

Análisis UML

Jordi Pradel Miquel
Jose Raya Martos

PID_00171155

Ninguna parte de esta publicación, incluido el diseño general y la cubierta, puede ser copiada, reproducida, almacenada o transmitida de ninguna forma, ni por ningún medio, sea éste eléctrico, químico, mecánico, óptico, grabación, fotocopia, o cualquier otro, sin la previa autorización escrita de los titulares del copyright.

Índice

Introducción.....	5
Objetivos.....	6
1. Análisis orientado a objetos con UML.....	7
1.1. Análisis orientado a objetos	7
1.2. El lenguaje UML	7
1.2.1. Motivación del lenguaje UML	7
1.2.2. Origen del lenguaje UML	9
1.2.3. Tipos de diagramas UML	9
1.3. Ejemplo	14
2. Modelo de casos de uso.....	15
2.1. El diagrama de casos de uso	15
2.1.1. Actores	16
2.1.2. La frontera del sistema	16
2.1.3. Los casos de uso	16
2.1.4. Relaciones entre casos de uso y actores	17
2.1.5. Relaciones entre casos de uso: inclusión	17
2.1.6. Relaciones entre casos de uso: extensión	19
2.2. El diagrama de actividades	19
2.2.1. Lógica condicional	20
2.2.2. Paralelización	21
2.2.3. Organización: carriles	22
2.2.4. Otros elementos de notación	23
2.3. El modelo resultante	23
3. Modelización de la interfaz.....	26
3.1. Casos de uso concretos	26
3.2. Modelos de las pantallas	27
3.2.1. Diagrama de estado de las pantallas (mapa navegacional)	28
3.3. Contratos de las operaciones del sistema	29
4. Modelo del dominio.....	31
4.1. Convenciones en los diagramas UML	31
4.2. Clases	32
4.2.1. Notación UML	32
4.2.2. Técnicas de modelización	32
4.3. Atributos	36
4.3.1. Notación UML	36

4.3.2. Técnicas de modelización	37
4.4. Asociaciones	40
4.4.1. Notación UML	40
4.4.2. Técnicas de modelización	40
4.5. Herencia	49
4.5.1. Notación UML	49
4.5.2. Técnicas de modelización	50
4.6. Información derivada y reglas de integridad	56
4.6.1. Reglas de integridad	56
4.6.2. Información derivada	60
4.7. Clases y atributos: elementos avanzados	61
4.7.1. Tipos de datos	61
4.7.2. Atributos: notación UML avanzada	63
4.8. Asociaciones: elementos avanzados	63
4.8.1. Composición	63
4.8.2. Asociaciones con repeticiones o con orden	64
4.8.3. Notación UML avanzada	65
4.8.4. Asociaciones ternarias (y N-arias en las que $N > 2$)	66
4.9. El modelo resultante	67
 Resumen	70
 Actividades	71
 Ejercicios de autoevaluación	75
 Solucionario	76
 Glosario	85
 Bibliografía	87

Introducción

Tal como hemos visto en el módulo "Orientación a objetos", la orientación a objetos es un paradigma que puede resultar muy útil no solamente para tareas de programación, sino también para tareas de análisis. Por otro lado, en el módulo "Requisitos" hemos visto los requisitos y la importancia de realizar una documentación de requisitos de calidad, y también los casos de uso como una manera flexible de documentar requisitos funcionales.

En este módulo veremos combinar el uso de la orientación a objetos y los casos de uso para hacer una documentación de requisitos usando UML. El tipo de documentación que elaboraremos es muy completo y puede llegar a ser formal, pero puede resultar útil para todo tipo de metodologías.

En primer lugar, veremos cómo el concepto de orientación a objetos se puede aplicar al análisis y cómo UML constituye un lenguaje estándar para la creación de modelos visuales de software. También veremos una pequeña introducción al lenguaje y mostraremos los diferentes tipos de diagramas que soporta.

A continuación, explicaremos cómo podemos usar UML para complementar las especificaciones textuales de casos de uso. Veremos el diagrama de casos de uso, que representa de manera visual los casos de uso, los actores y las relaciones entre éstos, y el diagrama de actividades, que permite modelar visualmente el comportamiento de un caso de uso como si fuera un proceso.

