido posible asistir a reuniones como la de Kansas City, ni organizar la última reunión en Madrid.

A Andrés Marín, por sus consejos y comentarios sobre formalización, le agradezco su ayuda desinteresada.

A Fredy, por facilitarme bibliografía y aportarme ideas desde una perspectiva radicalmente distinta y complementaria como es la filosofía de ciencia.

A mis compañeros de trabajo y amigos, por su paciencia y comprensión durante este tiempo. A Paloma, a Almudena, a Joaquín, a Jose Ignacio y a Juan; a todos ellos, en especial, por “estar ahí”.

A mi familia, la de verdad, y la adoptiva de Madrid.