Diseño conceptual de bases de datos

PID_00270598

Jordi Casas Roma
Josep Cuartero Olivera

Tiempo mínimo de dedicación recomendado: 6 horas

Universitat Oberta de Catalunya

Jordi Casas Roma

Licenciado en Ingeniería Informática por la Universitat Autònoma de Barcelona (UAB), máster en Inteligencia Artificial Avanzada por la Universidad Nacional de Educación a Distancia (UNED) y doctor en Informática por la UAB. Desde 2009 ejerce como profesor en los Estudios de Informática, Multimedia y Telecomunicación de la Universidad Oberta de Catalunya (UOC), y también como profesor asociado en la UAB. Es director del máster universitario en Ciencia de Datos de la UOC.

Josep Cuartero Olivera

Ingeniero informático y máster en Sociedad de la Información y del Conocimiento en la Universitat Oberta de Cataluña (UOC). Profesional informático en el sector privado ejerciendo de experto en sistemas de información. Es profesor colaborador de la UOC.

La revisión de este recurso de aprendizaje UOC ha sido coordinada por la profesora: Àngels Rius Gavidia

Quinta edición: febrero 2020

Av. Tibidabo, 39-43, 08035 Barcelona
Autoría: Jordi Casas Roma, Josep Cuartero Olivera
Producción: FUOC

Índice

Introducción

El diseño conceptual de bases de datos es la segunda etapa en el proceso de diseño de una base de datos, situándose después de la etapa de recogida y análisis de requisitos.

En esta etapa, partiendo del análisis detallado del problema descrito por el usuario final, se obtiene una estructura de la información independiente de la tecnología llamada modelo o esquema conceptual. Este esquema no es más que una representación gráfica, en un modelo de datos de alto nivel, que permite la resolución del problema planteado con independencia de la tecnología de implementación. Es decir, sin tener en cuenta aspectos tales como el tipo de base de datos (relacional, orientada a objetos, etc.), el sistema de gestión de la misma o el lenguaje de acceso a datos que con el que se implementará.

El objetivo de esta etapa es elaborar un esquema conceptual que describa el problema en cuestión. Uno de los modelos más utilizados a la hora de implementar el modelo conceptual es el modelo entidad-interrelación (modelo ER). Dicho modelo ha tenido una gran relevancia hasta nuestros días, aunque en la actualidad el uso del lenguaje unificado de modelización (UML) ha tomado mayor importancia en todos los ámbitos de la computación a causa, entre otras cosas, de la potencia y de la flexibilidad que ofrece –a pesar de que fuera inicialmente concebido para modelizar sistemas de software–.

En este módulo, concretamente, explicaremos los principios del diseño conceptual y como llevar a cabo este diseño mediante diagramas en lenguaje UML.

Objetivos

En los materiales didácticos de este módulo encontraréis las herramientas indispensables para lograr los objetivos siguientes:

1. Conocer los fundamentos del diseño conceptual de bases de datos.

2. Entender los elementos básicos de modelización como mecanismo de representación conceptual de datos.

3. Estudiar los diagramas de clases UML como herramienta para representar modelos de datos.

4. Ser capaces de leer y entender diagramas de modelos conceptuales de bases de datos.

5. Aprender a modelizar, mediante diagramas UML, las necesidades de un sistema de información a partir de una descripción de los requisitos iniciales.

1. Introducción al diseño conceptual

El diseño conceptual es la segunda etapa en el proceso de diseño e implementación de una base de datos, y sigue a la etapa inicial de recopilación y análisis de requisitos. El objetivo del diseño conceptual es crear un esquema conceptual para la base de datos mediante un modelo de datos conceptual de alto nivel e independiente de la tecnología a partir del análisis de requisitos.

Un **esquema conceptual** es una descripción concisa de los requisitos de datos por parte de los usuarios e incluye descripciones detalladas de las entidades que están involucradas, las relaciones entre estas entidades y las restricciones de integridad que tienen. Se expresa mediante conceptos proporcionados por un modelo de datos de alto nivel, que debe ser fácil de entender y no debe contener detalles de implementación.

1.1. El diseño conceptual

La etapa de diseño conceptual parte de la recopilación y el análisis de requisitos obtenido en la etapa previa con la ayuda de los futuros usuarios de la base de datos, y tiene como objetivo conseguir un esquema conceptual de la base de datos que sea consistente con los requisitos y las restricciones impuestas por los problemas que hay que resolver.

El esquema conceptual, además, debe servir como referencia para verificar que se han considerado todos los requisitos y que no hay conflicto entre ellos. Si algunos de los requisitos iniciales no se pueden representar gráficamente en el modelo conceptual, deben añadirse de manera textual, para asegurarnos de que quedan recogidos en el modelo conceptual y de que se tendrán en cuenta en las fases siguientes del diseño de la base de datos. Hay diferentes lenguajes para representar modelos conceptuales de datos. Algunas características importantes para representar modelos conceptuales de datos son estas:

- **Expresividad**: el modelo debe ser suficientemente expresivo para permitir representar los diferentes conceptos del mundo real y las asociaciones entre estos conceptos.

- **Simplicidad**: el modelo debe ser simple. Ha de permitir a los usuarios de la base de datos entender los conceptos que se expresan.

- **Representación diagramática**: el modelo debe utilizar una notación diagramática fácil de entender y útil para visualizar el esquema conceptual.

- **Formalidad**: la representación del modelo debe ser precisa, formal y no puede presentar ambigüedades.

Satisfacer estas características no es fácil y a menudo algunas características entran en conflicto con las demás.

Además de las características comentadas, hemos dicho que un esquema conceptual es una descripción de alto nivel e independiente de la tecnología. Estas dos características son importantes para un lenguaje que permita definir un modelo conceptual. Hay muchas razones que destacan estas dos características como importantes en el modelo conceptual, entre las cuales se pueden señalar las siguientes:

- El uso del modelo conceptual permite a los diseñadores de bases de datos concentrarse en expresar las propiedades, las asociaciones y las restricciones de los datos sin preocuparse de los detalles de implementación y con independencia de las particularidades de los diferentes sistemas gestores de bases de datos (SGBD[1]).

[(1)]SGBD es la sigla de *sistema gestor de bases de datos*.

- El esquema conceptual se convierte en un elemento de descripción estable del contenido de la base de datos.

- Un modelo de alto nivel independiente de las tecnologías específicas de cada SGBD permite más expresividad y un carácter más general.

- El modelo se puede utilizar como vehículo de comunicación entre los usuarios, los diseñadores y los analistas de la base de datos.

1.1.1. Metodologías de diseño

A partir de los requisitos recopilados y trabajados en la primera fase del proceso de diseño de una base de datos se pueden establecer dos metodologías básicas para el diseño conceptual de bases de datos:

1) Metodología centralizada (***one shot***): en esta primera metodología se fusionan todos los requisitos de los distintos grupos de usuarios y aplicaciones recopilados en la primera fase en un solo conjunto antes de empezar la fase de diseño. A partir de este único conjunto de requisitos se desarrolla un único esquema conceptual para toda la base de datos.

2) Metodología de integración de vistas: en esta metodología se construye un esquema (denominado *vista*) para cada conjunto de requisitos de cada grupo de usuarios y se validan de manera independiente. Posteriormente, en una subfase llamada *integración de vistas* hay que fusionar los diferentes esquemas o vistas en un único esquema conceptual global para toda la base de datos.

La metodología de integración de vistas permite crear esquemas conceptuales más pequeños que son validados por los diferentes grupos de usuarios de la base de datos, pero añade la subfase de integración de vistas para unificar los diferentes esquemas. Además, esta subfase añade una complejidad notable y requiere una metodología y unas herramientas específicas para poder llevarse a cabo de manera correcta. Generalmente, esta metodología se suele aplicar al diseño de grandes bases de datos.

En este material nos centraremos en la metodología centralizada.

1.1.2. Estrategias de diseño

A partir de un conjunto de requisitos, recopilados para un único usuario o para un conjunto de ellos, hay que diseñar un esquema conceptual que los satisfaga. Para llevar a cabo esta tarea, se pueden utilizar diferentes estrategias. La mayoría utilizan un método incremental, es decir, empiezan por modelizar algunas estructuras principales derivadas de los requisitos y las van modificando y refinando en diferentes iteraciones del mismo proceso.

Algunas de las estrategias más utilizadas para el diseño conceptual de bases de datos son las siguientes:

a) Estrategia descendente: se empieza el proceso con un esquema que contiene abstracciones de alto nivel y se van aplicando sucesivamente refinamientos sobre estas abstracciones.

b) Estrategia ascendente: se empieza el proceso con un esquema que contiene los detalles de las abstracciones y se van combinando de manera sucesiva formando conjuntos de más entidad o conceptos complejos.

c) Estrategia de dentro hacia fuera: es un caso concreto de la estrategia ascendente. Esta estrategia centra la atención en un conjunto central de conceptos que son muy evidentes y, poco a poco, incluye en el esquema nuevos conceptos relacionados directamente con los conceptos existentes.

d) Estrategia mixta: esta estrategia divide los requisitos según una estrategia descendente y entonces construye cada una de las divisiones siguiendo una estrategia ascendente. Finalmente se combinan las partes para generar el esquema completo.

1.2. El modelo ER

El **modelo entidad-interrelación**, o **modelo ER**, es un modelo conceptual de datos de alto nivel e independiente de la tecnología. Este modelo y las variaciones que tiene constituyen los modelos más utilizados por el diseño conceptual de las aplicaciones de bases de datos. Esto se debe, principalmente, a la simplicidad y la facilidad de uso. Los principales elementos que incluye el modelo son las entidades, los atributos y las relaciones entre entidades.

El objetivo principal del modelo ER es permitir a los diseñadores reflejar en un modelo conceptual los requisitos del mundo real que sean de interés para el problema.

Modelo ER

El nombre original proviene de la lengua inglesa, en la que las siglas *ER* significan *entity-relationship*. En español encontramos autores que lo traducen como *modelo entidad-relación* y otros que lo traducen como *modelo entidad-interrelación*. Ambos se refieren al mismo modelo.

Este modelo facilita el diseño conceptual de una base de datos y es aplicable al diseño de cualquier tipo de bases de datos (relacionales, orientadas a objetos, etc.), ya que en la etapa del diseño conceptual no se tiene en cuenta todavía la tecnología concreta que se empleará para implementar la base de datos.

El modelo ER tiene el origen en los trabajos hechos por Peter Chen en 1976. Posteriormente, otros muchos autores han propuesto variantes y ampliaciones para este modelo. Por lo tanto, en la literatura se pueden encontrar variaciones en el modelo ER que pueden diferir simplemente en la notación diagramática o en algunos conceptos en los que se basan para modelizar los datos.

Peter Pin-Shan Chen

Es científico y profesor de informática en la Universidad Estatal de Luisiana. Obtuvo el grado de doctor en la Universidad de Harvard en 1973. Posteriormente, en 1976, desarrolló el modelo ER, hecho por el que es conocido.

El modelo ER permite reflejar aspectos relacionados con la estructura de los datos y de la integridad de estos datos.

Para representar el modelo ER se ha empleado tradicionalmente el **diagrama entidad-interrelación**, o **diagrama ER**. Este diagrama define una nomenclatura específica para representar los diferentes conceptos de entidades y relaciones que requiere el modelo. A pesar del amplio uso de este diagrama, últimamente se tiende a emplear el lenguaje UML para representar el modelo ER. En este texto utilizaremos el lenguaje UML. No obstante, hay que tener en cuenta que los conceptos son los mismos y solo cambia la manera de representar los conceptos relacionados con las entidades, los atributos o las relaciones.

1.3. El lenguaje UML

El **lenguaje unificado de modelización**[2] **(UML)** es un lenguaje de propósito general para modelizar sistemas de software. El estándar fue creado y es mantenido por, el Object Management Group. Se añadió por primera vez a la lista de tecnologías empleadas por el OMG en 1997 y desde entonces se ha convertido en el estándar de la industria para modelizar sistemas de software.

(2)En inglés, *unified modeling language* (UML).

UML es un lenguaje gráfico diseñado para especificar, visualizar, modificar, construir y documentar un sistema. Permite una visualización estándar de diferentes artefactos, entre otros, actividades, actores, lógicas de negocio y esquemas de bases de datos.

Object Management Group

El Object Management Group (OMG) es un consorcio dedicado a establecer y promover varias especificaciones de tecnologías orientadas a objetos, como UML, XMI o CORBA. Es una organización sin ánimo de lucro que promueve el uso de la tecnología orientada a objetos mediante guías y especificaciones para estas guías.

El lenguaje UML lo desarrolló la compañía Rational Software Corporation durante los años noventa. Después de un periodo en el que coexistieron diferentes tendencias en la modelización orientada a objetos, la compañía unificó los esfuerzos de tres pioneros en esta área: James Rumbaugh, Grady Booch e Ivar Jacobson. En 1996 se creó el consorcio UML Partners con el objetivo de completar la especificación de UML. En enero de 1997 se publicó la versión 1.0 de UML. Durante el mismo año se publicó la versión 1.1, que aceptó y adoptó el consorcio OMG. Finalmente, la versión 2.0 se publicó en el 2005. En el momento de escribir este material, la última versión es la 2.3, publicada en mayo del 2010.

El lenguaje UML incorpora una gran cantidad de diagramas que permiten representar el modelo de un sistema desde diferentes perspectivas. Podemos encontrar diagramas que hacen referencia a la estructura (por ejemplo, diagramas de clase, diagramas de componentes o diagramas de paquetes), diagramas referentes al comportamiento (por ejemplo, diagramas de actividad o diagramas de casos de uso) o diagramas referentes a la interacción (por ejemplo, diagramas de comunicación o diagramas de secuencia).

UML define nueve tipos de diagramas divididos en dos categorías:

- **Diagramas estructurales**: describen las relaciones estructurales o estáticas entre los diferentes componentes del sistema.

- **Diagramas de comportamiento**: describen las relaciones de comportamiento o dinámicas entre los diferentes componentes del sistema.

Para el diseño conceptual de bases de datos nos interesa especialmente el diagrama de clases, que permite representar información del dominio de discurso. Aun así, a continuación veremos una breve descripción de cada uno de estos diagramas, a pesar de que el alcance de los diferentes diagramas UML cae fuera de los objetivos de este texto.

Dominio del discurso

El dominio del discurso, universo del discurso, o simplemente dominio, es el conjunto de cosas de las que se habla en un determinado contexto.

Diagramas estructurales:

a) Los **diagramas de clases** son diagramas estáticos que describen la estructura de un sistema a partir de las clases del sistema, los atributos de este sistema y las relaciones que se establecen entre éstas (también conocidas como *asociaciones* en terminología UML). Estos diagramas son uno de los principales bloques en el desarrollo orientado a objetos, pero también han demostrado una capacidad excelente para modelizar datos. Por este motivo, han sido cada vez más importantes en el diseño conceptual de bases de datos.

b) Los **diagramas de objetos** muestran las instancias (u objetos del mundo real) y las relaciones entre estas en un momento concreto. Las instancias de un sistema se modifican a lo largo de tiempo, es decir, se crean instancias nuevas, se destruyen instancias y se modifican ciertos valores de determinadas instancias. Por lo tanto, los diagramas de objetos son como una fotografía que muestra una vista estática de las diferentes instancias de un sistema y de las relaciones entre estas instancias en un instante de tiempo determinado.

c) Los **diagramas de componentes** ilustran las organizaciones y las dependencias entre los componentes del sistema. En entornos de bases de datos se utilizan para modelizar los espacios de tabla o las particiones.

Ved también

Los espacios de tabla (*tablespaces*) se ven en el subapartado 5.2 del módulo "Diseño físico de bases de datos" de esta asignatura.

d) Los **diagramas de implantación** representan la distribución de componentes del sistema y su relación con los componentes del hardware disponible.

Diagramas de comportamiento:

a) Los **diagramas de casos de uso** se utilizan para modelizar las interacciones funcionales entre los usuarios y el sistema.

b) Los **diagramas de secuencia** describen las interacciones entre distintos objetos en el transcurso del tiempo. Muestran el flujo temporal de mensajes entre varios objetos.

c) Los **diagramas de colaboración** representan las interacciones entre objetos como una serie de mensajes en secuencia. Estos diagramas centran la atención en la organización estructural de los objetos que envían y reciben mensajes.

d) Los **diagramas de estado** describen cómo cambia el estado de un objeto en respuesta a diferentes acontecimientos externos.

e) Los **diagramas de actividad** presentan una vista dinámica del sistema y modelizan el flujo de control de actividad a actividad.

2. Elementos básicos de modelización

Un modelo conceptual permite definir información de dominio a partir de tipos de entidades, tipos de relaciones, atributos y restricciones de integridad.

2.1. Tipos de entidades

Entendemos por **entidad** un objeto del mundo real, que tiene identidad propia y que es distinguible del resto de los objetos.

Las entidades pueden ser objetos con existencia física (por ejemplo, un coche o una persona) u objetos con existencia conceptual (por ejemplo, un trabajo o una asignatura de un curso universitario).

Ejemplos de entidades

Algunos ejemplos de entidad son una manzana, un coche o una persona. Otros ejemplos, que no son tangibles pero que tienen identidad y son distinguibles del resto, son un pedido a un proveedor, una petición de préstamo en una biblioteca o una incidencia municipal.

Entendemos por **tipo de entidad** la abstracción que permite definir el conjunto de objetos con las mismas propiedades, comportamiento común, semántica común e idéntica relación con los demás objetos.

El proceso para pasar de un conjunto de entidades a un tipo de entidad se denomina **abstracción**. Este proceso consiste en eliminar las diferencias o distinciones entre las entidades para poder observar aspectos comunes a todas estas entidades.

Ejemplos de entidades

El tipo de entidad *coche* se define como el concepto de *vehículo* que describe las partes comunes de diferentes coches con independencia de la marca, el color, el modelo y otras características concretas. Un coche concreto, por ejemplo mi coche, es una instancia u ocurrencia del tipo de entidad *coche*.

Utilizaremos la nomenclatura siguiente para diferenciar estos dos conceptos:

- El concepto que hemos definido como *entidad* también puede ser referenciado por los términos *objeto*, *instancia* u *ocurrencia*.
- El concepto que hemos definido como *tipo de entidad* también puede ser referenciado por los términos *clase*, *entidad tipo* o *tipo*.

Terminología UML

En la representación mediante diagramas UML, los tipos de entidad se denominan *clases* y las entidades, *objetos*.

En la literatura se pueden encontrar autores que utilizan las diferentes terminologías y, por lo tanto, hay que tenerlas todas presentes para identificar de manera rápida y unívoca a qué concepto se hace referencia cuando se utiliza cualquiera de estos términos.

2.2. Atributos

Un **atributo** es una propiedad que tienen todas las entidades de un mismo tipo de entidad y que permite representar sus características.

Atributos y relaciones binarias

En teoría, un atributo no es más que un tipo de relación binaria en la que uno de los participantes es un tipo de dato (podéis ver el subapartado siguiente). No obstante, para simplificar el modelo y facilitar su creación y legibilidad se tratan de manera diferente y más compacta.

Para representar gráficamente los tipos de entidad y sus atributos, utilizaremos los diagramas de clases del modelo UML.

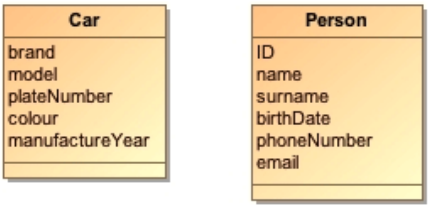
Diagrama de clases

El tipo de entidad 'coche' tiene un conjunto de atributos; por ejemplo: marca, modelo, matrícula, color y año de fabricación. Una entidad de *coche* tiene valores específicos para cada uno de estos atributos. Por ejemplo, un coche puede ser de la marca Fiat, modelo Punto, matrícula 3621-GHV, color negro y fabricado en el 2010.