Llegados a este punto, los casos de uso que habremos visto no están todavía enlazados con una interfaz gráfica de usuario. El paso siguiente será, por lo tanto, crear un modelo de esta interfaz que nos permitirá resolver dudas sobre el uso o la arquitectura de la información, entre otras.

Finalmente, veremos cómo podemos hacer modelización del dominio usando UML. El modelo del dominio será una representación de las clases conceptuales del mundo real en el dominio del sistema que estudiamos.

A lo largo de todo el módulo utilizaremos, como hilo conductor, ejemplos basados, todos, en un mismo sistema: una universidad que quiere ofrecer un campus virtual para sus alumnos y profesores.

Objetivos

Los objetivos que el estudiante debe haber alcanzado una vez trabajados los contenidos de este módulo son:

- 1.** Saber usar la orientación a objetos para hacer análisis de software para sistemas de información.
- 2.** Saber utilizar la notación UML para documentar modelos de análisis orientados a objetos.
- 3.** Saber emplear los casos de uso para llevar a cabo análisis funcional de software para sistemas de información.
- 4.** Saber usar los diagramas de actividades para documentar en detalle los casos de uso complejos como procesos.
- 5.** Saber modelizar la interfaz gráfica de usuario del software mediante casos de uso concretos, esbozos de las pantallas y mapas navegacionales.
- 6.** Saber hacer modelización del dominio mediante diagramas de clases UML.

1. Análisis orientado a objetos con UML

1.1. Análisis orientado a objetos

La orientación a objetos nació como una herramienta de programación en la que los programadores construyen una abstracción del problema que están tratando de resolver, en lugar de construir una abstracción del ordenador. Los programadores que usan orientación a objetos, por lo tanto, crean clases de software que representan aquellos conceptos del mundo real que son relevantes para el sistema.

Ved también

La orientación a objetos se estudia en el módulo "Orientación a objetos".

A medida que la orientación a objetos fue ganando terreno, los analistas se dieron cuenta de que podía ser una herramienta muy útil para construir sus modelos de análisis. Además, resultaba muy útil usar modelos deliberadamente parecidos en el software que se construye, puesto que expresar el análisis en términos fácilmente trasladables en software orientado a objetos facilita no solamente la tarea de diseño y programación, sino también el cambio, la extensión y el mantenimiento del software una vez desarrollado.

1.2. El lenguaje UML

1.2.1. Motivación del lenguaje UML

El lenguaje UML es el lenguaje estándar para crear modelos visuales de software.

Como tal, UML es un lenguaje de propósito general (intenta cubrir todos los posibles modelos de todos los tipos de software) y visual (los modelos se representan, principalmente, en forma de diagramas).

Uno de los motivos del éxito del lenguaje UML es que es independiente al método de desarrollo empleado. Así pues, diferentes métodos de desarrollo utilizarán de manera diferente los diagramas propuestos por UML.

Como lenguaje, la utilidad principal del UML es permitir la comunicación. La adopción por parte de toda la industria de un mismo lenguaje permite que todo el mundo pueda entender los modelos creados por otra persona o equipo y, por lo tanto, que se puedan discutir las propiedades de un sistema a partir de los modelos.

Según Fowler (2004), podemos distinguir tres usos principales para el lenguaje UML:

- **Como lenguaje para hacer un croquis.** Se trata de hacer un uso informal (normalmente dibujando a mano alzada) para explicar o explorar algún aspecto en concreto de un sistema informático. La clave aquí es la selectividad: sólo tenemos en cuenta los detalles que son relevantes para la discusión que estamos llevando a cabo en este momento y obviamos el resto. De este modo, conseguimos maximizar la relación entre el esfuerzo dedicado a la creación de los modelos y la utilidad obtenida en cambio.
- **Como lenguaje para hacer un plano.** Construir modelos más detallados que nos permitan documentar el sistema con suficiente detalle para facilitar la implementación (incluso con generación automática de código) o para capturar de manera visual los detalles sobre la estructura y el comportamiento de un sistema existente (ingeniería inversa). La idea es poder conseguir que un diseñador cree los planos del sistema y que, más adelante, los programadores sólo tengan que escribir el código, pero no tomar decisiones sobre cómo ha de ser el sistema por desarrollar.
- **Como lenguaje de programación.** Éste sería el caso de las herramientas MDA. Se trata de crear modelos tan detallados del sistema que el código que se pueda generar de manera automatizada no tenga que ser modificado (ni siquiera leído) por los desarrolladores. Éste es un campo en el que, actualmente, se están haciendo muchos esfuerzos de investigación.