De manera similar, el tipo de entidad 'persona' tiene un conjunto de atributos; por ejemplo: identificación (en el caso de España es el documento nacional de identidad o DNI), nombre y apellidos, fecha de nacimiento, número de teléfono y correo electrónico. Una entidad de 'persona' puede ser, por ejemplo, una persona con DNI 33551058D, nombre Fred y apellido Smith, nacido el 1 de marzo de 1968, con número de teléfono 85542154 y correo electrónico fredsmith@mydom.com.

La figura 1 muestra un ejemplo de representación de los tipos de entidad 'coche' (*Car*) y 'persona' (*Person*).

Figura 1. Ejemplos de los tipos de entidad 'coche' (*Car*) y 'persona' (*Person*)



En los diagramas de clases del modelo UML representaremos los tipos de entidad (denominados *clases* en los diagramas UML) mediante un rectángulo con tres secciones. En la primera sección indicaremos el nombre del tipo de enti-

dad. Los atributos se representan como una lista en la segunda sección, y la tercera sección permite representar ciertas restricciones sobre los datos u operaciones, que no utilizaremos en el diseño conceptual de bases de datos.

La tabla 1 muestra un posible ejemplo de un conjunto de entidades de persona.

Tabla 1. Ejemplo de representación de entidades del tipo de entidad 'persona' (*Person*)

ID	name	surname	birthDate	phone-Number	email
33941857B	John	Smith	12/10/1987	936542798	johnsmith@ejemplo.com
77854623Q	Mary	Jane	17/02/1972	696275345	maryjane@ejemplo.com
96725466Z	Fred	Sanchez	07/09/2001	649785467	fredsanchez@ejemplo.com

2.2.1. Representación de los atributos

Los diagramas UML presentan una nomenclatura concreta para especificar las propiedades de cada uno de los atributos:

```
visibilidad nombre [etiquetas] [: tipo] [multiplicidad] [= valor inicial]
```

Donde:

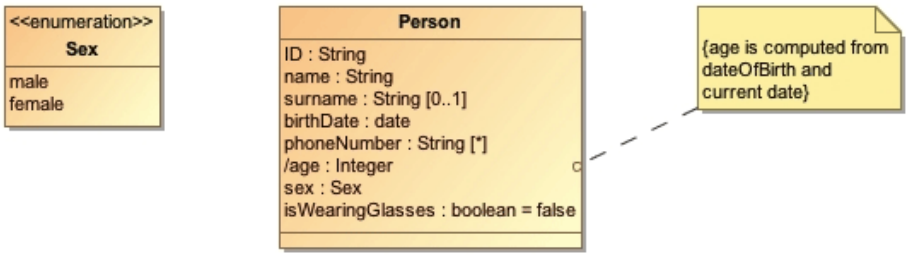
- Visibilidad: visibilidad del atributo (*public*, *protected*, *package* o *private*). Normalmente este concepto no se aplica al diseño conceptual de bases de datos.
- Nombre: denominación del atributo.
- Etiquetas: etiquetas opcionales para indicar ciertos conceptos relacionados con el atributo.
- Tipo: tipo o dominio de los datos.
- Multiplicidad: número de ocurrencias del atributo.
- Valor inicial: valor por defecto.

Como ya se ha visto en algunos ejemplos, se pueden utilizar notas asociadas a los tipos de entidades o atributos para indicar restricciones especiales o cualquier casuística que el diseñador considere oportuno remarcar.

Tipos de entidad

La figura 2 muestra un tipo de entidad en el que podemos ver un atributo opcional (*surname*), un atributo derivado (*age*), un atributo de tipo enumeración (*sex*), un atributo con valor inicial especificado (*isWearingGlasses*) y un atributo multivalor (*phoneNumber*). Una nota asociada al atributo *age* indica la fórmula de cálculo de este atributo derivado.

Figura 2. Ejemplo del tipo de entidad 'persona' (*Person*)



Atributo de tipo enumeración

Un atributo de tipo enumeración presenta un conjunto cerrado de valores que el atributo puede tomar. Por ejemplo, se pueden definir los días de la semana como un atributo de tipo enumeración, donde el conjunto de posibles valores es lunes, martes, miércoles, jueves, viernes, sábado y domingo.

Reflexión

El uso de lenguajes como OCL, por su complejidad, cae fuera del alcance de este texto y, por lo tanto, aquí utilizaremos una nota que indique de manera textual la restricción asociada al atributo en cuestión.

2.2.2. Dominio de los atributos

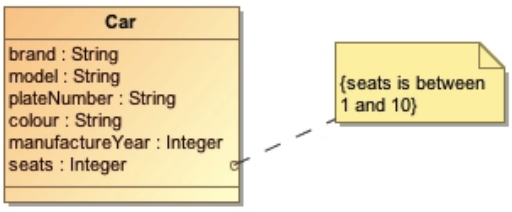
El conjunto de posibles valores legales que puede tomar un atributo se denomina **dominio del atributo**. Este conjunto restringe los posibles valores que puede tomar el atributo en cualquiera de las entidades. Generalmente se especifican a partir de los tipos de datos básicos disponibles en la mayoría de los lenguajes de programación, como entero, real, cadena de texto, booleano o tipo enumerado.

Dominio de los atributos

La figura 3 muestra el tipo de entidad 'coche' (*Car*), en el que se especifica el dominio para cada uno de sus atributos.

En este ejemplo hemos definido un nuevo atributo para indicar el número de plazas de un coche (*seats*), que corresponde al dominio de los enteros y que está limitado por un valor mínimo de 1 plaza y un valor máximo de 10 plazas (para este ejemplo se considera que no hay coches que tengan un número de plazas superior a 10). Para indicar esta restricción sobre el dominio en UML se pueden utilizar notas textuales o recurrir a uno de los lenguajes que permiten definir restricciones sobre los modelos UML, como por ejemplo OCL.

Figura 3. Ejemplo de los tipos de entidad 'coche' (*Car*) con el dominio de los atributos



2.2.3. Atributos compuestos y atributos atómicos

Un **atributo compuesto** es aquel que se puede dividir en atributos más básicos con significado independiente. Un **atributo atómico** o **simple** es aquel que no es divisible sin perder el significado.

Atributo compuesto

Un ejemplo de atributo compuesto es el concepto de 'dirección postal': el atributo de dirección postal con valor "Avenida Santa Cristina, 75, 08280 Barcelona" se puede descomponer en los atributos *calle*, *número*, *código postal* y *ciudad*. De este modo, cada nuevo

atributo mantiene significado independiente y el conjunto de todos juntos identifica una dirección postal. Un ejemplo de atributo simple es el nombre de una población. Aunque este atributo sea compuesto, por ejemplo "Arenys de Mar", no se puede dividir sin perder el significado original.

En lenguaje UML, los atributos compuestos se representan mediante un nuevo tipo de entidad.

2.2.4. Atributos monovalor y atributos multivalor

Un atributo monovalor es aquel que tiene un solo valor para cada entidad. En contraposición a este concepto, un atributo multivalor es aquel que puede tomar más de un valor para la misma entidad.

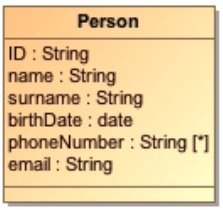
Atributos monovalor y multivalor

Un ejemplo de atributo monovalor es el atributo 'fecha de nacimiento' de una persona, puesto que una persona sólo puede haber nacido en una única fecha.

Un ejemplo de atributo multivalor es el atributo 'teléfono' de una persona, puesto que se puede dar el caso de que una persona tenga ninguno, uno o más de un teléfono (por ejemplo, un teléfono fijo y un teléfono móvil).

La figura 4 muestra el tipo de entidad 'persona' (*Person*) con el atributo 'teléfono' convertido en atributo multivalor.

Figura 4. Ejemplo del tipo de entidad 'persona' (*Person*) con un atributo multivalor



Para indicar la **cardinalidad** de un atributo multivalor utilizaremos:

- Un número para indicar el número exacto de valores que debe tener el atributo.
- El intervalo *min..max* para indicar el número mínimo y máximo de valores que puede tener el atributo.
- El símbolo * para indicar que el atributo puede tener indefinidos valores (cero o más).

Significado del símbolo *

Para indicar que el atributo 'teléfono' (*phoneNumber*) del tipo de entidad 'persona' (*Person*) de la figura 4 puede tener entre 0 y 5 valores hay que indicar "phoneNumber: String[0..5]" en la definición del tipo de entidad.

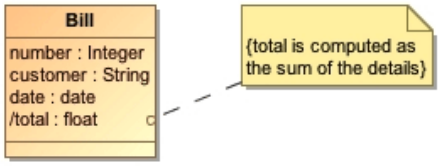
2.2.5. Atributos derivados

Un **atributo derivado** es aquel que se calcula a partir de otros atributos. Por lo tanto, un atributo derivado lleva asociado un procedimiento de cálculo para obtener el valor a partir de otros atributos, que pueden ser derivados o no. Un atributo no derivado es aquel al que hay que asignar un valor.

Atributos derivados

Supongamos la clase 'factura' (*Bill*) con los atributos 'número de factura' (*number*), 'cliente' (*customer*), 'fecha de la factura' (*date*) e 'importe total de la factura' (*total*). Para obtener la información del importe total de la factura, tenemos que crear el atributo 'importe total' (*total*) como atributo derivado, calculado a partir de la suma de cada una de las líneas de los detalles de la factura, tal como se muestra en la figura 5.

Figura 5. Ejemplo de atributo derivado



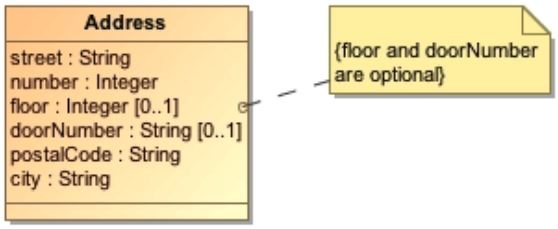
2.2.6. El valor NULL

El valor NULL es un valor especial diseñado para indicar que un atributo es desconocido o no se aplica en una entidad concreta. Para cada atributo, se puede indicar si acepta un valor NULL o no. En caso de que se especifique que el atributo no acepta un valor NULL, todas las entidades deben tener informado dicho atributo con un valor.

Atributo con valor NULL

La figura 6 muestra el tipo de entidad 'dirección' (*Address*), que permite definir la dirección física de una casa, un piso o un local. Hay que señalar que en las casas unifamiliares no tiene sentido el atributo de 'número de piso y puerta'. Los atributos que indican el 'número de piso' (*floor*) y 'puerta de escalera' (*doorNumber*) especifican una multiplicidad de 0 o 1, es decir, que para cada entidad pueden tener un valor asociado o tener valor no definido (NULL).

Figura 6. Ejemplo de atributo con valor NULL



2.3. Claves

Una **clave candidata** es un conjunto de atributos mínimo que permite identificar de manera única todas las entidades de un tipo de entidad.

Un tipo de entidad puede tener una o más claves candidatas. Pero es necesario que tenga al menos una clave candidata que permita identificar de manera unívoca todas las entidades.

Una clave candidata puede estar formada por uno o más atributos. En el caso de estar formada por varios atributos, recibe el nombre de *clave candidata compuesta*. En este caso, la combinación de los diferentes valores de los atributos

debe ser única para identificar de manera unívoca a todas las entidades de un mismo tipo de entidad. Una clave candidata compuesta debe ser mínima, es decir, debe incluir el número mínimo de atributos que permitan identificar de manera unívoca las entidades. La definición de clave candidata se puede ver como una restricción sobre los datos, puesto que prohíbe que dos entidades tengan el mismo valor en los atributos que forman la clave candidata.

El diseñador de la base de datos elige una de las claves candidatas para que sea la clave primaria. Esta es la clave elegida para identificar las entidades de manera unívoca. Como clave candidata, no puede contener valores duplicados pero además, de manera implícita, no puede contener valores NULL en ninguna entidad.

En UML no hay ninguna notación para indicar las claves candidatas ni la clave primaria. En este texto tomamos un convenio para indicar las claves candidatas y las claves primarias. Las claves candidatas se marcan con la etiqueta <U_n>, donde n ($n > 0$) agrupa cada uno de los conjuntos de claves candidatas. Por lo tanto, si dos o más atributos tienen la misma etiqueta, por ejemplo <U_1>, entendemos que se trata de una clave candidata compuesta por todos los atributos con esta etiqueta. La clave primaria se marca con la etiqueta <P> para distinguirla del resto de claves.

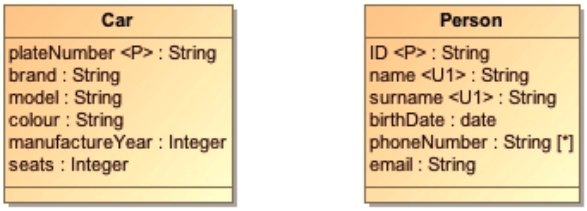
Otros autores prefieren indicar las claves con la etiqueta {U_n} y {P} o utilizar restricciones textuales como método de notación. Aunque el significado es el mismo para las diferentes notaciones, en este texto utilizaremos la notación mencionada inicialmente.

Tipos de entidad con clave primaria y claves candidatas

La figura 7 muestra ejemplos de los tipos de entidad 'coche' y ' persona' con las marcas de la clave primaria y de las claves candidatas, en caso de que existan.

El tipo de entidad 'persona' (*Person*) tiene dos claves candidatas: por un lado tenemos la clave formada por el atributo 'ID' y por el otro tenemos la clave formada por los atributos 'nombre' (*name*) y 'apellidos' (*surname*). Para este ejemplo, suponemos que no pueden existir dos personas con los mismos nombre y apellidos. Aun así, elegimos como clave primaria el atributo 'ID'. El tipo de entidad 'coche' (*Car*) sólo tiene una clave candidata, formada por el atributo 'matrícula' (*plateNumber*), que por lo tanto es elegida para ser clave primaria.

Figura 7. Ejemplos de los tipos de entidad 'coche' (*Car*) y 'persona' (*Person*) con la clave primaria y las claves candidatas marcadas



2.4. Tipos de relaciones

Denominamos **tipos de relaciones** a las asociaciones entre tipos de entidades, y **relaciones**, a las asociaciones entre las entidades.

Los tipos de relaciones indican generalmente las relaciones en las que pueden participar las entidades de un tipo de entidad, y en qué condiciones.

Utilizaremos la nomenclatura siguiente para diferenciar estos dos conceptos:

- El concepto que acabamos de definir como **tipo de relación** también puede ser referenciado por los términos **interrelación**, **relación** o **asociación**.
- El concepto que acabamos de definir como **relación** también puede ser referenciado por los términos **instancia** u **ocurrencia** de la relación, **interrelación** o **asociación**.

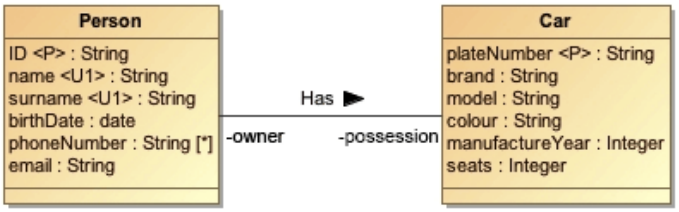
Terminología UML

En la representación en diagramas UML, los tipos de relaciones se denominan *asociaciones* y las relaciones, *instancias de las asociaciones*.

Ejemplo de tipo de relación

Entre los tipos de entidades 'coche' (*Car*) y 'persona' (*Person*) podemos establecer un tipo de relación de propiedad que indique que una persona posee un coche. Para representar esta información debemos crear en el diagrama conceptual de la figura 8 un nuevo tipo de relación denominado *Has*, para indicar que una persona tiene un coche.

Figura 8. Ejemplo de tipo de relación *Has* entre los tipos de entidad 'persona' (*Person*) y 'coche' (*Car*)



Tal como se puede ver en la figura 8, en un tipo de relación pueden aparecer diferentes etiquetas, todas con el fin de facilitar la comprensión de la relación entre las entidades:

- Los tipos de relación casi siempre tienen una **etiqueta de tipo de relación** que indica el significado de la asociación entre los dos tipos de entidades. Esta etiqueta indica la acción entre los dos tipos de entidades, y suele ser un verbo. Para especificar mejor su significado, de manera opcional se puede indicar la dirección del tipo de relación. Esto facilita la lectura de la acción o de la relación que existe entre los dos tipos de entidad. En la figura 8 podemos leer que "una persona tiene un coche". Aunque no es muy ha-

bitual, cuando esta relación es lo bastante evidente se puede considerar que no es necesario indicarlo explícitamente y se puede obviar la etiqueta o la dirección.

- También se puede dar la situación en que sea necesario definir mejor el papel que tiene cada una de las entidades en la relación. En estos casos se puede definir una etiqueta en cada extremo del tipo de relación, que indica el papel que juega cada una de las entidades en la relación y que ayuda a explicar el significado de la relación. Estas etiquetas se denominan **etiquetas de rol** o **nombres de rol**. En términos generales no suelen ser necesarias y no siempre se utilizan, pero en algunos casos pueden ser útiles para definir el papel de las entidades en la relación. La figura 8 muestra los nombres de roles para los dos tipos de entidades que aparecen. Utilizando las etiquetas de rol, podemos leer que "una persona posee (*owns*) un coche" y, de manera inversa, que "un coche es propiedad (*belongs to*) de una persona". Es necesario que todo tipo de relación tenga definidos los roles de los participantes o el nombre del tipo de relación.

En algunos casos los tipos de relaciones, en primera instancia, se modelizan como atributos. Un análisis posterior con más detalle ayuda a ver que lo que se había modelizado en un primer momento como un atributo es un tipo de relación hacia un tipo de entidad nuevo o existente.

Diseño de un nuevo tipo de relación

Supongamos que queremos modelizar el tipo de entidad empleado (*Employee*) para almacenar datos de los empleados de una empresa. Entre otros atributos, nos interesa indicar a qué departamento de la empresa pertenece cada empleado. Una primera aproximación sugiere un diseño en el que se incluye el atributo 'departamento' en el tipo de entidad 'empleado', tal como muestra la figura 9. Pero refinando este modelo, es posible que nos interese tener categorizados los diferentes departamentos, e incluso nos podría interesar almacenar cierta información relativa al departamento. En este nuevo diseño se crea un nuevo tipo de entidad para los departamentos y se establece un tipo de relación entre los empleados y los departamentos (tipos de entidad *Department*) que indica a qué departamento pertenece cada empleado. La figura 10 muestra el nuevo diseño.

Figura 9. Ejemplo de diseño del departamento como un atributo del tipo de entidad 'empleado' (*Employee*)

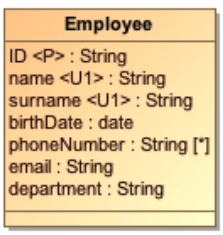


Figura 10. Ejemplo de diseño del departamento como un nuevo tipo de entidad 'departamento' (*Department*) relacionado con el tipo de entidad 'empleado' (*Employee*)

Se define el **grado de un tipo de relación** como el número de tipos de entidades que participan en la asociación.

Los tipos de relaciones se clasifican, según su grado, en:

- **Tipos de relaciones binarias**: asociaciones en las que intervienen dos tipos de entidades.
- **Tipos de relaciones ternarias**: asociaciones en las que intervienen tres tipos de entidades.
- **Tipos de relaciones *n*-arias**: asociaciones en las que intervienen *n* tipos de entidades. A pesar de que los tipos de relaciones binarias y ternarias son un caso concreto de los tipos de relaciones *n*-arias, estos tipos se usan para denotar relaciones con un grado superior a tres.

2.4.1. Tipos de relaciones binarias

Se define un **tipo de relación binaria** como la asociación entre dos tipos de entidades.

Tipos de relación binaria

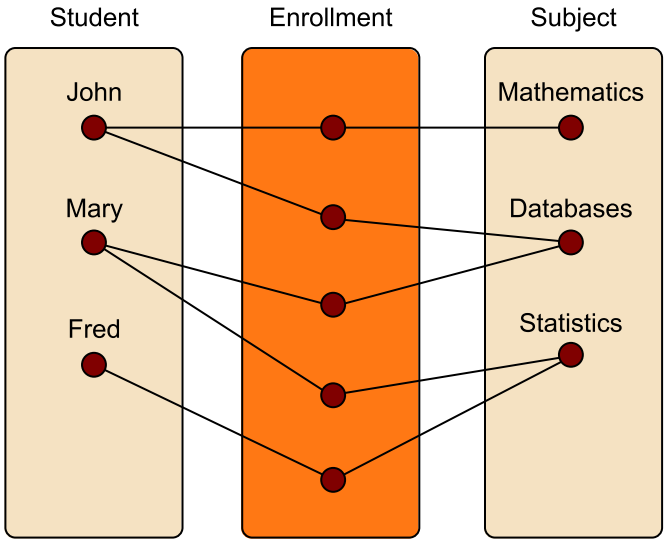
Figura 11. Ejemplo del tipo de relación binaria 'matrícula' (*Enrollment*) entre los tipos de entidad 'estudiante' (*Student*) y 'asignatura' (*Subject*)

La figura 11 muestra el tipo de relación 'matrícula' (*Enrollment*) entre los tipos de entidad 'estudiante' (*Student*) y 'asignatura' (*Subject*). Esta relación indica en qué asignaturas se ha matriculado cada uno de los estudiantes.

Observación

En algunos diagramas de ejemplo de los tipos de relaciones obviaremos los atributos y las características que no sean necesarias para explicar el concepto que nos interese, con objeto de simplificar los diagramas y centrarnos en los tipos de relaciones.

Figura 12. Ejemplo en el nivel de entidades de la relación 'matrícula' (*Enrollment*)



Siguiendo con el ejemplo de la figura 11, la figura 12 muestra la misma relación, pero a escala de entidades. Es decir, este ejemplo muestra una posible asociación entre los datos de las dos entidades mediante la relación *Enrollment*. Se puede ver, por ejemplo, que el estudiante *John* está matriculado en las asignaturas *Mathematics* y *Databases*.

Conectividad

La **conectividad de un tipo de relación** expresa el tipo de correspondencia que hay entre los tipos de entidades que participan en un tipo de relación. En el caso de los tipos de relación binarios, expresa el número de ocurrencias de un tipo de entidad con el que se puede asociar una ocurrencia del otro tipo de entidad según el tipo de relación.

Un tipo de relación binaria puede tener tres tipos de conectividad:

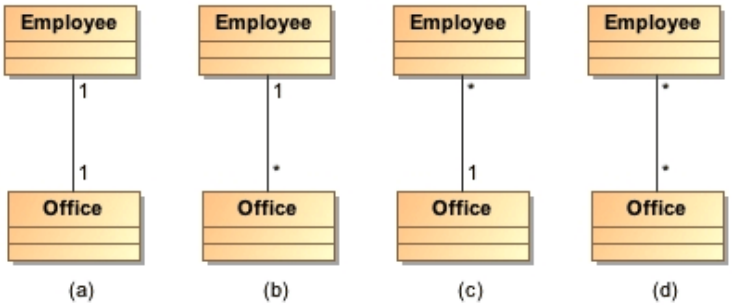
1) Conectividad uno a uno (1:1): indica que cada entidad de un tipo se relaciona solo con una de las entidades del otro tipo. En la figura 13a se puede ver un ejemplo de conectividad 1:1, donde, según el modelo, cada empleado tiene un despacho y en cada despacho hay un solo empleado. La conectividad 1:1 se denota poniendo un 1 en cada parte del tipo de relación.

2) Conectividad uno a muchos (1:*N* o 1..*): indica que cada entidad de un tipo se relaciona con varias entidades del otro tipo, pero las entidades de este segundo tipo solo se relacionan con una única entidad del primer tipo. En la figura 13b se puede ver un ejemplo de conectividad 1:*N*, donde la *N* está en la parte del tipo de entidad 'despacho'. Según el modelo, cada empleado tiene varios despachos, pero en cada despacho hay un solo empleado. En la figura 13c se puede ver la relación inversa. En este caso la *N* está en la parte del tipo de entidad 'empleado'. Según el modelo, cada empleado tiene un único

despacho, pero un mismo despacho lo pueden compartir varios empleados. La conectividad 1:*N* se denota poniendo un 1 en una parte del tipo de relación y una *N* o * en la otra parte.

3) Conectividad muchos a muchos (*M*:*N* o *..*): indica que cada entidad de un tipo de entidad se relaciona con varias entidades del otro tipo. En la figura 13d se puede ver un ejemplo de conectividad *M*:*N*, donde, según el modelo, cada empleado tiene varios despachos y en cada despacho hay varios empleados. La conectividad *M*:*N* se denota poniendo una M o * en una parte del tipo de relación y una *N* o * en la otra parte.

Figura 13. Ejemplos de conectividad en los tipos de relaciones binarias entre los tipos de entidad 'empleado' (*Employee*) y 'despacho' (*Office*)



Estas restricciones las determinan los requisitos de cada problema y son los usuarios de la base de datos quienes han de conocer las restricciones implícitas en los problemas.

Cuando se habla de "conectividad a muchos" se puede indicar de diferentes maneras. Según el matiz del problema, usaremos:

- Un número para indicar el número exacto de entidades que pueden intervenir en la relación.

- El intervalo *min..max* para indicar el número mínimo y máximo de entidades que pueden intervenir en la relación.

- Lo etiqueta *N* o * para indicar que un número indefinido (cero o más) de entidades pueden participar en la relación.

Conectividad "a muchos"

La figura 14 muestra el tipo de relación 'matrícula' (*Enrollment*) entre el tipo de entidad 'estudiante' (*Student*) y 'asignatura' (*Subject*), donde se indica que un estudiante tiene que estar matriculado en un mínimo de dos asignaturas y un máximo de cinco.

Figura 14. Ejemplo del tipo de relación 'matrícula' (*Enrollment*) donde se indica el número mínimo y máximo de instancias que pueden intervenir en la relación

Dependencia de existencia o restricciones de participación

En algunos casos, las entidades de un tipo de entidad no pueden existir si no están relacionadas con las entidades de otro tipo de entidad mediante una relación binaria determinada. En estos casos, se dice que el segundo tipo de entidad es un tipo de entidad obligatorio en la relación. En otro caso, se dice que es un tipo de entidad opcional en la relación.

Tipos de entidad obligatoria y opcional

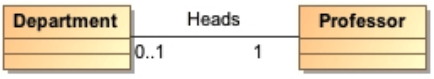
Tipo de entidad obligatoria: para designar este concepto también se puede decir que el tipo de entidad tiene una participación total en la asociación o que presenta una dependencia de existencia respecto a la asociación.

Tipo de entidad opcional: para designar este concepto también se puede decir que el tipo de entidad tiene una participación parcial en la asociación.

Dependencia de existencia en los tipos de relaciones binarias

La figura 15 muestra un modelo en el que aparece un tipo de relación binaria que expresa la relación de 'dirección de departamento' (*Heads*) entre los tipos de entidad 'profesor' (*Professor*) y 'departamento' (*Department*). El tipo de entidad 'profesor' es obligatorio en la relación 'dirección de departamento'. Esto indica que no puede haber un departamento que no tenga un profesor que haga de director de departamento. El tipo de entidad 'departamento', en cambio, es opcional en la relación 'dirección de departamento'. Puede ser que haya uno o más profesores que no tengan la responsabilidad de dirigir un departamento.

Figura 15. Ejemplo de dependencia de existencia en los tipos de relaciones binarias



Esta dependencia de existencia sólo es aplicable en los tipos de relaciones binarias. No se da en los tipos de relaciones *n*-arias.

En los diagramas UML se utiliza la cardinalidad del tipo de relación para expresar la obligatoriedad o no de cada uno de los tipos de entidades que participan en la asociación. Considerando los tipos de conectividad que hemos visto, hay que aplicar:

- **Tipo de entidad con cardinalidad 1**: un tipo de entidad conectada con cardinalidad 1 a una asociación expresa la obligatoriedad indicando una cardinalidad exactamente igual a 1..1, mientras que si es opcional en la asociación, se indica con la cardinalidad 0..1.

- **Tipo de entidad con cardinalidad *N***: un tipo de entidad conectada con cardinalidad *N* a una asociación expresa la obligatoriedad indicando una cardinalidad 1..*, mientras que si es opcional en la asociación, se indica con la cardinalidad 0..*.

Métodos abreviados de cardinalidades:

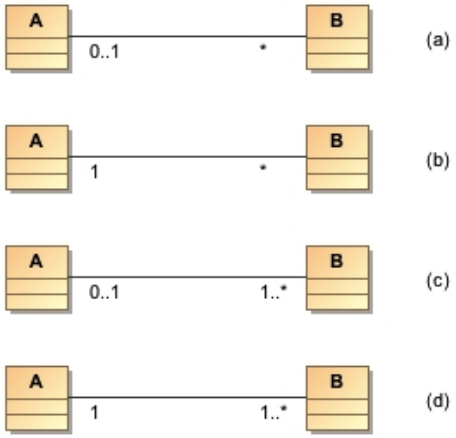
- La cardinalidad 1 es una manera abreviada de referirse a la cardinalidad 1..1.

- La cardinalidad * es una manera abreviada de referirse a la cardinalidad 0..*.

Obligatoriedad en la dependencia de existencia

La figura 16 muestra las cuatro posibilidades de expresar la obligatoriedad de las clases A y B en un tipo de relación binaria 1:*N*. En la figura 16 vemos en 16a el caso en el que los dos tipos de entidades son opcionales, en 16b el caso en el que el tipo de entidad A es obligatorio, en 16c el caso en el que el tipo de entidad B es obligatorio y en 16d el caso en el que los dos tipos de entidad son obligatorios.

Figura 16. Ejemplos de obligatoriedad en la dependencia de existencia



Igual que las restricciones de conectividad, estas restricciones son inherentes a los problemas que se quieren resolver y a partir de la compilación de requisitos de los usuarios de la base de datos se podrá determinar cómo se debe proceder en cada caso.

2.4.2. Tipo de relaciones ternarias

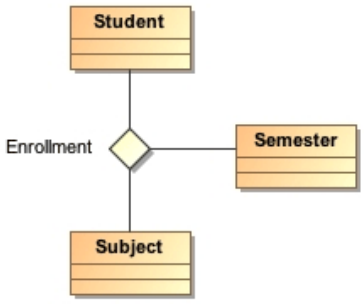
Se define un **tipo de relación ternaria** como la asociación entre tres tipos de entidades.

Hay situaciones en las que no basta modelizar una asociación entre dos tipos de entidades para representar un concepto y es necesario modelizar la asociación entre tres tipos de entidades para representar el concepto deseado.

Tipos de relación ternaria

Siguiendo un ejemplo visto antes, no es suficiente modelizar el tipo de relación 'matrícula' (*Enrollment*) entre los tipos de entidades 'estudiante' (*Student*) y 'asignatura' (*Subject*) para representar la realidad de los estudiantes que suspenden una asignatura y tienen que volver a cursarla en un semestre posterior. En estos casos, una entidad 'estudiante' puede tener cero, una o más de una matrícula para una misma entidad 'asignatura'. El requisito es que sólo es posible efectuar una matrícula por semestre. Por lo tanto, hay que introducir el tipo de entidad 'semestre' (*Semester*) en el ejemplo anterior y modificar el modelo para que la relación 'matrícula' asocie los tres tipos de entidad. La figura 17 muestra el nuevo modelo.

Figura 17. Ejemplo de tipo de relación ternaria entre los tipos de entidad 'estudiante' (*Student*), 'asignatura' (*Subject*) y 'semestre' (*Semester*)



Conectividad

En la conectividad de un tipo de relación ternaria hay implicados tres tipos de entidades y cada uno puede tomar la conectividad "a uno" o "a muchos".

Un tipo de relación ternaria puede tener cuatro tipos de conectividad:

- Conectividad *M*:*N*:*P* o *..*..*.
- Conectividad *M*:*N*:1 o *..*..1.
- Conectividad *M*:1:1 o *..1..1.
- Conectividad 1:1:1 o 1..1..1.

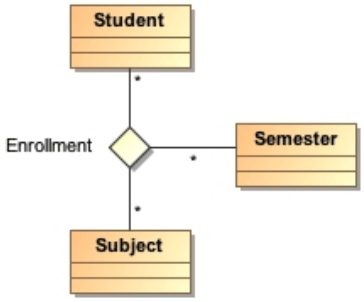
Para decidir cómo se debe conectar cada uno de los tipos de entidad a la asociación, hay que fijar "a una" las otras entidades y entonces preguntarse si es posible conectar con la relación "a una" o "a muchas" entidades del primer tipo de entidad.

Conectividad en un tipo de relación ternaria

Veamos cómo definimos la conectividad en un tipo de relación ternaria mediante el ejemplo que ilustra la figura 18.

En la figura podemos ver los tres tipos de entidad, 'estudiante' (*Student*), 'asignatura' (*Subject*) y 'semestre' (*Semester*), y el tipo de relación 'matrícula' (*Enrollment*).

Figura 18. Ejemplo de conectividad en un tipo de relación ternaria



Para determinar el grado de conectividad de cada tipo de entidad en la asociación, debemos fijar "a una" las instancias del resto de los tipos de entidades y nos tenemos que preguntar si es posible conectar con la relación "a una" o "a muchas" instancias del primer tipo de entidad.

En primer lugar, por ejemplo, nos preguntamos si con referencia a la relación 'matrícula' y dados una asignatura y un semestre concretos se pueden tener "uno" o "muchos" estudiantes. La respuesta es que puede haber "muchos" estudiantes matriculados, pues-

to que varios estudiantes pueden matricularse de una misma asignatura en el mismo semestre. Por lo tanto, el tipo de entidad 'estudiante' participa con grado *N* en la relación 'matrícula'.

En segundo lugar, nos preguntamos, por ejemplo, si fijados un estudiante y una asignatura concretos puede existir matrícula en "uno" o "muchos" semestres. La respuesta es que pueden tener "muchos" semestres, puesto que un estudiante se puede matricular más de una vez en diferentes semestres hasta superarlos. Por lo tanto, el tipo de entidad 'semestre' participa con grado *N* en la relación 'matrícula'.

Finalmente, nos preguntamos si fijados un estudiante y un semestre concretos pueden tener matrícula de "una" o "muchas" asignaturas. La respuesta es que se pueden tener "muchas" asignaturas matriculadas, puesto que un estudiante se puede matricular de varias asignaturas en un mismo semestre. Por lo tanto, el tipo de entidad 'asignatura' también participa con grado *N* en la relación matrícula.

Así pues, vemos que el tipo de relación 'matrícula' tiene cardinalidad *M*:*N*:*P* o *..*..*.

2.4.3. Tipos de relaciones *n*-arias

Se define un **tipo de relación *n*-aria** como la asociación entre tres o más tipos de entidades.

El tipo de relación ternaria es un caso concreto de tipo de relación *n*-aria. Lo que hemos visto en el subapartado anterior se puede generalizar para los tipos de relaciones *n*-arias. Por lo tanto, en una relación *n*-aria puede haber $n + 1$ tipos de conectividad, puesto que cada una de las *n* entidades puede estar conectada "a uno" o "a muchos" tipos en la asociación.

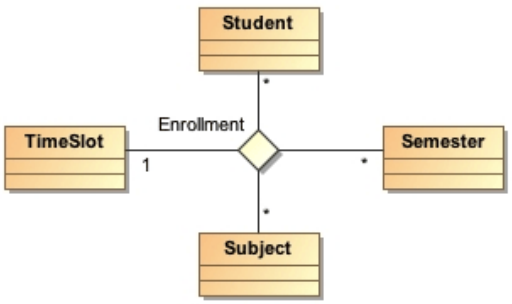
Conectividad en un tipo de relación cuaternaria

La figura 19 presenta una ampliación del ejemplo anterior, en el que se presentaban los tipos de entidad 'estudiante' (*Student*), 'asignatura' (*Subject*) y 'semestre' (*Semester*) asociados mediante el tipo de relación 'matrícula' (*Enrollment*).

En este caso queremos añadir el concepto de *turno* o *franja horaria* en el esquema para indicar que una misma asignatura puede ser de turno de mañana o de tarde. Para modelizar este concepto, hemos creado el tipo de entidad 'franja horaria' (*TimeSlot*).

Para determinar el grado de conectividad del nuevo tipo de entidad seguiremos el mismo proceso visto para el caso del tipo de relaciones ternarias. En este caso, nos preguntamos si con referencia al tipo de relación 'matrícula' y fijados un estudiante, un semestre y una asignatura concretos puede haber "una" o "muchas" franjas horarias. La respuesta es que solo puede haber "una" franja horaria. Por lo tanto, el tipo de entidad de 'franja horaria' o 'turno' (*TimeSlot*) participa con grado 1 en este tipo de relación.

Figura 19. Ejemplo de conectividad en un tipo de relación cuaternaria



Cardinalidades mínimas en los tipos de relaciones n-arias

Normalmente, la cardinalidad mínima con la que las entidades participan en una relación n-aria es 0. La razón es que, dadas las instancias concretas de todas las $n - 1$ entidades que participan en la relación, podría ser que aquella relación concreta nunca se haya dado. Vemos esto con el ejemplo de la figura 19. Dados un estudiante, una asignatura y un semestre concretos, es perfectamente posible que ese estudiante haya estudiado esa asignatura pero en un semestre diferente, o que nunca la haya estudiado, o incluso que la asignatura ni siquiera se ofrezca durante el semestre en cuestión. Así, la cardinalidad mínima de *TimeSlot* sería 0. En el caso de que el estudiante sí hubiera estudiado la asignatura ese semestre, lo habría hecho en una franja horaria concreta y, por lo tanto, la cardinalidad máxima sería 1. Es decir, nos quedaría una cardinalidad 0..1 del lado *TimeSlot*. De modo similar, dados un estudiante concreto y una asignatura concreta en una franja horaria determinada, podría ser que esa combinación nunca se hubiese dado o que se hubiera dado muchas veces: tendremos una cardinalidad 0..* (o, de manera abreviada, *). En el ejemplo de la figura 19 tenemos una cardinalidad 1 del lado de *TimeSlot*, esto implica que, en nuestro modelo, todos los estudiantes han estudiado todas las asignaturas existentes y, además, que las han estudiado cada semestre que contemplamos. Para cada una de estas combinaciones posibles, lo han hecho en una franja horaria específica.

2.4.4. Tipos de relaciones reflexivas o recursivas

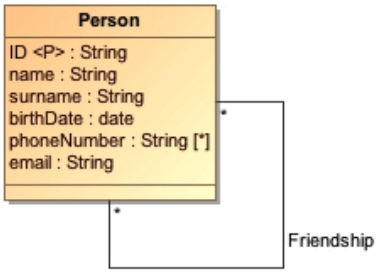
Un **tipo de relación reflexiva** o **recursiva** es un tipo de relación binaria en la que un tipo de entidad se relaciona consigo misma.

Se presenta una relación entre dos entidades del mismo tipo de entidad. Igual que en el resto de los tipos de relaciones binarias, pueden tener conectividad 1:1, 1:*N* o *M*:*N* y pueden expresar dependencia de existencia.