Otra manera de caracterizar el uso del UML es teniendo en cuenta qué se quiere modelizar. Según Fowler (2004), podemos distinguir entre dos perspectivas:

- **Perspectiva conceptual.** El modelo representa una descripción de los conceptos del dominio que estamos estudiando.
- **Perspectiva de software.** El modelo representa el sistema informático y, por lo tanto, existe una correspondencia directa entre los elementos del modelo y las partes del sistema.

A la hora de estudiar el lenguaje UML habrá que decidir qué uso querremos hacer de él bajo estas dos clasificaciones, puesto que en función de la necesidad que tengamos, usaremos el UML de una u otra manera.

En cuanto a la primera clasificación, la mayoría de los modelos que crearemos serán croquis del que sería un sistema real. Esto nos permitirá centrarnos en los aspectos didácticos de los modelos sin tener que preocuparnos por los detalles del sistema real. Por otro lado, esto también nos permitirá no tener que depender de ninguna herramienta CASE en concreto.

Ved también

Podéis encontrar más información sobre MDA en el módulo "Introducción a la ingeniería del software".

En cuanto a la segunda clasificación, durante el análisis nos situaremos en la perspectiva conceptual. Los modelos que realizamos hacen referencia a los conceptos del dominio que estamos analizando (el espacio del problema), pero no al software en sí mismo (el espacio de la solución).

1.2.2. Origen del lenguaje UML

Para comprender el valor (y algunos de los defectos) del UML, es importante conocer su origen. Antes de la publicación del UML cada método de desarrollo utilizaba su propia notación. Esto era un problema, puesto que dificultaba enormemente la comunicación entre ingenieros de software.

Parte del problema residía en que, mientras que todas las notaciones eran más o menos similares, había multitud de detalles que cambiaban de una a otra, lo que hacía todavía más complicada la comunicación. A pesar de que hubo algunos esfuerzos de estandarización, la reacción por parte de los metodólogos no fue muy positiva.

Esta situación empezó a cambiar cuando Jim Rumbaugh pasó a trabajar, junto con Grady Booch, para la compañía Rational, en la creación del **método unificado**. Más adelante, Ivar Jacobson se unió a la compañía y, entre los tres, sentaron las bases del que sería el lenguaje UML.

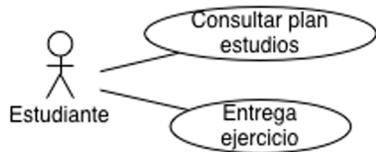
Durante la segunda mitad de la década de los noventa tuvo lugar la estandarización, por medio del OMG, del lenguaje UML. De este modo, UML dejaba de ser un producto de una sola compañía (Rational) para pasar a ser un estándar abierto que cualquier otra compañía podía implementar.

Este proceso de estandarización fue clave para que se incorporara el trabajo de otros metodólogos y se pudiera llegar a un estándar que fuera aceptado por toda la industria. Como contrapartida, tenemos un estándar que, en ocasiones, puede parecer un tanto inconsistente, puesto que incorpora elementos con orígenes muy dispares.

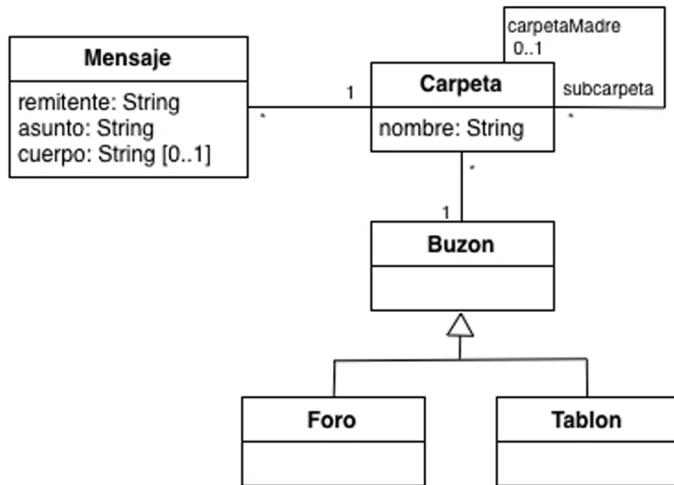
1.2.3. Tipos de diagramas UML

La versión 2 UML define trece tipos de diagrama. A continuación, veremos un ejemplo de cada tipo de diagrama. La idea no es estudiar los diagramas en detalle, sino hacernos una idea general de su aspecto visual y su utilidad.

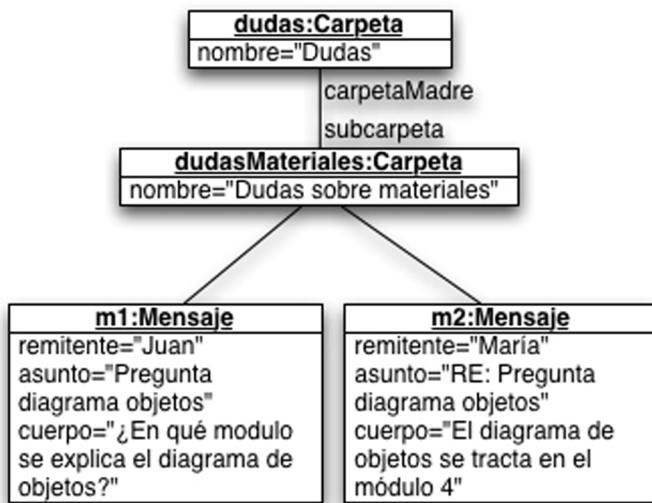
El **diagrama de casos de uso** (*use case diagram*) sirve para modelizar cuáles son los casos de uso del sistema y cuáles son los actores que están relacionados con ellos.



El **diagrama de clases** (*class diagram*) es, probablemente, el diagrama más conocido del UML. Sirve para modelizar las clases de objetos y sus relaciones, haciendo hincapié en los aspectos estructurales más que en el comportamiento.

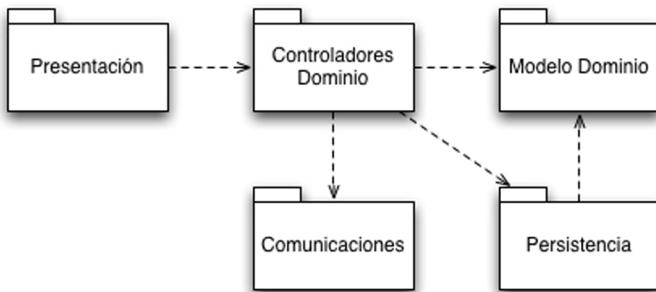


La versión 2 del lenguaje UML oficializó por primera vez el **diagrama de objetos** (*object diagram*). Éste es un diagrama similar al de clases en el que lo que representamos no son clases, sino instancias de clase (objetos).

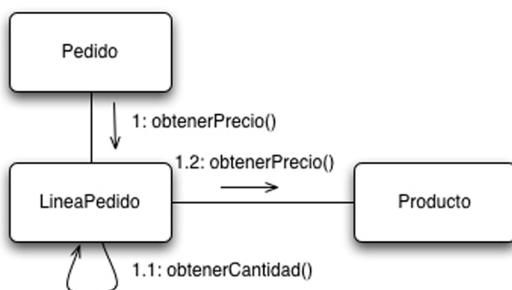


El **diagrama de paquetes** (*package diagram*) nos permite modelizar las dependencias entre paquetes. Un paquete es un grupo de elementos UML, como, por ejemplo, las clases o los casos de uso. Por lo tanto, podemos usar la notación de paquetes en otros diagramas. Así pues, un paquete de un diagrama