Tipos de relación recursiva binaria

Un claro ejemplo de un tipo de relación recursiva binaria es el concepto de *amistad* entre dos personas. En este caso, tenemos el tipo de entidad 'persona' (*Person*) y el tipo de relación 'amistad' (*Friendship*), que une dos entidades de 'persona'. La figura 20 muestra su modelo conceptual. La conectividad de la asociación es *M*:*N*, puesto que una persona puede tener varios amigos y, a la vez, puede haber varias personas que la tengan de amiga.

Figura 20. Ejemplo de tipo de relación recursiva



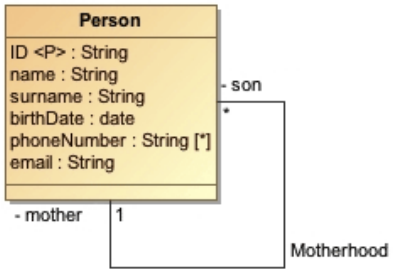
Generalmente no se hace referencia explícita al rol de cada entidad en una relación, puesto que normalmente cada entidad participa una única vez en una relación y su rol se sobreentiende. Pero en las relaciones recursivas puede ser que el rol de cada entidad no sea tan claro. Por lo tanto, si se considera oportuno, se puede indicar el rol de cada asociación entre la entidad y la relación.

Otro ejemplo de tipo de relación recursiva

La figura 21 muestra un tipo de relación recursiva que describe la relación de maternidad entre las personas. Podemos ver que una persona puede tener cero o más hijos, y que

una persona solo puede tener una madre. Las etiquetas o nombres de rol, en este caso, pueden ayudar a comprender el rol de cada entidad en esta relación recursiva.

Figura 21. Ejemplo de tipo de relación recursiva con nombres de rol



2.5. Tipos de entidades asociativas

Las relaciones también pueden tener atributos que permitan describir propiedades interesantes. Estos atributos se definen y siguen las mismas reglas que en el caso de los atributos de las entidades.

Un **tipo de entidad asociativa** es el resultado de considerar una relación entre entidades como si fuera una entidad. Su nombre se corresponde con el nombre de la relación sobre la que se define.

Los tipos de entidades asociativas añaden nuevas funcionalidades a las relaciones. Algunas de las principales son las siguientes:

1) Permiten definir atributos que hacen referencia a la relación. Es decir, son específicos de la asociación que hay entre las dos instancias de los dos tipos de entidad que forman la relación.

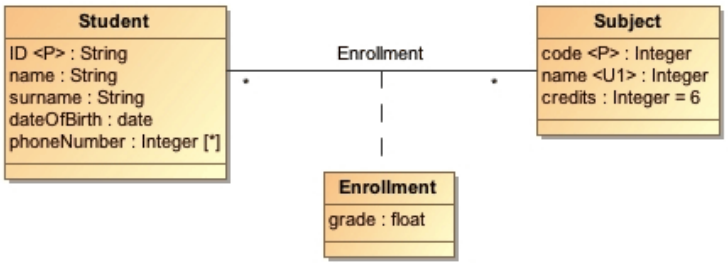
2) Permiten que el tipo de entidad asociativa, y por lo tanto la relación entre los tipos de entidades, participe en otras relaciones con otros tipos de entidades o con otros tipos de entidades asociativas.

Tipos de entidad asociativa

La figura 22 muestra el tipo de relación 'matrícula' (*Enrollment*) entre los tipos de entidad 'estudiante' (*Student*) y 'asignatura' (*Subject*). Esta relación indica en qué asignaturas están matriculados cada uno de los estudiantes. La entidad asociativa contiene el atributo 'nota' (*Grade*), que indica la nota que ha obtenido un estudiante en una asignatura. Para cada relación entre un estudiante y una asignatura hay una relación 'matrícula' con un atributo 'nota'.

En el caso de situar el atributo 'nota' (*Grade*) en la entidad 'estudiante' (*Student*), decimos que un estudiante siempre tendrá la misma nota de todas las asignaturas de las que se matricule. En el caso de situar el atributo en la entidad 'asignatura' (*Subject*), en cambio, decimos que todos los estudiantes tienen la misma nota en una misma asignatura. Por lo tanto, la única manera de modelizar la realidad de manera correcta es considerar el atributo 'nota' como un atributo de la relación entre estos dos tipos de entidad.

Figura 22. Ejemplo del tipo de entidad asociativa 'matrícula' (*Enrollment*)

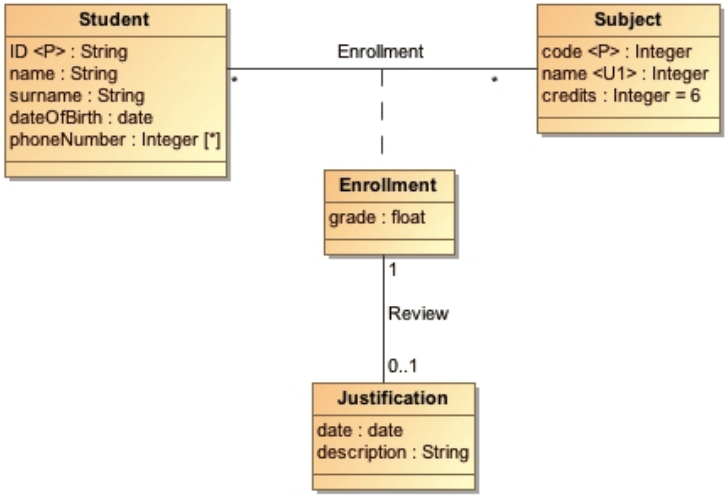


Los tipos de entidades asociativas permiten establecer relaciones con otras entidades y también con otras entidades asociativas, es decir, permiten crear relaciones entre relaciones.

Tipos de relación entre una entidad y una entidad asociativa

Supongamos que queremos añadir en el ejemplo anterior el concepto de *justificación* o *retorno personalizado* de la nota. Es decir, queremos expresar que en algunos casos, por ejemplo cuando lo solicite, el estudiante recibirá una descripción en la que se justifica la puntuación que ha obtenido y los errores que ha cometido en una asignatura determinada. La figura 23 muestra el modelo conceptual con el nuevo requisito. Podemos ver cómo el nuevo tipo de entidad 'justificación' (*Justification*) indica la fecha y la descripción de la justificación y se relaciona con algunas matriculaciones. Este punto es relevante, puesto que, dada la cardinalidad del tipo de relación 'revisión' (*Review*), se puede deducir que pueden existir parejas estudiante-asignatura sin una justificación.

Figura 23. Ejemplo de tipo de relación entre una entidad y una entidad asociativa



Cabe señalar que esta misma situación no se puede modelizar mediante un tipo de relación ternaria.

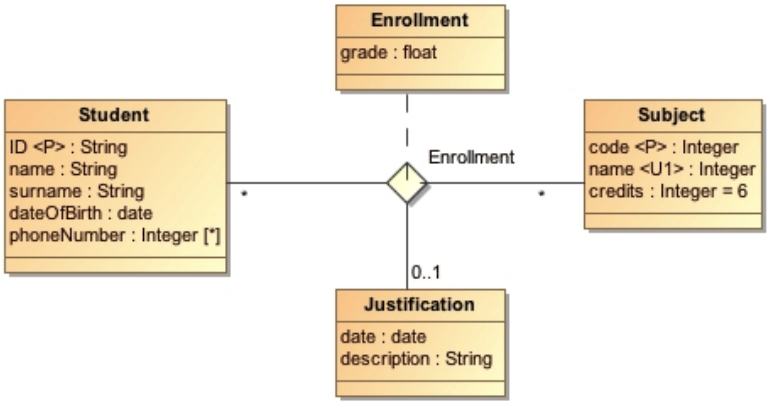
Error de expresión

La figura 24 muestra el modelo conceptual y emplea un tipo de relación ternaria. En este caso el significado del modelo es diferente del que planteábamos en el ejemplo anterior. Este modelo nos dice que para cada estudiante (*Student*) y asignatura (*Subject*) puede haber ninguna o una justificación (*Justification*). Pero también indica que fijadas una justificación y una asignatura puede haber muchos estudiantes, afirmación que es falsa según los requisitos iniciales. Igualmente, el modelo indica que fijados un estudiante y una justificación puede haber muchas asignaturas, afirmación que también es falsa. Si modificamos las cardinalidades de los tipos de entidad 'estudiante' y 'asignatura' y las establecemos a 1, entonces no podremos representar de manera correcta que un estudiante puede estar matriculado en varias asignaturas.

El problema reside en el hecho de que intentamos representar con un tipo de relación ternaria un modelo que es binario. Hay que señalar que el concepto de *justificación* hace

referencia a la relación que existe entre las entidades 'estudiante' y 'asignatura', y para ello es necesario que sea modelizado como un tipo de entidad que se relaciona con el tipo de entidad asociativa que surge del tipo de relación entre los dos tipos de entidades.

Figura 24. Ejemplo de error de expresión sobre el ejemplo de la figura 23



Es importante que quede claro que no se representa el mismo concepto con un tipo de relación ternaria y un tipo de relación binaria con una entidad asociativa.

2.6. Tipos de entidades débiles

Se define un **tipo de entidad fuerte** como un tipo de entidad que posee un conjunto de atributos suficientes para formar claves candidatas e identificar de manera inequívoca todas sus instancias.

Tipos de entidades fuertes

También se pueden denominar *tipos de entidades propietarias*, *dominantes* o *padres*.

La existencia de las instancias de un tipo de entidad fuerte no depende de ningún otro tipo de entidad o tipo de relación. Es decir, el significado de las entidades fuertes no necesita ninguna relación con otras entidades que complemente su significado. Por lo tanto, se pueden ver estas entidades como aquellas que tienen existencia propia y muy definida.

Se define un **tipo de entidad débil** como un tipo de entidad cuyos atributos no la identifican completamente, sino que sólo la identifican de manera parcial. Estas entidades deben participar en una relación con otras entidades que ayuden a identificarlas de manera unívoca.

Tipos de entidades débiles

También se pueden denominar *tipos de entidades subordinadas* o *hijas*.

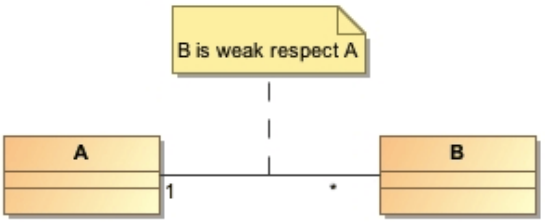
Un **tipo de entidad débil**, por lo tanto, necesita una asociación con un tipo de entidad que le permite identificar de manera única sus instancias. Este tipo de relación recibe el nombre de *tipo de relación identificativa* del tipo de entidad débil. El tipo de entidad débil debe tener una restricción de participación total (dependencia de existencia) respecto al tipo de relación identificativa, puesto que si no es así, hay instancias que no se podrán identificar de manera única.

Toda entidad débil debe formar parte de una relación binaria con conectividad 1:*N*, donde la entidad débil debe estar en el lado *N*. Esta relación le permitirá identificarse completamente.

Las entidades débiles no pueden sobrevivir si se elimina la entidad fuerte con la que están relacionadas mediante la relación identificativa. Por lo tanto, hay que tener en cuenta que, si se elimina una entidad fuerte, hay que eliminar las entidades débiles que están relacionadas con ella.

En los diagramas UML un tipo de entidad débil se indica mediante la dependencia de su existencia hacia el tipo de entidad fuerte. El tipo de entidad débil, debido a la dependencia de existencia o restricción de participación, necesita mantener una relación 1:*N* con el tipo de entidad fuerte. Por lo tanto, siempre mantendrá una relación con una cardinalidad igual a 1 hacia la parte del tipo de entidad fuerte. La figura 25 muestra la notación para indicar un tipo de entidad débil en los diagramas UML.

Figura 25. Notación UML para indicar un tipo de entidad débil (*B*)



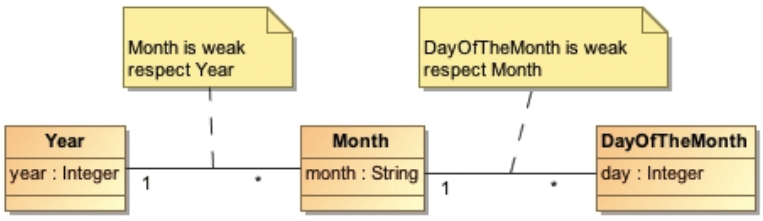
Hay que destacar que un tipo de relación identificativa no suele tener atributos, puesto que en las relaciones 1:*N* siempre es posible poner los atributos en el tipo de entidad débil, es decir, en la parte con cardinalidad *N*.

Un tipo de entidad puede ser fuerte en una determinada asociación y, al mismo tiempo, ser débil en otra asociación.

Relatividad del tipo de relación

La figura 26 muestra el tipo de entidad 'mes del año' (*Month*), que es débil respecto a la relación con el tipo de entidad 'año' (*Year*) y fuerte respecto a la relación con el tipo de entidad 'día del mes' (*DayOfTheMonth*).

Figura 26. Ejemplo de tipo de entidad fuerte respecto a una relación y, al mismo tiempo, débil respecto a otra relación



Una **clave parcial** en un tipo de entidad débil es el conjunto de atributos que permite identificar de manera única todas las instancias del tipo de entidad débil que están relacionadas con la misma instancia del tipo de entidad fuerte.

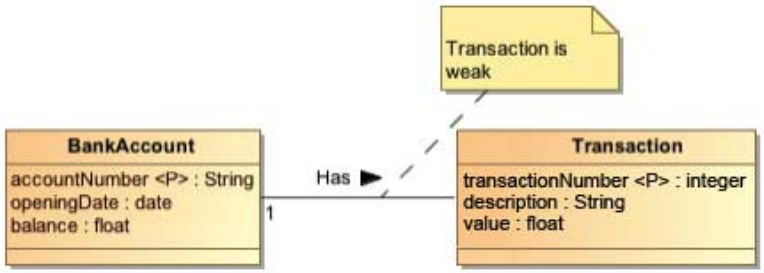
En el peor de los casos, la clave parcial es el conjunto de todos los atributos del tipo de entidad débil.

Se considera que la clave primaria del tipo de entidad débil está formada por la clave primaria del tipo de entidad fuerte y la clave parcial del tipo de entidad débil. De este modo se puede referenciar cada una de las instancias del tipo de entidad débil de manera única.

Notación única de dos tipos de entidades débiles

Si consideramos el concepto *cuenta corriente* en una entidad bancaria, podemos afirmar que es un tipo de entidad fuerte, puesto que posee el conjunto de atributos necesarios para identificar de manera única cada una de sus instancias. En cambio, si consideramos el concepto de *movimiento* o *transacción*, nos damos cuenta de que sólo tiene sentido y puede existir si está asociado a una cuenta bancaria. Por lo tanto, para modelizar esta situación se puede crear un tipo de entidad fuerte, *BankAccount*, y un tipo de entidad débil, *Transaction*, como se muestra en la figura 27. La clave primaria del tipo de entidad *BankAccount* es el atributo *accountNumber*. En el tipo de entidad *Transaction* la clave parcial es el atributo *transactionNumber*, que indica el número de transacción. Por lo tanto, la clave primaria del tipo de entidad débil es compuesta y está formada por los atributos {*accountNumber*, *transactionNumber*}.

Figura 27. Ejemplo de tipo de entidad débil (*Transaction*) y la relación identificativa con el tipo de entidad fuerte (*BankAccount*)



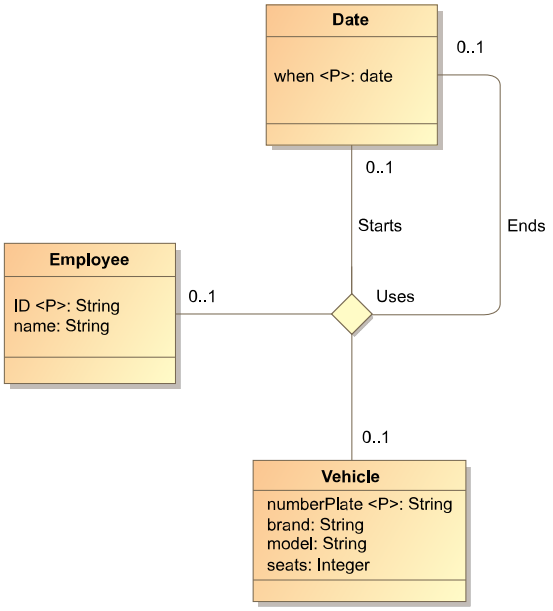
2.7. Modelización de intervalos de tiempo

Cuando tenemos un conjunto de requerimientos para diseñar un modelo conceptual, es habitual que sea posible representar estos requerimientos de diferentes maneras y que los modelos resultantes sean equivalentes. En el caso particular de los históricos, en los que interesa saber durante qué periodos de tiempo se da la relación entre dos o más entidades, el concepto de "periodo de tiempo" se puede modelizar de diversos modos. Normalmente, intervienen dos (o más) entidades, con el requerimiento de guardar una serie de fechas que pueden ser, por ejemplo, de inicio y fin. A continuación, ponemos por ejemplo que queremos representar en qué intervalos de tiempo son utilizados los vehículos de una empresa por sus empleados y veremos cómo quedaría el modelo con diferentes opciones de modelización.

Opción 1: con una relación

Con esta solución tendríamos una relación en la que participarían todas las entidades sobre las que queremos guardar el intervalo de tiempo, y también la/las entidad/es que modelizan este intervalo.

Figura 28. Modelización de intervalo de tiempo con una relación

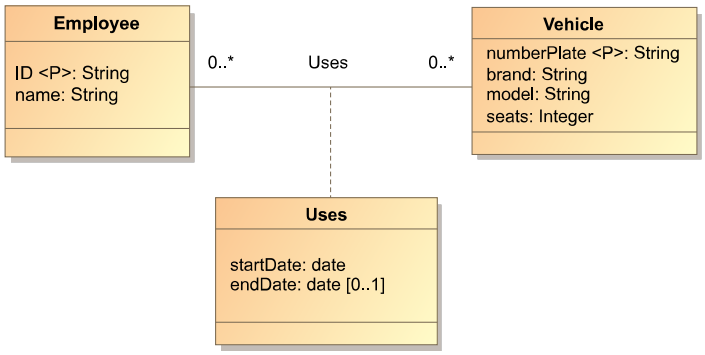


Vemos que en la relación n-aria participan las entidades *Employee* y *Vehicle* para representar los datos de los empleados y los vehículos de la empresa, respectivamente. También participa la entidad *Date* para dar relevancia a la temporalidad. En este caso, la entidad *Date* interviene por partida doble: una vez para representar la fecha de inicio, cuando el empleado comienza a utilizar el vehículo, y otra para representar la fecha en la que deja de usarlo.

Opción 2: con una relación más una entidad asociativa

Esta variante pasa por tener una relación en la que participan todas las entidades sobre las que queremos guardar el intervalo de tiempo. De esta relación dependería una entidad asociativa que nos serviría para modelizar el intervalo. Vemos cómo queda el modelo con esta variante, siguiendo el ejemplo anterior.