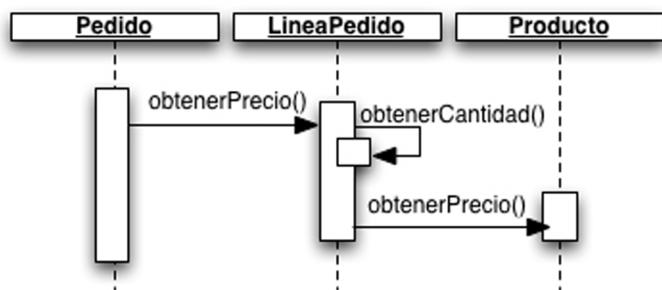
de paquetes puede representar un subsistema, un grupo de clases o cualquier otra agrupación de elementos. En el diagrama de paquetes mostraremos los paquetes y las dependencias entre éstos.



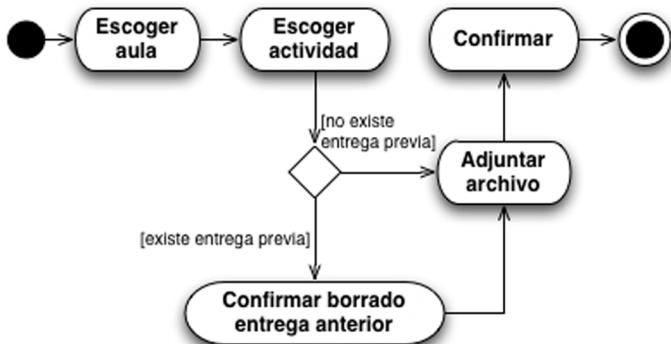
El **diagrama de comunicación** (*communication diagram*) (llamado *diagrama de colaboración* en la versión 1 de UML) nos permite modelizar la interacción entre varios objetos poniendo énfasis en el aspecto estructural (esto es, qué objetos están conectados a qué otros y colaboran con ellos).



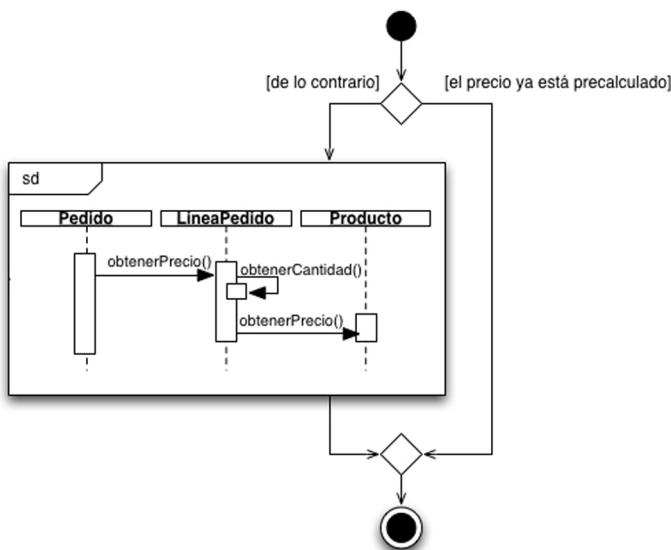
El **diagrama de secuencia** (*sequence diagram*) es similar al de comunicación en el sentido de que modeliza lo mismo (la interacción entre varios objetos) pero, en este caso, poniendo énfasis en la secuencia temporal.



El **diagrama de actividades** (*activity diagram*) se utiliza, habitualmente, para describir procesos de manera similar a los diagramas *flow-chart*.



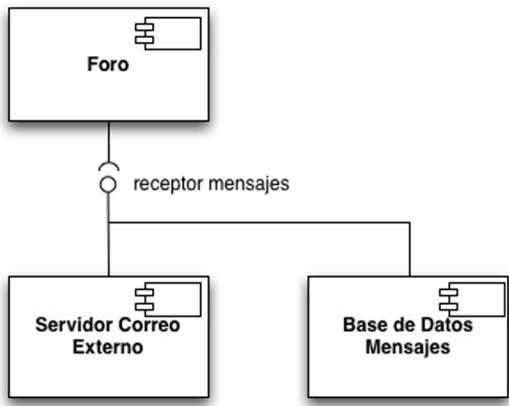
Los dos diagramas anteriores (secuencia y actividades) los podemos combinar en un **diagrama de visión general de interacción** (*interaction overview diagram*), que lo podemos ver como un diagrama de actividades en el que hemos sustituido las actividades por diagramas de secuencia. Éste es un tipo de diagrama que introdujo por primera vez la versión 2 de UML y no está tan difundido como los otros.



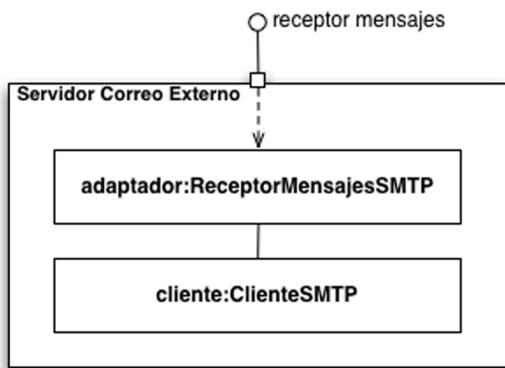
El **diagrama de estados** (*state machine diagram*) modeliza estados y transiciones. La idea del diagrama de estados es modelizar cómo afectan a un objeto los diferentes acontecimientos que pueden tener lugar en el sistema.



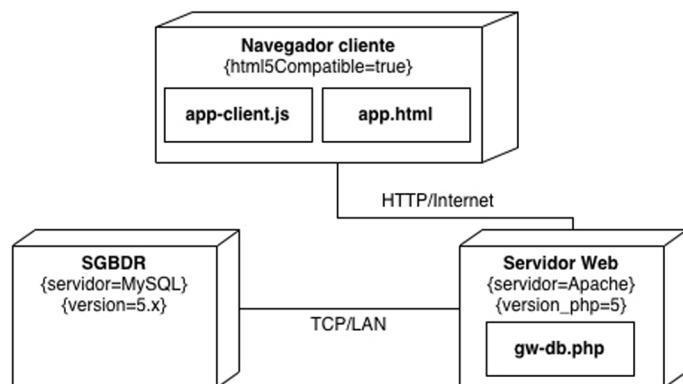
El **diagrama de componentes** (*component diagram*) modeliza los diferentes componentes que forman parte de nuestro sistema y las interfaces que usan para comunicarse entre sí.



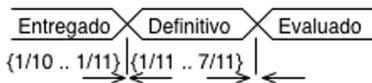
El **diagrama de estructura compuesta** (*composite structure diagram*) nos permite modelizar la estructura interna en tiempo de ejecución de una clase o componente.



El **diagrama de despliegue** (*deployment diagram*) modeliza la distribución física, en tiempo de ejecución, de los diferentes artefactos de software.



El **diagrama de tiempo** (*timing diagram*) es muy habitual en el mundo de la electrónica y, desde su versión 2, forma parte del conjunto de diagramas estándar de UML, a pesar de que su aplicabilidad al desarrollo del software se limita a casos muy concretos, como por ejemplo el software de tiempo real.



Software de tiempo real

El software de tiempo real es aquel que se usa en sistemas en los que hay restricciones de tiempo real, esto es, en los que hay un límite de tiempo entre que se produce un acontecimiento y que el sistema le da respuesta.

1.3. Ejemplo

A lo largo de este módulo usaremos un ejemplo de dominio que iremos desarrollando en pequeños ejemplos a lo largo del módulo.

Tomaremos como ejemplo una universidad para la que queremos desarrollar un sistema de información que gestione la información de los cursos impartidos, las matrículas de los estudiantes y las actividades que realizan, incluyendo las calificaciones.

La universidad tiene una serie de asignaturas, para cada una de las cuales se imarte una edición (que también llamamos *cursos*) cada semestre. Para cada curso de una asignatura se crean una serie de grupos, cada uno de los cuales tiene un profesor y una serie de alumnos matriculados.