Figura 29. Modelización de intervalo de tiempo con relación más entidad asociativa

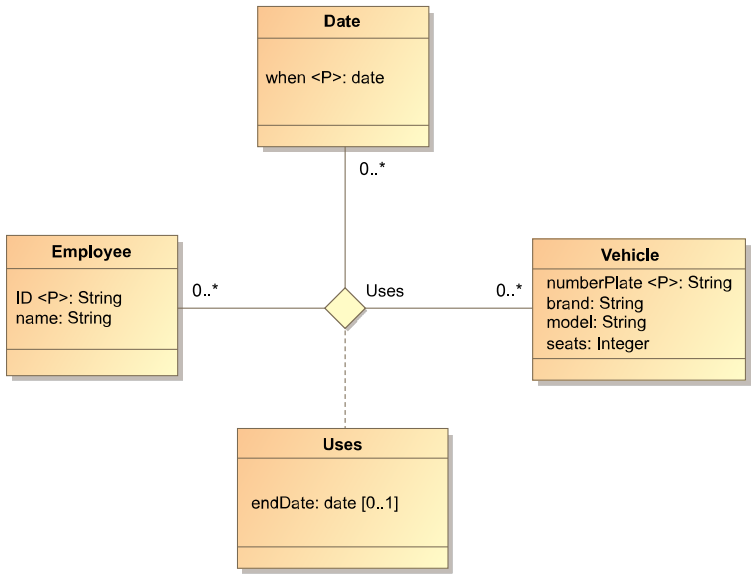


En este modelo tenemos una relación binaria para relacionar los empleados con los vehículos que utilizan. De esta relación binaria depende una entidad asociativa para modelizar la fecha de inicio y la fecha de finalización. La fecha de finalización es opcional, ya que puede ser que algún empleado aún no haya devuelto algún vehículo. Fijémonos, sin embargo, en que esta opción de modelización tiene una limitación: una entidad asociativa, como hemos visto en el apartado 2.5, sirve para modelar datos relacionados a una instancia concreta de la relación de la que depende, en el caso de nuestro ejemplo para cada combinación empleado/vehículo. Esto implica que, si un mismo empleado utiliza un mismo vehículo más de una vez, o bien no podemos guardar los datos históricos de estas nuevas ocurrencias, o bien tenemos que sobrescribirlos. En determinados contextos esto no sería aceptable y, por lo tanto, esta opción queda limitada a situaciones en las que la pérdida de datos no sea un problema.

Opción 3: combinación de ambas opciones

Incluso podemos elegir esta tercera alternativa, con una relación en la que participaría una de las fechas del intervalo (por ejemplo, la fecha de inicio). La otra fecha del intervalo (por ejemplo, la fecha de finalización) se incorporaría como una entidad asociativa dependiendo de la relación.

Figura 30. Modelización de intervalo de tiempo mixta



Esta opción relaja la limitación que veíamos en el ejemplo anterior: en este caso sí que podríamos guardar todas las fechas de inicio y fin en el caso de que el mismo empleado utilizara el mismo vehículo más de una vez, con la única limitación de una vez para cada fecha de inicio.

2.8. Opciones de diseño

En algunas situaciones no es trivial determinar si un concepto debe ser modelizado como un tipo de entidad, un atributo o un tipo de relación. Hay que tener en cuenta que normalmente no existe un único diagrama conceptual para un problema concreto, y que puede haber diferentes conceptualizaciones que den lugar a diferentes diagramas, todos correctos.

A continuación, mencionamos algunos consejos que pueden ser útiles durante el diseño conceptual de una base de datos:

- El proceso de diseño conceptual de una base de datos se puede enfocar como un proceso iterativo de refinamiento, donde se crea un primer diseño inicial y se va refinando de manera iterativa hasta conseguir el nivel de diseño deseado.

- Es frecuente modelizar un concepto como un atributo en un primer momento, y en refinamientos posteriores convertirlo en un tipo de relación. Modelizar el atributo como una relación a un tipo de entidad nuevo o existente permite categorizar los valores de este concepto y, además, permite que se puedan añadir nuevos atributos relacionados con el tipo de entidad nuevo o existente, ahora o en el futuro.

De atributo a tipo de entidad

Si los tipos de entidades *Estudiante*, *Profesor* y *Curso* tienen el atributo *Departamento* es interesante crear el nuevo tipo de entidad *Departamento*, con un único atributo que indica el nombre del departamento, y crear tres tipos de relaciones binarias entre *Estudiante*, *Profesor* y *Curso*, y el nuevo tipo de entidad *Departamento*.

- También es habitual que un atributo existente en varios tipos de entidades sea convertido en un nuevo tipo de entidad. A continuación hay que establecer relaciones desde los tipos de entidades que contenían este atributo hacia el nuevo tipo de entidad.

- También puede existir el refinamiento inverso. En este caso, se puede eliminar un tipo de entidad y representar el concepto como un atributo si solo afecta a un tipo de entidad (o a muy pocos).

2.9. Criterios de asignación de nombres

En el diseño del esquema conceptual de una base de datos es importante elegir adecuadamente los nombres de los tipos de entidades, atributos y tipos de relaciones que aparecen en él, puesto que no siempre es fácil determinar el papel de cada uno de ellos en un modelo conceptual. Hay que escoger nombres que transmitan de la mejor manera posible el significado de las diferentes estructuras del esquema.

Los criterios empleados en este texto son una opción, aunque no hay normativa y pueden variar respecto a otros autores.

Ejemplos de grafías posibles

Grafía Pascal: la primera letra del identificador y la primera letra de las siguientes palabras se escriben en mayúscula. El resto, en minúscula. Por ejemplo: BackColor.

Grafía Camel: la primera letra del identificador se escribe en minúscula y la primera letra de las siguientes palabras concatenadas, en mayúscula. El resto, en minúscula. Por ejemplo: backColor.

En los puntos siguientes establecemos la base de la nomenclatura en los diagramas UML:

- Etiquetar los tipos de relaciones utilizando nombres simples en singular, siempre que sea posible. Si es necesario, se pueden utilizar hasta dos o tres nombres. Hay que emplear la grafía Pascal para escribir el nombre de los tipos de entidades.

- Etiquetar los atributos utilizando nombres simples en singular y evitando que aparezca el nombre del tipo de entidad. Si es necesario, se pueden utilizar hasta dos o tres nombres o la combinación de un nombre y uno o dos adjetivos. Hay que emplear la grafía Camel para escribir el nombre de los atributos.

- En el caso de atributos booleanos se suele utilizar el nombre o nombres deseados precedidos de *es* (de la forma verbal *es*, o *is* en la versión inglesa) para dar el énfasis de valor booleano que responde que sí o que no ante una pregunta. Por ejemplo: *esNegrita* o *isBold*.

- Etiquetar los tipos de relaciones utilizando verbos en tercera persona del singular. En caso de requerir etiquetas compuestas, se suele utilizar un verbo seguido de una preposición y otro verbo. Las etiquetas de los tipos de relaciones siguen la grafía Pascal.

Las etiquetas para identificar los roles de las entidades en las relaciones siguen la grafía Camel.

Como práctica general, a partir de una descripción funcional de los requisitos de la base de datos, los nombres que aparecen tienden a ser candidatos de nombres para los tipos de entidades que se vayan a definir, mientras que los verbos tienden a ser utilizados para los tipos de relaciones del esquema conceptual. Los nombres de los tipos de relaciones se eligen para que se puedan leer de izquierda a derecha y de arriba abajo, como criterio general. De este modo se facilita la lectura de las relaciones. Puede haber excepciones a estas normas, y en estos casos nos podemos ayudar de los nombres de roles o podemos indicar la dirección del tipo de relación para facilitar la comprensión de las relaciones y el papel de las entidades en cada una de estas relaciones. Los atributos suelen ser nombres extraídos de las descripciones de los objetos definidos como tipos de entidades.

2.10. Ejemplo completo

En este apartado hemos visto los elementos básicos del diseño conceptual según el modelo ER. Antes de continuar con los elementos avanzados en el apartado 3, puede resultar muy ilustrativo ver una aplicación práctica de los conceptos vistos hasta ahora. Para facilitar el análisis de un ejemplo más o menos real, plantearemos a continuación una descripción y un conjunto de requisitos y restricciones que deben permitir implementar un modelo conceptual de una base de datos destinada a la gestión universitaria. Este modelo es una reducción de un modelo real y no se quiere adaptar a la gestión real de una universidad. Sólo pretende servir como modelo de ejemplo para los conceptos vistos hasta ahora.

A continuación enumeramos los diferentes aspectos de los requisitos de los usuarios que hay que tener en cuenta al realizar el diseño conceptual de la futura base de datos:

1) La universidad está formada por diferentes departamentos. Cada departamento está formado por un conjunto de profesores y está dirigido por un único profesor. Nos interesa conocer el nombre de los departamentos y el número de profesores adscrito a cada departamento.

2) De cada profesor nos interesa poder almacenar los datos personales, como, por ejemplo, nombre, DNI, edad y si es un hombre o una mujer. De los profesores que son directores de departamento, nos interesa poder identificar la fecha en la que empezaron a ejercer este cargo.

3) En la universidad se imparten un conjunto de asignaturas. Nos interesa conocer el código de cada una de estas asignaturas, el nombre, el número de créditos, la descripción y los prerrequisitos, es decir, otras asignaturas que sea

necesario haber cursado anteriormente. Además, hay que tener en cuenta que una asignatura pertenece a un único departamento y es impartida por un único profesor.

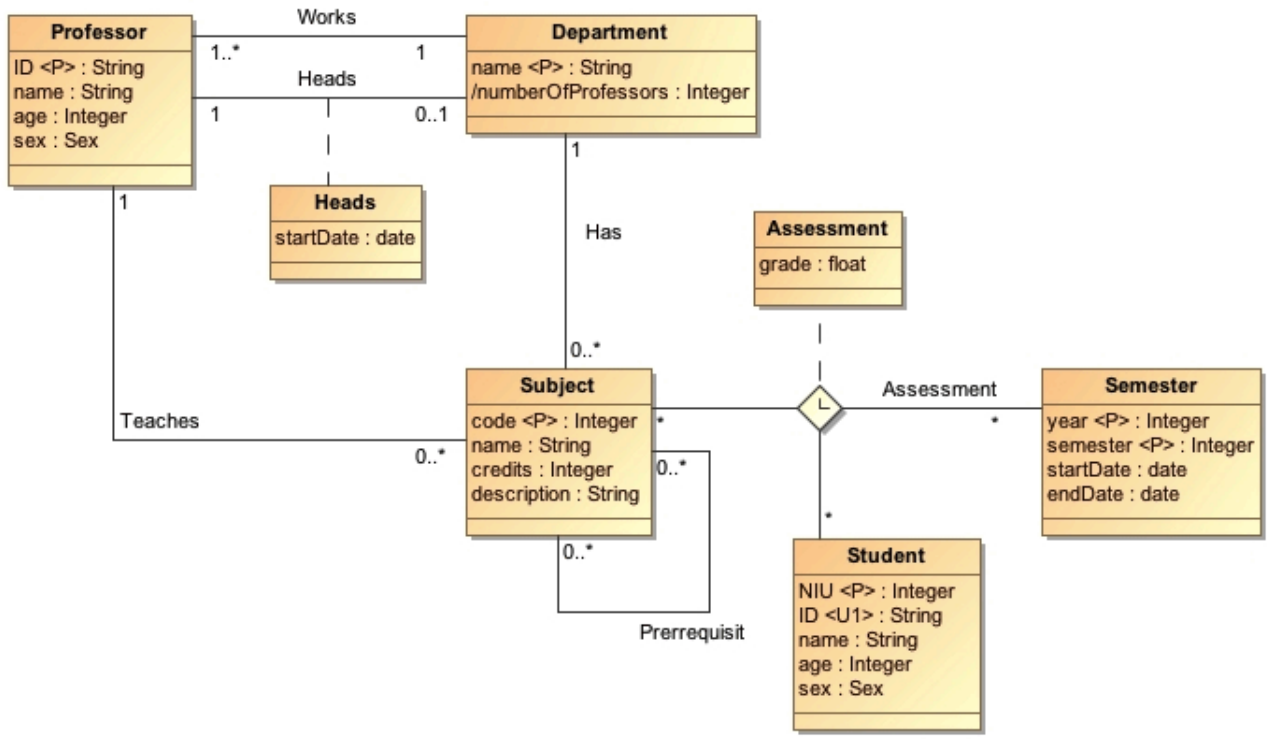
4) Los estudiantes son una parte muy importante de la base de datos, ya que se desea información sobre sus datos personales (nombre, DNI, edad, sexo) y número de identificación universitario (conocido como NIU).

5) Cada estudiante puede estar matriculado de más de una asignatura en cada semestre. Y para cada una de las asignaturas en las que está matriculado cada semestre debemos poder almacenar su nota final de aquel estudiante.

6) También se quiere dejar constancia de las fechas de inicio y final de cada semestre.

A partir de los requisitos expresados, la figura 31 muestra un diagrama del modelo conceptual. Como hemos comentado, este modelo no es único, y pueden existir diferentes aproximaciones y conceptualizaciones válidas para una misma representación del mundo real.

Figura 31. Diagrama del modelo conceptual para la base de datos de gestión universitaria



A continuación comentamos los aspectos más relevantes de este modelo conceptual:

- En el tipo de entidad 'departamento' (*Department*), el atributo 'número de profesores' (*numberOfProfessors*) es un atributo derivado, puesto que se calcula a partir del número de profesores que están vinculados a cada departamento.

- Los tipos de entidad 'departamento' y 'profesor' (*Professor*) tienen dos tipos de relaciones, para indicar por un lado en qué departamento trabaja cada profesor, y por otro lado, para determinar qué profesor es el director de cada departamento. La conectividad de la primera asociación indica que un profesor debe pertenecer a un departamento y que los departamentos deben tener como mínimo un profesor. La segunda relación es un tipo de entidad asociativa y contiene el atributo 'fecha de inicio' (*startDate*) para indicar la fecha en la que el profesor empieza la tarea como jefe de departamento. Un profesor puede impartir cero, una o más de una asignatura, como indica la conectividad del tipo de relación con el tipo de entidad 'asignatura' (*Subject*).

- El concepto *prerrequisito* de una asignatura se ha modelizado como un tipo de relación recursiva que asocia diferentes entidades de asignaturas. Tal como se desprende de la conectividad, una asignatura puede tener ninguno o múltiples prerrequisitos y ser a la vez prerrequisito de ninguna o de múltiples asignaturas.

- El semestre se ha modelizado como un tipo de entidad (*Semester*).

- El tipo de relación ternaria entre los tipos de entidad 'asignatura', 'semestre' y 'estudiante' (*Student*) determina la nota obtenida por cada estudiante como calificación de cada asignatura en la que está matriculado cada semestre. Tal como se expresaba en los requisitos, un estudiante puede estar matriculado de múltiples asignaturas en un semestre, y se puede matricular de la misma asignatura en varios semestres.

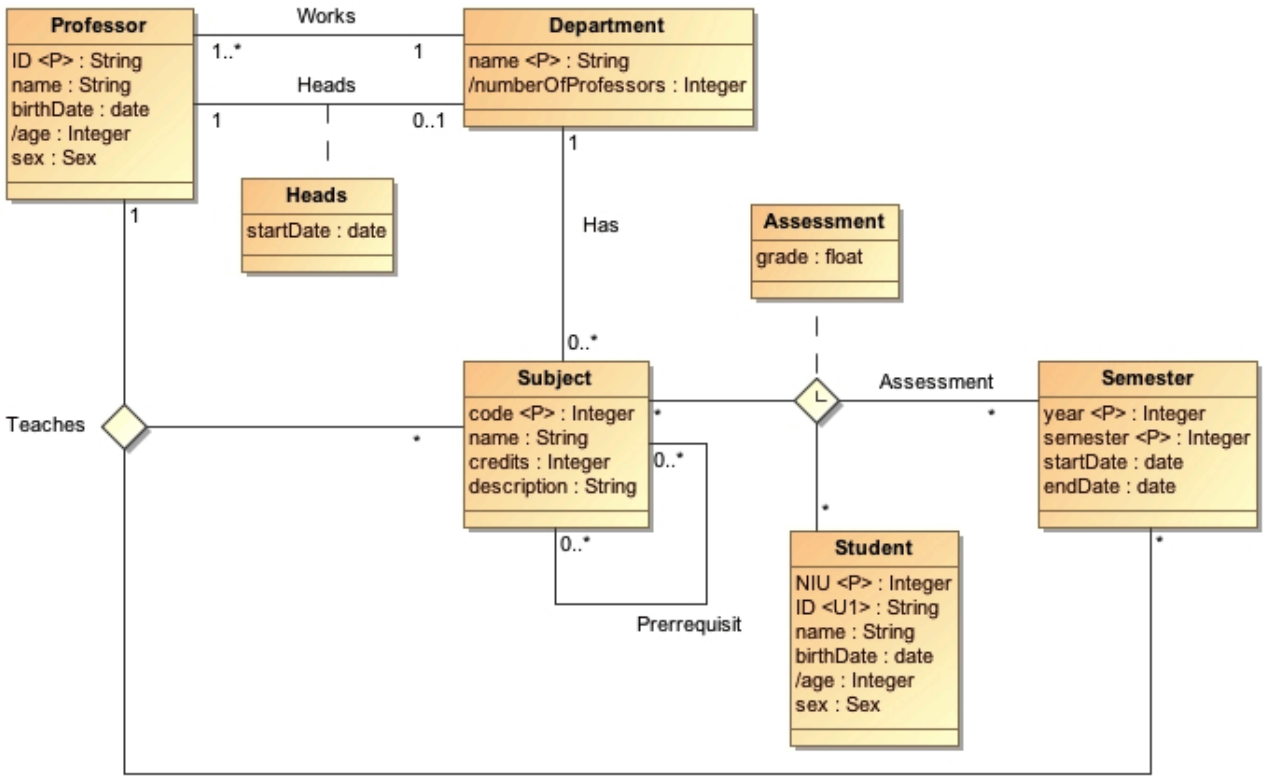
Pero este modelo conceptual también presenta algunas deficiencias, que comentamos a continuación:

- Según el modelo, un profesor imparte un conjunto de asignaturas, pero no se mantiene relación con cada uno de los semestres. Es decir, el tipo de relación 'imparte' (*Teaches*) indica el profesor que imparte una asignatura, pero no permite saber qué profesor la había impartido en semestres anteriores.

- En los tipos de entidad 'profesor' y 'estudiante' se almacena la edad mediante un atributo numérico que contiene el valor para cada profesor o estudiante en un momento determinado. Esto implica que cada año debería de actualizarse la base de datos y sumar 1 a los valores de este atributo. Lógicamente, este modelo no es el adecuado y habría que almacenar la

fecha de nacimiento de los profesores y los estudiantes y crear un atributo derivado que indique la edad (*age*) de los profesores o los estudiantes de manera automática.

Figura 32. Diagrama del modelo conceptual con las correcciones aplicadas



La figura 32 muestra el modelo conceptual modificado para corregir las deficiencias detectadas. El tipo de relación 'imparte' (*Teaches*) se ha convertido en un tipo de relación ternaria. De este modo, podemos indicar qué profesor es responsable de cada asignatura en cada semestre.

3. Elementos avanzados de modelización

Los conceptos vistos hasta ahora permiten representar una gran cantidad de los esquemas de bases de datos en aplicaciones tradicionales. Pero para poder aproximar de manera más precisa algunas características del mundo real y fomentar la reusabilidad, se han desarrollado múltiples ampliaciones y complementos a los elementos vistos hasta ahora. En este apartado veremos algunas de estas ampliaciones y de estos complementos que permiten representar realidades difícilmente expresables utilizando los elementos de modelización vistos hasta ahora.

3.1. Generalización/especialización

En algunos casos, se pueden encontrar entidades que tengan algunas características propias y diferenciadas de otras entidades del mismo tipo de entidad, y que interese poder representar estas diferencias en el modelo conceptual.

Relación 'es-un' o 'es-una'

Una relación de generalización/especialización a veces también se denomina *relación* 'es-un' o 'es-una' por la manera de hacer referencia al concepto. Por ejemplo, un estudiante 'es una' persona, un profesor 'es una' persona.

Ejemplos de generalización/especialización de tipos de entidades

Dentro de un entorno universitario podemos modelizar el tipo de entidad 'persona', que contiene atributos como por ejemplo el nombre, los apellidos, el DNI, la fecha de nacimiento, el sexo, el teléfono o el correo electrónico. Pero dentro de este tipo de entidad podemos encontrar algunas entidades con características propias que nos interesa modelizar. Las entidades que corresponden a estudiantes tienen algunas características propias relevantes (número de identificación universitario, año de primera matrícula, etc.) que hay que incluir en el modelo. Por otro lado, podemos definir otras entidades que pertenezcan al colectivo de profesores y sobre los cuales queremos representar otros tipos de atributos (departamento al que pertenecen, especialidad, etc.) o al colectivo de personal administrativo de la universidad, que tiene otros requisitos (cargo, sección, etc.). Es decir, todas las entidades tienen un conjunto de características comunes importante, pero también tienen algunas diferencias entre sí que hay que poder representar en el modelo de base de datos.

La **generalización/especialización** es una relación entre un tipo de entidad general, denominado *superclase* o *tipo de entidad padre*, y un tipo de entidad más específico, denominado *subclase* o *tipo de entidad hijo*.

Esta relación permite definir los tipos de entidades en dos niveles: en el primer nivel encontramos el tipo de entidad general (*superclase*), que contiene las características comunes a todas las entidades, y en el segundo nivel encontramos los tipos de entidades que contienen las características propias de las diferentes especializaciones (*subclases*).

Cualquier entidad de una subclase debe ser también entidad de la superclase. En sentido contrario no es imprescindible, es decir, puede haber entidades de la superclase que no pertenezcan a ninguna subclase.

Los atributos o relaciones de la superclase se propagan hacia las subclases. Es lo que se denomina **herencia de propiedades**.

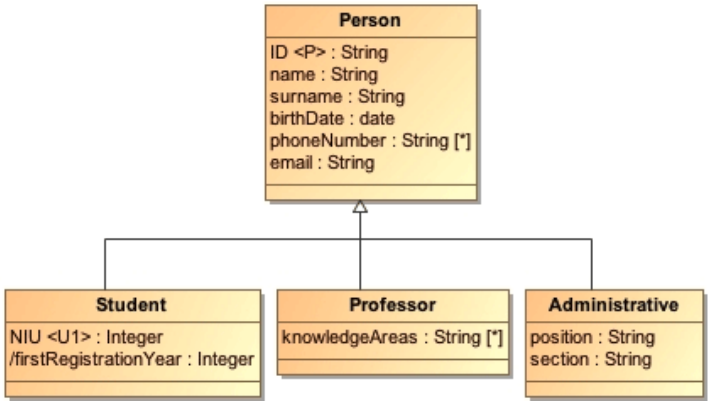
Los atributos de las subclases reciben el nombre de **atributos específicos** o **atributos locales**. Solo las entidades que pertenecen a una subclase determinada tienen valor para estos atributos. Por el contrario, cualquier entidad que pertenezca a cualquiera de las subclases debe tener valor para los atributos que se definen en la superclase. En notación UML esta relación se representa mediante una flecha vacía que sale de las subclases y que se dirige hacia la superclase.

Modelo de generalización/especialización

Siguiendo el ejemplo anterior, la figura 33 muestra un modelo de generalización/especialización para representar los diferentes tipos de personas que podemos encontrar en el entorno universitario que hemos descrito. La superclase de la relación es el tipo de entidad 'persona' (*Person*), que incluye los datos básicos que nos interesa almacenar para cualquier persona del modelo. A continuación encontramos las tres subclases descritas antes, que contienen la información relevante de los estudiantes (*Student*), profesores (*Professor*) y personal de administración (*Administrative*).

Por lo tanto, cualquier entidad de la subclase 'estudiante' tiene todos los atributos de la superclase 'persona' y además los atributos específicos o locales.

Figura 33. Ejemplo de generalización/especialización



Las subclases se denominan **especializaciones de la superclase** y la superclase se denomina **generalización de las subclases**.

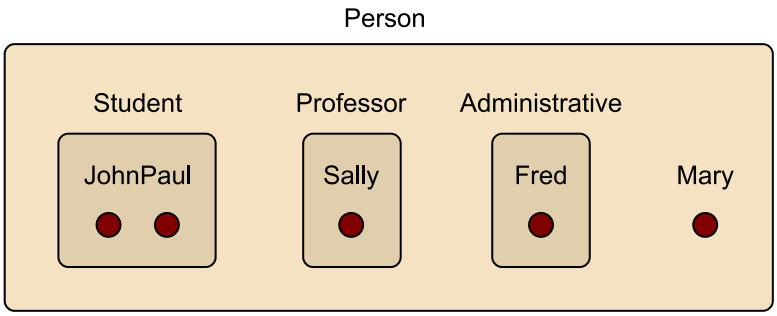
En el momento de realizar el diseño del modelo, se puede optar por una estrategia descendente (*top-down*) y empezar diseñando la superclase como tipo de entidad principal, y posteriormente refinar el diseño añadiendo las características específicas de las subclases. Esta metodología de diseño recibe el nombre de **proceso de especialización**.

La estrategia contraria, basada en una metodología ascendente (*bottom-up*), consiste en diseñar las subclases y posteriormente, en el momento de darse cuenta de las características comunes entre estas subclases, definir la superclase que las une. Esta metodología recibe el nombre de **proceso de generalización**.

Relación entre las entidades en una generalización/especialización

La figura 34 muestra una posible relación entre las entidades del ejemplo anterior. Podemos ver que hay cinco entidades de la superclase 'persona' (*Person*), con los nombres *John*, *Mary*, *Paul*, *Fred* y *Sally*, dos de las cuales son estudiantes y pertenecen a la subclase 'estudiante' (*Student*), una es un profesor (*Professor*) y otra es administrativo (*Administrative*). La entidad *Mary* no pertenece a ninguna de las subclases y solo pertenece a la superclase.

Figura 34. Ejemplo de la relación entre las entidades en una generalización/especialización

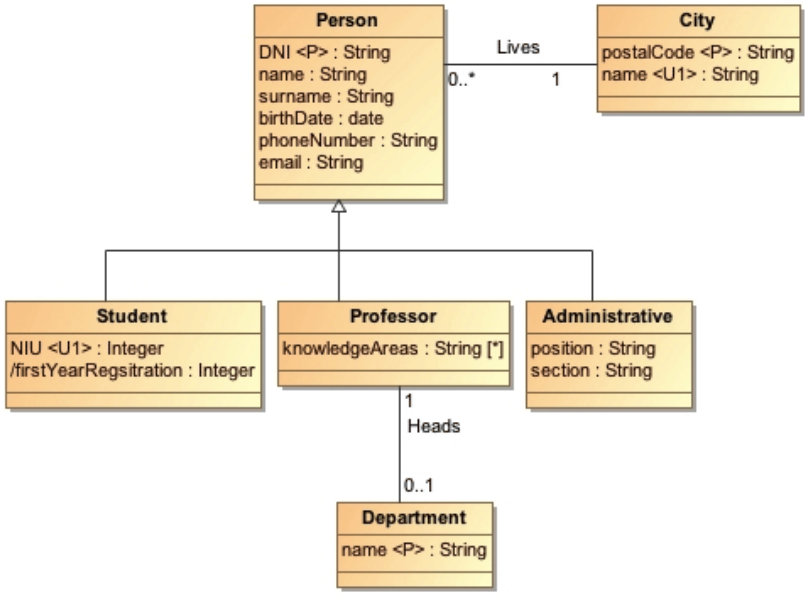


Tanto las superclases como las subclases pueden intervenir en relaciones sin ningún tipo de restricción.

Generalización/especialización con tipos de relaciones

En la figura 35 se puede ver cómo la superclase 'persona' (*person*) participa en la relación 'vive' (*Lives*) con la entidad 'población' (*City*) y la subclase 'profesor' (*Professor*) participa en la relación 'dirige' (*Heads*) con la entidad 'departamento' (*Department*). Evidentemente no tiene sentido considerar que cualquier entidad de la superclase 'persona' puede dirigir un departamento y hay que indicar en el modelo conceptual que solo aquellas personas que son entidades de la subclase 'profesor' pueden dirigir un departamento.

Figura 35. Ejemplo de generalización/especialización con tipos de relaciones



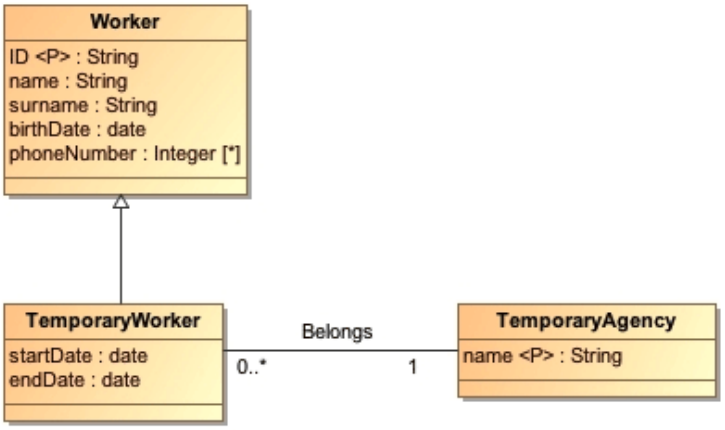
Hay dos motivos principales que inducen al uso de las relaciones de generalización/especialización entre tipos de entidades:

1) El primer motivo se debe al hecho de que en determinadas situaciones hay que modelizar casos en los que algunos atributos sólo son aplicables a algunas de las entidades, pero no a toda la superclase. Por lo tanto, se define una relación de generalización/especialización que permite agrupar estas entidades en

una subclase y añadirles los atributos necesarios, mientras continúan formando parte de la superclase y compartiendo el resto de los atributos con todas las entidades de la superclase. La figura 35 muestra un ejemplo de este caso.

2) La segunda situación se produce cuando hay que modelizar tipos de relaciones que solo se pueden establecer con algunas entidades de la superclase. Por ejemplo, supongamos que hemos creado un tipo de entidad 'trabajador' (*Worker*) para modelizar a los trabajadores de una fábrica y queremos establecer un tipo de relación que indique de qué empresa de trabajo temporal provienen. Lógicamente, esta relación sólo es aplicable a los trabajadores que sean temporales y no a los trabajadores fijos de la empresa. Para modelizar esta situación podemos crear una subclase para identificar a los 'trabajadores temporales' (*TemporaryWorker*) y relacionarlos con el tipo de entidad de las empresas temporales (*EmploymentAgency*). De este modo, solo los trabajadores que pertenezcan a la subclase podrán estar relacionados con la empresa de trabajo temporal. La figura 36 muestra un esquema que ejemplifica este caso.

Figura 36. Ejemplo de generalización/especialización que incorpora relaciones con la subclase



Por lo tanto, un proceso de generalización/especialización permite:

- Crear una taxonomía de tipos de entidades, definiendo un conjunto de entidades tipo específicas (subclases) a partir del tipo de entidad genérica (superclase).
- Establecer atributos específicos adicionales a cada tipo de entidad específico (subclase).
- Establecer tipos de relaciones adicionales entre los tipos de entidades específicos (subclases) y otros tipos de entidad.

Observación

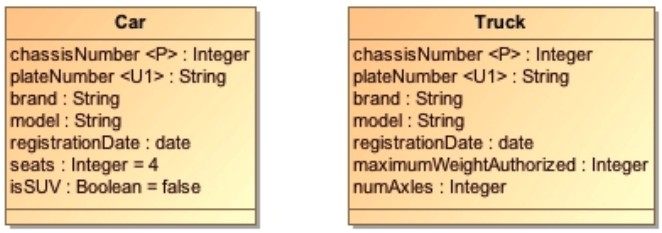
A partir de este punto, denominamos *superclase* a los tipos de entidad genéricos, y *subclase*, a los tipos de entidad específicos.

Generalización de tipo de entidad

Supongamos que queremos modelizar una base de datos para mantener información sobre los vehículos que tiene una empresa. En un momento del diseño aparece el tipo de entidad 'coche' (*Car*), del que queremos almacenar el número de bastidor, la matrícula, la marca, el modelo, la fecha de matriculación, el número de plazas disponibles y si es un vehículo todoterreno o no. Más adelante, en la misma fase de diseño, nos aparece el tipo de entidad 'camión' (*Truck*), del que queremos almacenar el número de bastidor, la matrícula, la marca, el modelo, la fecha de matriculación, el peso máximo autorizado de

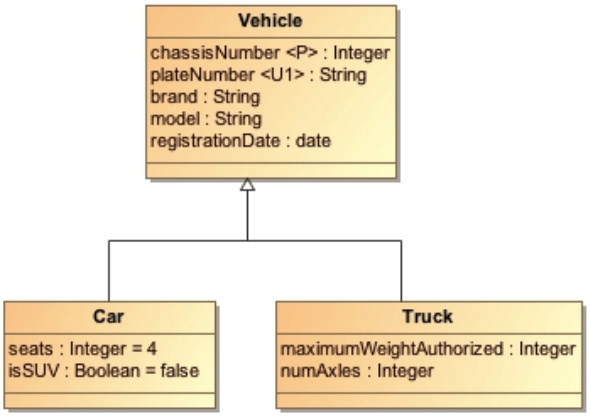
carga y el número de ejes del camión. La figura 37 muestra el diseño conceptual de los dos tipos de entidades.

Figura 37. Ejemplo de los tipos de entidad 'coche' (*Car*) y 'camión' (*Truck*)



Pero en este punto el diseñador se da cuenta de que los dos tipos de entidad comparten una buena parte de los atributos y de que sería interesante generalizar esta parte común para obtener una nueva superclase que permita identificar un vehículo, sea camión o coche. Por lo tanto, se crea la superclase 'vehículo' (*Vehicle*), que contiene todos los atributos comunes (el número de bastidor, la matrícula, la marca, el modelo y la fecha de matriculación), y entonces se crean dos subclases con los atributos propios de cada una. La figura 38 muestra el diseño conceptual después del proceso de generalización.

Figura 38. Ejemplo de generalización de los tipos de entidad 'coche' (*Car*) y 'camión' (*Truck*)



3.1.1. Restricciones en la generalización/especialización

Hay tres tipos de restricciones básicas en los procesos de generalización/especialización.

1) Pertenencia definida por atributo o usuario

En algunos casos la superclase contiene un conjunto de atributos que indican, para cada entidad, qué subclase pertenece. En estos casos se denominan **subclases de predicado definido** o **subclases de condición definida**.

Ejemplo de pertenencia definida por atributo o usuario

Continuando con el ejemplo anterior de los vehículos, podemos especificar a qué subclase pertenece cada entidad a partir de un atributo que define el tipo de vehículo (*vehicleType*) de la superclase, que tome valor igual a *car* en caso de que la entidad tenga que pertenecer a la subclase *Car* o que tome valor igual a *truck* en caso de que tenga que pertenecer a la subclase *Truck*.

En caso de que todas las subclases tengan una condición de pertenencia en el mismo atributo de la superclase, el atributo recibe el nombre de **atributo definitorio** o **discriminador** y se dice que la relación de generalización/espe-

cialización es de **atributo definido**. En caso de que no exista ningún atributo que permita identificar la pertenencia de las entidades a las diferentes subclases, se dice que es una generalización/especialización **definida por el usuario**.

En UML, el atributo discriminador, si lo hay, se indica en la flecha de la relación de generalización/especialización. La figura 40 muestra un ejemplo de generalización/especialización de atributo definido.

2) Restricción de exclusividad

Otra restricción que se puede aplicar a las generalizaciones/especializaciones es la **restricción de exclusividad**, que indica si las subclases han ser disjuntas entre sí. Dicho de otro modo, expresa si una misma entidad puede pertenecer a más de una subclase. En este sentido, la generalización/especialización puede ser:

- **Disjunta**[3]: una entidad solo puede pertenecer a una única subclase.
- **Solapada**[4]: una entidad puede pertenecer a más de una subclase al mismo tiempo.

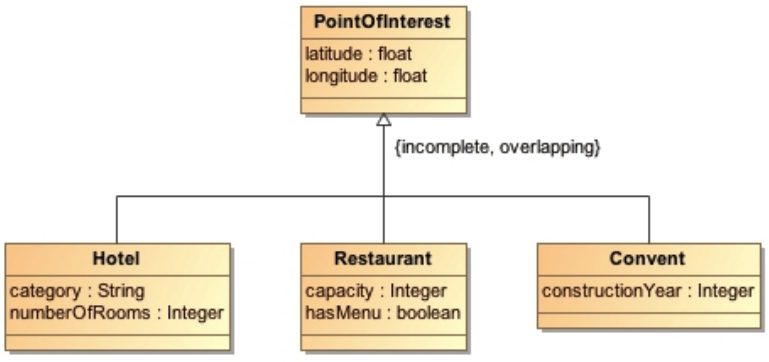
(3)En inglés, *disjoint*.

(4)En inglés, *overlapping*.

Generalización/especialización solapada

La figura 39 muestra un ejemplo de generalización/especialización solapada. En este modelo podemos ver la superclase 'punto de interés' (*PointOfInterest*), que permite almacenar información sobre diferentes puntos de interés turístico de una zona. Este concepto se puede especializar en diferentes subclases. Por ejemplo, podemos decir que los hoteles (*Hotel*), los restaurantes (*Restaurant*) y los conventos (*Convent*) son especializaciones de 'punto de interés'. Debe tenerse en cuenta que puede haber hoteles que también son restaurantes, o conventos que hacen funciones de hotel. Esto implica que una misma entidad puede pertenecer a más de una subclase al mismo tiempo. Por lo tanto, podemos decir que esta generalización/especialización es de tipo solapada.

Figura 39. Ejemplo de generalización/especialización solapada



3) Restricción de participación

Finalmente, la última de las restricciones referentes a la generalización/especialización se denomina **restricción de participación** e indica la obligatoriedad de las entidades de pertenecer a alguna de las subclases o no. Siguiendo este enfoque se puede establecer la clasificación siguiente:

- **Total**[5]: toda entidad de la superclase debe pertenecer a alguna de las subclases.

- **Parcial**[6]: puede haber entidades de la superclase que no pertenezcan a ninguna de las subclases.

(5)En inglés, *complete*.

(6)En inglés, *incomplete*.

En una generalización/especialización con restricción de participación total, todas las entidades deben pertenecer a una o más subclases. Por lo tanto, todas las entidades pertenecen a alguna de las subclases y no puede haber entidades que pertenezcan exclusivamente a la superclase. En estos casos se dice que la superclase es un **tipo de entidad abstracta**, puesto que no contiene ninguna entidad. Los tipos de entidades abstractas en UML se indican con el nombre del tipo de entidad en letra cursiva.

Si en una generalización/especialización no se indican las restricciones, por defecto se asume que se trata de una relación **definida por el usuario**, **solapada** y **parcial**.

La figura 40 muestra la nomenclatura para identificar los dos casos en UML. En la misma relación se indica entre llaves {} los conceptos de total o parcial (*complete*/*incomplete*) y disjunta o solapada (*disjoint*/*overlapping*).

Generalización/especialización disjunta parcial y completa

Las figuras 40 y 41 muestran un esquema similar, con una diferencia importante en el diseño.

En la figura 40 podemos ver la superclase 'vehículo' (*Vehicle*) y las subclases 'coche' (*Car*) y 'camión' (*Truck*) con una herencia disjunta e incompleta. Esto nos indica, en primer lugar, que una entidad no puede pertenecer a las dos subclases al mismo tiempo, es decir, que no puede haber un vehículo que sea coche y camión a la vez, y, en segundo lugar, que algunas entidades puede ser que no pertenezcan a ninguna de las subclases, es decir, que puede haber instancias de vehículos que no son coches ni camiones.

En la figura 41, en cambio, se presenta una herencia similar, pero en este caso es completa. El hecho de que sea completa implica que la superclase no podrá contener entidades; por lo tanto, será un tipo de entidad abstracta. Es decir, que cualquier entidad de 'vehículo' deberá ser un coche o un camión.