Algunos cursos son presenciales; para éstos queremos saber el horario semanal y, para cada franja horaria del curso, el aula donde se imparte. Las aulas están situadas en edificios que, a su vez, pertenecen a un determinado campus.

Hay, también, un campus virtual, donde cada grupo dispone de un tablero (un buzón de mensajes en el que el profesor puede escribir pero los alumnos sólo pueden leer) y un foro (un buzón de mensajes donde tanto el profesor como los alumnos pueden participar). Los mensajes, dentro del tablero o foro, se pueden agrupar por carpetas.

Los profesores proponen, cada curso, unas actividades comunes a todos los grupos de la asignatura (como por ejemplo exámenes o prácticas). La calificación de cada alumno en una asignatura se calculará a partir de las calificaciones que saque en cada actividad entregada.

2. Modelo de casos de uso

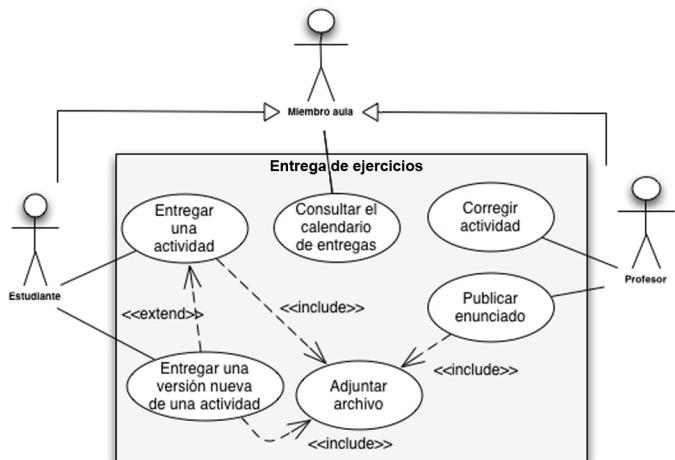
Anteriormente hemos visto la técnica de los casos de uso para la documentación de requisitos. También hemos visto que la notación más habitual para documentar los casos de uso es la documentación textual. En este apartado, veremos cómo podemos utilizar el lenguaje UML para complementar la documentación de los casos de uso mediante diagramas.

Ved también

Hemos visto la técnica de los casos de uso para la documentación de requisitos en el módulo "Requisitos".

2.1. El diagrama de casos de uso

El primer diagrama que veremos es el diagrama de casos de uso. La notación del diagrama es muy sencilla; de hecho, a continuación veremos un ejemplo de este diagrama en el que se utiliza toda la notación necesaria para este módulo.



Hay que fijarse, no obstante, en que este diagrama no contiene información sobre el comportamiento del sistema (por ejemplo, qué ocurre si un estudiante intenta entregar una actividad fuera de plazo).

Ved también

El diagrama de casos de uso complementa, pero en ningún caso sustituye, la descripción textual de los casos de uso, puesto que no incluye información sobre cuál es el comportamiento del sistema.

Podéis encontrar más información sobre los casos de uso y su especificación textual en el módulo "Requisitos".

¿Cuál es, pues, la utilidad de este diagrama? Por un lado, nos permite relacionar visualmente los actores y los casos de uso y, por el otro, nos proporciona una visión rápida de cuál es la funcionalidad que el sistema ofrece a los diferentes actores.

2.1.1. Actores

En el diagrama del ejemplo observamos tres actores: "Estudiante", "Profesor" y "Miembro aula", que se representan con una especie de muñeco. Esto es porque, habitualmente, los actores serán personas (los usuarios del sistema) a pesar de que, incluso en el caso de que un actor sea un sistema externo, lo representaremos con el mismo símbolo.

La flecha que va de Estudiante a Miembro aula (y la que va de Profesor a Miembro aula) significa que este actor es una especialización del actor *Miembro aula*. A efectos prácticos, esto quiere decir que todo lo que se dice sobre el actor *Miembro aula* es aplicable al actor *Estudiante* (y también al actor *Profesor*).