Figura 40. Ejemplo de generalización/especialización disjunta (*disjoint*) y parcial (*incomplete*)

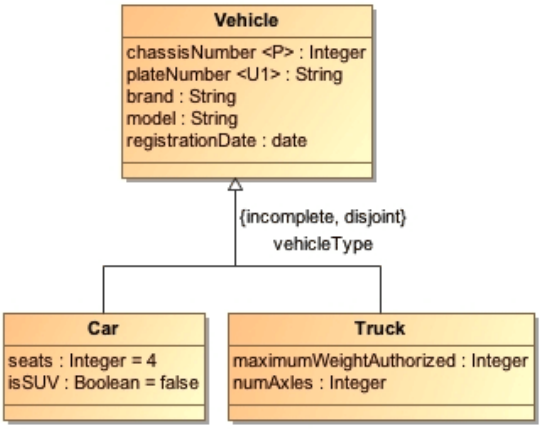
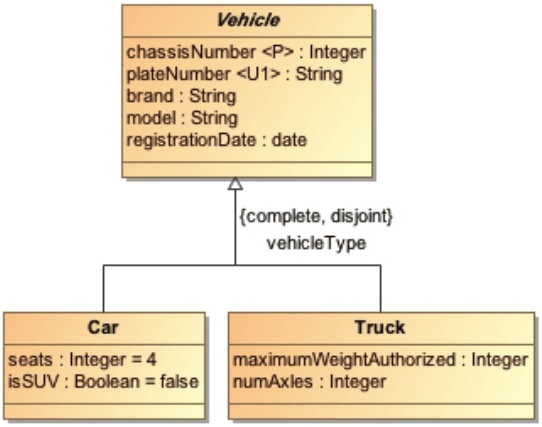


Figura 41. Ejemplo de generalización/especialización disjunta (*disjoint*) y completa (*complete*)



Los conceptos de *restricciones de exclusividad* y de *participación* son independientes entre sí, y, por lo tanto, se pueden combinar para dar lugar a cuatro posibles restricciones de generalización/especialización:

- Disjunta y total
- Disjunta y parcial
- Solapada y total
- Solapada y parcial

3.1.2. Herencia simple y múltiple

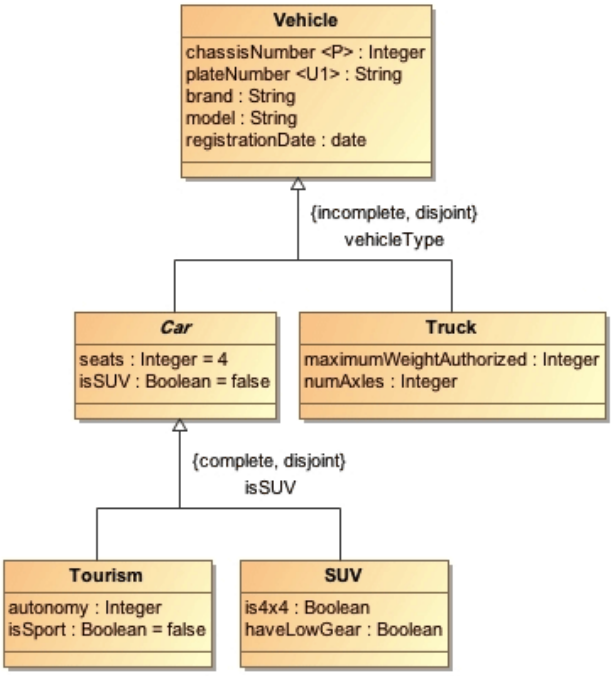
Una subclase puede tener a la vez subclases que permitan refinar más los conceptos.

La **herencia simple** en un proceso de generalización/especialización se da cuando todos los tipos de entidades involucradas en la relación sólo tienen una superclase, es decir, hay un único tipo de entidad padre o clase raíz. Como resultado de esta condición se genera una estructura en forma de árbol.

Herencia simple

La figura 42 muestra una relación de herencia simple, en la que a partir de la superclase raíz 'vehículo' (*Vehicle*) y del atributo discriminador 'tipo de vehículo' (*vehicleType*) se refinan dos subclases según si el vehículo es un coche (*Car*) o un camión (*Truck*). Al mismo tiempo, el tipo de entidad 'coche' se especializa en dos subclases, según si es un turismo (*Tourism*) o un vehículo todoterreno (*SUV*). El tipo de entidad 'coche' es superclase en esta nueva relación y a la vez subclase en la relación con el tipo de entidad 'vehículo'. Hay que señalar, además, que el tipo de entidad 'coche' se convierte en abstracta, puesto que la especialización es completa.

Figura 42. Ejemplo de herencia simple



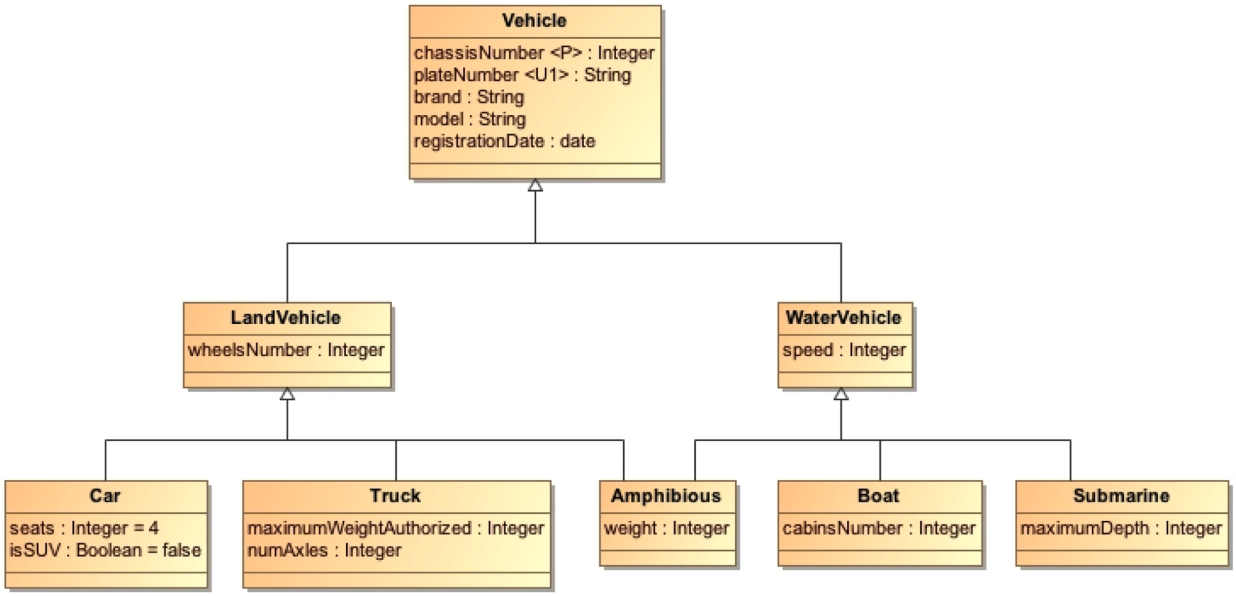
El tipo de entidad origen de la jerarquía, en este caso el tipo de entidad 'vehículo', es la **superclase raíz** o **tipo de entidad base**. Las subclases que no tienen otras subclases, en este caso las clases 'turismo', 'todoterreno' y 'camión' son **clases hija**.

La **herencia múltiple** en un proceso de generalización/especialización se da cuando alguno de los tipos de entidad involucrados en la relación tiene dos o más superclases. En estos casos se obtiene una estructura de tipo grafo dirigido.

Herencia múltiple

La figura 43 muestra una relación de herencia múltiple. A partir del tipo de entidad base, en este ejemplo el tipo de entidad 'vehículo' (*Vehicle*), creamos una especialización en la que indicamos si se trata de un vehículo terrestre (*LandVehicle*) o acuático (*WaterVehicle*). Los vehículos terrestres se pueden especializar en coches (*Car*), camiones (*Truck*) o anfibios (*Amphibious*). Los vehículos anfibios se pueden desplazar por tierra y por medios acuáticos, y, por lo tanto, este tipo de entidad también es una especialización del tipo de entidad 'vehículo acuático', junto con los tipos de entidades 'barco' (*Boat*) y 'submarino' (*Submarine*).

Figura 43. Ejemplo de herencia múltiple



El tipo de entidad 'anfibio' se denomina *subclase compartida*, puesto que tiene más de una superclase o tipo de entidad padre. Hay que señalar que si un atributo o relación que se origina en una misma superclase es heredado más de una vez por caminos diferentes del entramado, sólo se puede incluir una sola vez en la subclase compartida. Es decir, los atributos y las relaciones de la clase 'vehículo' sólo pueden aparecer una única vez en la subclase compartida.

Es importante darse cuenta de que la existencia de una única subclase compartida provoca que la relación sea de herencia múltiple en lugar de simple.

Observación

Una relación de herencia múltiple no puede contener ciclos.

3.1.3. Clasificación múltiple

Un **modelo de clasificación múltiple** es aquel que permite que una entidad sea instancia de dos tipos de entidades, E_1 y E_2, de modo que E_1 no es subclase de E_2, E_2 no es subclase de E_1 y no hay ningún tipo de entidad E_3 que sea subclase de E_1 y E_2.

Las relaciones de generalización/especialización solapadas permiten clasificación múltiple, puesto que una entidad puede pertenecer a diferentes subclases a la vez. La figura 39 muestra un ejemplo de generalización/especialización solapada que permite clasificación múltiple.

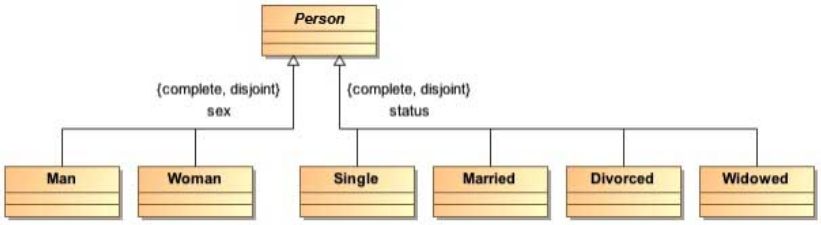
Otro tipo de clasificación múltiple se da cuando la superclase tiene dos o más jerarquías de especialización según diferentes discriminantes. Cada entidad puede pertenecer a diferentes subclases a la vez.

Generalización/especialización con clasificación múltiple

Partiendo del tipo de entidad 'persona' (*Person*) podemos establecer un proceso de especialización para definir su estado civil, que indique si la persona está soltera (*Single*), ca-

sada (*Married*), divorciada (*Divorced*) o viuda (*Widowed*). Por otro lado, una persona también se puede especializar, según el sexo, en 'hombre' (*Man*) o 'mujer' (*Woman*). La figura 44 muestra el modelo conceptual descrito.

Figura 44. Ejemplo de generalización/especialización con clasificación múltiple



3.2. Agregación y composición

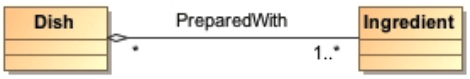
La **agregación** es un tipo de relación entre entidades que indica una relación de "parte-todo" entre las instancias.

En esta asociación se distingue la parte que representa el "todo", y que recibe el nombre de **compuesto**, y las diferentes "partes" que lo componen, y que reciben el nombre de **componentes**.

Agregación

La figura 45 muestra un ejemplo de agregación entre un plato (*Dish*) y los ingredientes con los que se ha preparado (*Ingredient*). Como se puede ver en el modelo conceptual, un plato debe estar compuesto por uno o más ingredientes, mientras que un mismo ingrediente puede estar incluido en cualquier número de platos.

Figura 45. Ejemplo de agregación



La distinción entre tipos de relación y agregación es sutil, y a menudo subjetiva. En general, se considera que es necesario que haya una cierta relación de ensamblaje, sea física o lógica, entre las entidades para hablar de *agregación* en lugar de *tipo de relación*. La única restricción que añade la agregación respecto al tipo de relación es que las cadenas de agregaciones entre entidades no pueden formar ciclos.

La composición es un tipo concreto de agregación en el que la multiplicidad del tipo de entidad compuesta debe ser como máximo 1 y los componentes presentan dependencia de existencia hacia el compuesto que tienen.

Es decir, la composición impone dos restricciones respecto a la agregación:

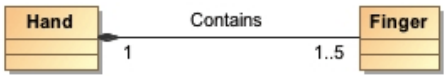
- Cada componente puede estar presente en un único compuesto (la multiplicidad del tipo de entidad compuesta debe ser como máximo 1).

- Si se elimina el elemento compuesto, hay que eliminar todos los componentes vinculados a este elemento (los componentes presentan dependencia de existencia hacia el elemento compuesto).

Composición

Podemos pensar en el conjunto de dedos que forman una mano. La figura 46 muestra el modelo conceptual en el que se representa la composición de una mano (tipo de entidad *Hand*) como conjunto de dedos (tipos de entidad *Finger*) y donde cada dedo solo puede pertenecer a una sola mano.

Figura 46. Ejemplo de composición



Observación

La agregación se marca con un rombo hueco, mientras que la composición se marca con un rombo negro.

A diferencia de la agregación, en la composición hay una fuerte dependencia entre la parte compuesta y la parte componente, de modo que los componentes no pueden vivir sin el compuesto.

3.3. Restricciones de integridad

3.3.1. Restricciones en los tipos de entidad

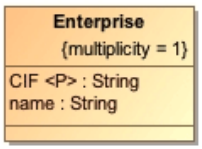
La **multiplicidad de tipo de entidad** establece un rango de posibles cardinalidades para las entidades de un determinado tipo de entidad. Es decir, indica cuántas instancias de un determinado tipo de entidad pueden aparecer en la base de datos.

Generalmente, y por defecto, esta restricción toma el valor indefinido, pero algunas veces puede ser interesante poder determinar un intervalo en el número de ocurrencias de un tipo de entidad, especialmente en los casos en los que sólo puede haber una entidad.

Tipos de entidad con multiplicidad 1

Supongamos que queremos modelizar una base de datos para gestionar pequeñas o medianas empresas. Además de toda la información de clientes, proveedores, productos y otras cosas, hay que almacenar los datos de la empresa (CIF, nombre, dirección, etc.) para poder mostrar esta información en facturas, listas y otros datos. Una manera de modelizar este requisito es crear un tipo de entidad que permita almacenar esta información. En este caso, hay que especificar una multiplicidad de tipo de entidad igual a 1, puesto que sólo debe haber una entidad con los datos de la empresa. La figura 47 muestra el diagrama UML de este tipo de entidad.

Figura 47. Tipos de entidad con multiplicidad 1



3.3.2. Restricciones en los atributos

En la definición de los atributos hemos visto algunas restricciones básicas asociadas a los atributos. Estas restricciones son las siguientes:

- **Restricciones de dominio**: son las restricciones asociadas al conjunto de posibles valores legales que puede tomar un atributo. En este sentido, se puede especificar el tipo de datos que utilizará el atributo (cadena de texto, entero, real, booleano, etc.) y las restricciones adicionales específicas para cada tipo de datos (por ejemplo, la longitud en el caso de cadenas de texto o un intervalo válido en el caso de valores enteros).

- **Atributos derivados**: son atributos que calculan el valor a partir de otros atributos y que, por lo tanto, implícitamente son atributos de sólo lectura desde el punto de vista del usuario o de la aplicación. No obstante, su valor puede cambiar si cambia el valor de los atributos de los que dependen.

Restricciones de cambiabilidad

Esta restricción indica si los valores de un atributo pueden cambiar o no.

Hay cuatro valores permitidos:

- 'Cambiable' (*changeable* o *unrestricted*): es el valor por defecto. Indica que se permite cualquier cambio sobre el atributo.

- 'Congelado' (*frozen* o *readOnly*): indica que una vez que se ha asignado valor al atributo no se podrá modificar ni eliminar.

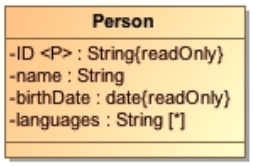
- 'Solo-añadir' (*addOnly*): indica que una vez que se ha asignado valor al atributo no se podrá modificar ni eliminar. Pero, en caso de ser un atributo multivalor, sí que se podrán asignar nuevos valores para este atributo.

- 'Solo-eliminar' (*removeOnly*): indica que la única operación permitida es eliminar el valor del atributo.

Restricciones de cambiabilidad en los atributos

La figura 48 muestra el tipo de entidad *Person*, donde podemos ver un atributo no restringido o cambiable (*Name*) y dos atributos congelados o de sólo lectura (*ID* y *birthDate*).

Figura 48. Ejemplo de restricciones de cambiabilidad en los atributos



3.3.3. Restricciones en los tipos de relaciones

Aparte de las restricciones en la multiplicidad en los tipos de relaciones, es posible expresar otras restricciones sobre las relaciones en los diagramas UML. A pesar de que la mayoría de los casos quedan cubiertos con las restricciones que hemos visto hasta ahora, estas nuevas restricciones permitirán una mayor expresividad del modelo conceptual. A continuación mencionamos algunas de las más relevantes.

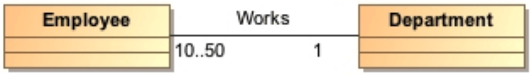
a) Restricciones de cardinalidad *min..max*

Además de todas las restricciones referentes a la cardinalidad de las relaciones que hemos visto, es posible indicar un valor o rango concreto en la cardinalidad de una relación. Para indicar un rango en UML, se expresa el valor mínimo seguido de dos puntos y el valor máximo (*min..max*).

Ejemplo

La figura 49 modeliza una situación en la que un empleado (*Employee*) sólo puede pertenecer a un único departamento (*Department*) y cada departamento debe tener entre 10 y 50 empleados.

Figura 49. Ejemplo de restricción de cardinalidad *min..max*



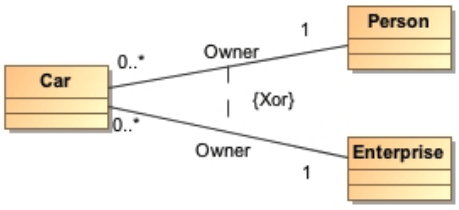
b) Restricción *xor*

Esta restricción se puede aplicar cuando hay varias relaciones ligadas a un mismo tipo de entidad. La restricción indica que una entidad sólo puede participar en uno de los tipos de relación unidos por esta restricción.

Ejemplo

Un coche puede pertenecer a una persona o a una empresa. Para modelizar esta situación, la figura 50 muestra cómo los tipos de relaciones que unen los tipos de entidad 'coche' (*Car*) con los tipos de entidad 'persona' (*Person*) y 'empresa' (*Enterprise*) están relacionados con la restricción *xor*, que indica que un coche ha de ser de una persona o de una empresa, pero no de ambos a la vez.

Figura 50. Ejemplo de restricción *xor*



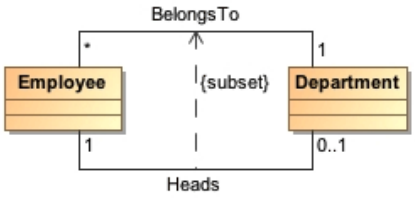
c) Restricción *subset*

Esta restricción indica que un tipo de relación es un subconjunto de otro tipo de relación.

Ejemplo de restricción *subset*

El jefe de un departamento debe pertenecer a este departamento. Para indicar esta restricción, podemos utilizar el concepto *subset* tal como se muestra en la figura 51.

Figura 51. Ejemplo de restricción *subset*



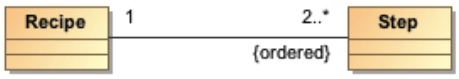
d) Restricción *ordered*

Esta restricción indica que hay una relación de orden en la asociación entre los tipos de entidades.

Ejemplo de restricción *ordered*

Una receta de cocina está formada por un conjunto de dos o más pasos que deben seguir un orden determinado. Para indicar esta restricción, podemos utilizar el concepto *ordered* tal como se muestra en la figura 52.

Figura 52. Ejemplo de restricción *ordered*



e) Restricciones de cambiabilidad

Esta restricción indica si los valores del extremo de una relación pueden cambiar o no.

Hay cuatro valores permitidos:

- 'Cambiable' (*changeable* o *unrestricted*): es el valor por defecto. Indica que se permite cualquier cambio sobre la asociación.

- 'Congelado' (*frozen* o *readOnly*): indica que una vez que la relación ha sido creada no se podrá modificar ni eliminar. Tampoco se podrán crear nuevas relaciones.

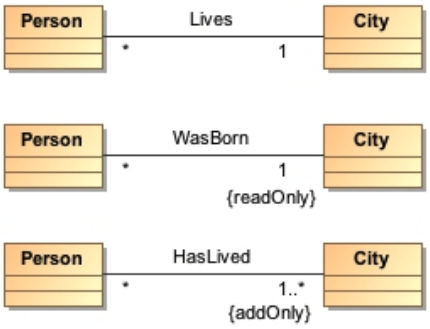
- 'Solo-añadir' (*addOnly*): indica que una vez que la relación ha sido creada no se podrá modificar ni eliminar. Pero sí que se podrán crear nuevas relaciones.

- 'Solo-eliminar' (*removeOnly*): indica que la única operación permitida es eliminar el valor de la relación.

Restricciones de cambiabilidad en los tipos de relaciones

Para poder comparar diferentes ejemplos relacionados con la cambiabilidad, la figura 53 nos muestra tres situaciones diferentes en las que intervienen los mismos tipos de entidades. En la primera, representamos la ciudad donde vive una persona, que evidentemente puede cambiar. Por lo tanto, el extremo de la asociación que conecta con el tipo de entidad 'ciudad' (*City*) es cambiable o no restringida (opción por defecto). En la segunda situación, representamos el lugar de nacimiento de una persona, y en este caso etiquetamos como congelado el extremo de esta asociación. Finalmente, si consideramos las ciudades donde ha vivido una persona, etiquetamos con el atributo 'Solo-añadir', puesto que la lista de ciudades donde ha vivido una persona puede aumentar, pero no modificar o eliminar valores existentes.

Figura 53. Ejemplos de restricciones de cambiabilidad en los tipos de relaciones



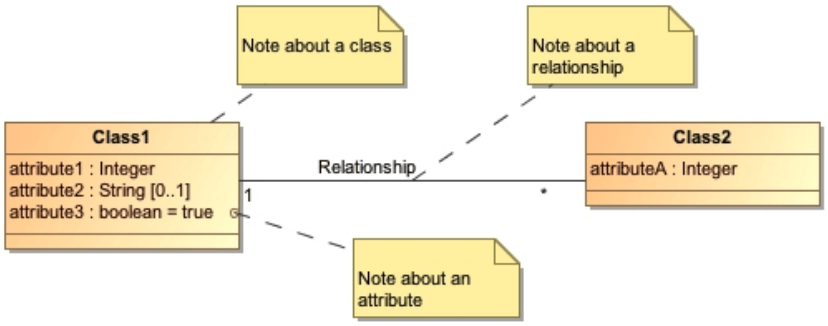
3.3.4. Otras restricciones

En el análisis de requisitos muchas veces se pueden encontrar requisitos o restricciones de carácter semántico. Estas restricciones dependen completamente del problema, y hay que dejar constancia de ellas. Generalmente, se utilizan notas ligadas a los tipos de entidades, atributos o tipos de relaciones a los que hacen referencia para dejar constancia de ellas.

Uso de notas para indicar otras restricciones

La figura 54 muestra tres notas que permiten notaciones de cualquier tipo para clarificar conceptos del modelo, asociadas a un tipo de entidad, un atributo o un tipo de relación.

Figura 54. Ejemplo de uso de notas para indicar requisitos o restricciones del modelo



3.4. Modelización de datos históricos

Las bases de datos modelizan el estado de algunos aspectos del mundo exterior. Generalmente, sólo modelizan un estado del mundo real –el estado actual– y no almacenan información sobre los estados anteriores. Cuando se produce un cambio de estado en el mundo real, la base de datos se actualiza y se pierde la información sobre los estados anteriores. No obstante, en algunas aplicaciones es importante poder almacenar y recuperar información sobre los estados anteriores del sistema.

Básicamente hay dos tipos de requisitos a la hora de modelizar el tiempo asociado a un estado del mundo real:

1) Instante de tiempo: en algunas acciones sólo hay que señalar en qué instante de tiempo han ocurrido. Para modelizar este tipo de actividad, se puede añadir una marca de tiempo que permita identificar en cada instancia el momento de tiempo con el que está asociado. Este comportamiento se puede modelizar de dos maneras:

- Añadiendo un atributo de tipo 'fecha y hora'. Cada instancia indica el valor de tiempo en este atributo.

- Añadiendo una relación con el tipo de entidad 'fecha' (*Date*), que contiene un atributo de tipo 'fecha y hora'.

2) Intervalo de tiempo: otras acciones tienen un tiempo de vida. Es decir, están activas durante un intervalo de tiempo específico. En estos casos hay que indicar el intervalo de tiempo en el que es válida cada una de las entidades. Este comportamiento se puede modelizar de dos maneras:

- Si se quiere aplicar este criterio sobre un concepto que está modelizado como un tipo de entidad, se pueden añadir dos atributos de tipo fecha y hora en la entidad para indicar el inicio y el final del intervalo.

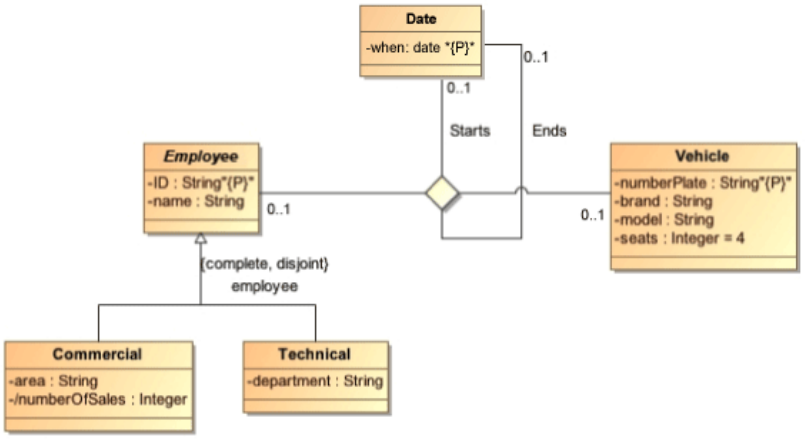
- Si se quiere aplicar este criterio sobre un concepto que está modelizado como una relación, hay que añadir en la relación un tipo de entidad 'fecha' (*Date*) que determinará los tiempos inicial y final del intervalo.

Modelización de datos históricos

La lectura de un sistema de monitorización de una fábrica irá tomando diferentes lecturas en diferentes estados de tiempo. Nos interesa guardar el valor de la lectura y el momento de tiempo en el que se ha producido esta lectura.

En otras situaciones nos interesa definir no un instante de tiempo, sino un intervalo de tiempo. Por ejemplo, supongamos que una empresa tiene un conjunto de vehículos con diferentes características y un conjunto de comerciales y personal técnico que los utiliza. Según las tareas y las visitas de cada día, cada uno de los comerciales y técnicos puede necesitar un vehículo diferente. Por lo tanto, a esta empresa le interesará saber qué coche ha estado utilizando cada uno de los comerciales y técnicos en cada momento (por ejemplo, en caso de recibir una sanción de tráfico). Para modelizar esta situación hay que definir un tipo de relación cuaternaria en la que intervienen un comercial, un vehículo y dos entidades de tiempo (una que indica el inicio de la asociación y otra que indica cuándo la asociación deja de ser válida). La figura 55 muestra un modelo que implementa la situación descrita.

Figura 55. Ejemplos de situación con modelización de datos históricos



La tabla 2 muestra, a modo de ejemplo, una posible configuración de entidades del ejemplo anterior en un instante de tiempo concreto.

Tabla 2. Ejemplo de representación de entidades del modelo de la figura 55

ID	name	numberPlate	starts	ends
33941857B	John	8754-GFD	2001-05-12 09:25	2001-05-12 17:36
33941857B	John	1258-CGC	2001-05-17 11:13	--
15488574Q	Mary	8754-GFD	2001-05-10 15:11	2001-05-10 19:47
15488574Q	Mary	1258-CGC	2001-05-13 07:02	2001-05-14 14:41
15488574Q	Mary	1126-BMR	2001-05-16 20:14	--
25486257F	Fred	1126-BMR	2001-05-11 09:13	2001-05-11 18:56

3.5. Ejemplo completo

A continuación planteamos un ejemplo completo para poder ver un esquema conceptual en el que aparezcan algunos de los elementos vistos en este apartado. Continuaremos con el ejemplo del apartado anterior, que se basa en el diseño de una base de datos para la gestión universitaria.

A continuación enumeramos los diferentes aspectos de los requisitos de los usuarios que hay que tener en cuenta al realizar el diseño conceptual de la base de datos:

1) La universidad está formada por diferentes departamentos. Cada departamento está formado por un conjunto de profesores y está dirigido por un único profesor. Nos interesa conocer el nombre de los departamentos y el número de profesores que trabajan en cada uno, así como su localización física, la dirección, teniendo en cuenta que una misma dirección puede existir en más de una ciudad.

2) Para cada profesor nos interesa poder almacenar sus datos personales como, por ejemplo, nombre, DNI, fecha de nacimiento, sexo, dirección y números de teléfono. Para los profesores que son directores de departamento, nos interesa poder identificar la fecha en la que empezaron a ejercer este cargo.

3) En la universidad se imparten un conjunto de asignaturas. Interesa saber, para cada una de estas asignaturas, el código, el nombre, el número de créditos, la descripción, si tiene laboratorio asociado y los prerrequisitos, es decir, otras asignaturas que hay que haber cursado anteriormente. Además, hay que tener en cuenta que una asignatura pertenece a un único departamento.

4) Los departamentos están agrupados en escuelas o facultades. Por ejemplo, el departamento de matemáticas pertenece a la facultad de ciencias. Un departamento pertenece a una única escuela o facultad.

5) Los estudiantes son una parte muy importante de la base de datos. Se quiere almacenar información sobre los datos personales de estos estudiantes (nombre, DNI, fecha de nacimiento, sexo, dirección y números de teléfono) y su número de identificación universitaria (conocido como NIU).

6) Cada estudiante puede estar matriculado de varias asignaturas en cada semestre. Y para cada una de las asignaturas en las que está matriculado cada semestre, tenemos que poder almacenar una nota final.

7) También se quiere dejar constancia de las fechas de inicio y final de cada semestre.

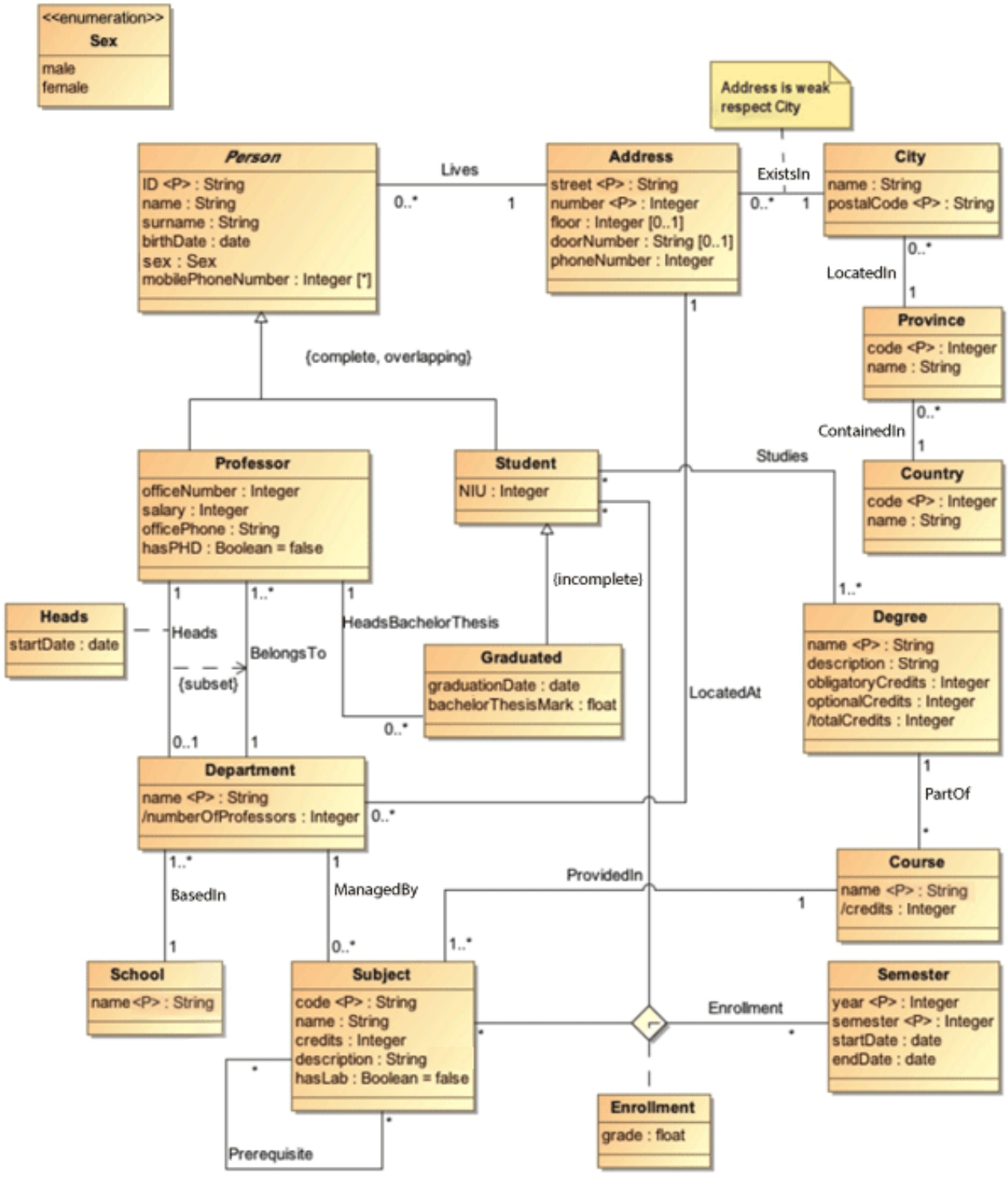
8) Las asignaturas se agrupan en los diferentes cursos que forman cada uno de los grados que ofrece la universidad. Por ejemplo, la asignatura de *Álgebra* pertenece al primer curso del grado de Matemáticas. Sobre cada uno de los cursos sólo nos interesa almacenar el conjunto de asignaturas que lo forman. Sobre cada uno de los grados, nos interesa almacenar información referente a su número de créditos (obligatorios, opcionales y totales) y una descripción.

9) Los estudiantes estudian uno o más grados.

10) Cuando un estudiante se gradúa, elabora un proyecto de final de carrera sobre el que nos interesa almacenar información acerca de la fecha en la que se ha presentado, la nota que ha obtenido y el profesor que lo ha dirigido.

A partir de los requisitos expresados, la figura 56 muestra un diagrama del modelo conceptual. Como hemos comentado, este modelo no es único, sino que puede haber diversas aproximaciones e interpretaciones válidas para una misma representación del mundo real.

Figura 56. Diagrama del modelo conceptual para la segunda aproximación de la base de datos de gestión universitaria



Resumen

En este módulo hemos visto el proceso de diseño conceptual. Este proceso es una de las etapas del diseño de bases de datos, concretamente, es la segunda etapa, y se realiza después del análisis de requisitos.

El diseño conceptual permite crear un esquema conceptual de alto nivel e independiente de la tecnología de implementación a partir de las especificaciones y los requisitos de un problema del mundo real.

El enfoque de este material nos ha permitido ver las bases del diseño conceptual empleando los diagramas UML como sistema de notación.

En la primera parte de este material hemos visto una breve introducción en la que hemos detallado las bases, los objetivos y los requisitos del diseño conceptual y de los diagramas en lenguaje UML.

En la segunda parte hemos visto los elementos básicos de modelización en el diseño conceptual. Entre los elementos principales hay que destacar los tipos de entidades, los atributos y los tipos de relaciones. A pesar de que estos tres elementos forman la base principal del modelo conceptual, también hemos visto los tipos de entidades asociativas y los tipos de entidades débiles, que permiten una mayor riqueza en la representación del modelo conceptual.

Finalmente, en la tercera parte de este material hemos visto algunos de los elementos avanzados en el diseño conceptual. Los elementos vistos en la parte anterior no permiten representar situaciones que encontramos en el mundo real y que queremos poder representar en nuestro modelo. Fruto de esta necesidad se amplía el modelo para incluir conceptos como la generalización/especialización, la agregación o la composición, que permiten una mayor riqueza en la representación del modelo conceptual. Para finalizar este texto, nos hemos referido brevemente a las restricciones de integridad básicas y a la modelización de datos históricos.

Glosario

atributo *m* Propiedad que interesa representar de un tipo de entidad.

clase *f* Nombre que reciben los tipos de entidades en el modelo UML.

conectividad *f* Expresión del tipo de correspondencia entre las entidades de una relación.

diseño conceptual *m* Etapa del diseño de una base de datos que obtiene una estructura de la información de la futura base de datos independiente de la tecnología que se quiere emplear.

diseño lógico *m* Etapa del diseño de una base de datos que parte del resultado del diseño conceptual y lo transforma de manera que se adapte al modelo de sistema gestor de bases de datos con el cual se desea implementar la base de datos.

generalización/especialización *f* Construcción que permite reflejar que existe un tipo de entidad general, llamada superclase, que se puede especializar en diferentes tipos de entidades más específicas, llamadas subclases.

grado de una relación *m* Número de entidades que asocia la relación.

entidad *f* Objeto del mundo real que podemos distinguir del resto de objetos y del cual nos interesan algunas propiedades.

lenguaje unificado de modelización *m* Lenguaje gráfico para modelizar, visualizar, especificar, construir y documentar sistemas de software o de bases de datos.
sigla UML
en unified modeling language

modelo entidad-interrelación *m* Modelo de datos de alto nivel ampliamente utilizado para el diseño conceptual de las aplicaciones de bases de datos. El objetivo principal del modelo ER es permitir a los diseñadores reflejar en un modelo conceptual los requisitos del mundo real.

sistema gestor de bases de datos *m* Tipo de software específico dedicado a servir de interfaz entre la base de datos, el usuario y las aplicaciones que la utilizan.
sigla SGBD
en database management system (DBMS)

tipo de relación *m* Asociación entre entidades.

tipo de entidad asociativa *m* Tipo de entidad resultante de considerar una relación entre entidades como una nueva entidad.

tipo de entidad débil *m* Tipo de entidad cuyos atributos no la identifican completamente, sino que solo la identifican de manera parcial.

tipo de relación recursiva *m* Asociación a la cual alguna entidad está asociada más de una vez.

Bibliografía

Elmasri, Ramez; Navathe, Shamkant, B. (2007). *Fundamentos de sistemas de bases de datos* (5.ª ed.). Madrid: Pearson Educación.

Teorey, T. J. (2008). *Database design: Know it all*. Burlington: Morgan Kaufmann.

Camps, R.; Cassany M. J.; Conesa, J.; Costal, D.; Figuls, D.; Martín, C.; Rius, A.; Rodríguez, M. E.; Urpí, T. (2011). *Uso de bases de datos*. FUOC.

Date, C. J. (2001). *Introducción a los sistemas de bases de datos*. Madrid: Pearson Educación.

Rumbaugh, James; Jacobson, Ivar; Booch, Grady (2007). *El lenguaje unificado de modelado. Manual de referencia* (2.ª ed.). Madrid: Pearson Educación.