La ventaja del uso de la relación de generalización/especialización entre actores es que nos permite simplificar el diagrama, puesto que elimina la necesidad de representar repetidamente aquello que es común a más de un actor (como, por ejemplo, sus relaciones con los casos de uso). Así, en el ejemplo, no nos es necesario representar la relación de estudiantes y profesores con el caso de uso "Consultar el calendario de entregas", puesto que la hemos representado en el actor más general.

2.1.2. La frontera del sistema

El recuadro con la etiqueta "Entrega ejercicios" representa la frontera del sistema: todo lo que queda dentro del recuadro forma parte de él, mientras que todo lo que queda fuera del recuadro (en este caso, los actores) no.

La etiqueta nos puede ayudar a aclarar el ámbito de los casos de uso: qué sistema o subsistema estamos teniendo en cuenta.

2.1.3. Los casos de uso

Los casos de uso se representan mediante elipses que incluyen un texto: el nombre del caso de uso. De este modo, podemos relacionar los casos de uso que aparecen en el diagrama con los casos de uso que aparecen en la documentación de los requisitos.

Para facilitar la comprensibilidad del diagrama, es muy conveniente evitar mezclar casos de uso de nivel de abstracción diferente en el mismo diagrama. Por ejemplo, en nuestro diagrama de ejemplo, todos los casos de uso (menos "Adjuntar archivo", del cual hablaremos más adelante) son del mismo nivel (usuario).

Si queremos representar gráficamente los casos de uso en los que se descompone otro caso de uso, siempre lo podemos hacer en un diagrama aparte.

2.1.4. Relaciones entre casos de uso y actores

Las líneas que vemos entre los casos de uso y los actores indican que existe alguna relación entre éstos. Un actor puede estar relacionado con un caso de uso porque sea el actor principal o porque sea un actor de apoyo.

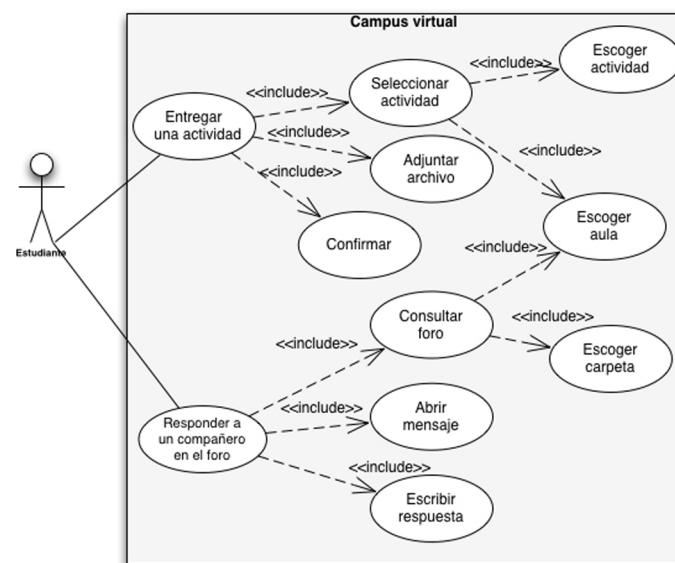
En nuestro ejemplo, podemos ver que cualquier miembro del aula (y esto incluye a los estudiantes y a los profesores) puede consultar el calendario de entregas, que los estudiantes pueden entregar una actividad o entregar una versión nueva de una actividad y que los profesores pueden corregir una actividad o publicar un enunciado.

2.1.5. Relaciones entre casos de uso: inclusión

En el diagrama de ejemplo podemos ver una relación de inclusión entre los casos de uso "Entregar una actividad" y "Adjuntar archivo" y otra entre "Publicar enunciado" y, de nuevo, "Adjuntar archivo". La manera de leer esta relación es, por ejemplo, que "Entregar una actividad" incluye "Adjuntar archivo", tal como indica el sentido de la flecha.

Como ya hemos visto, las relaciones de inclusión se dan a menudo entre casos de uso de diferente nivel. Teniendo en cuenta lo que hemos dicho anteriormente sobre mezclar casos de uso de diferente nivel en el mismo diagrama, es discutible la conveniencia de mostrar el caso de uso en el mismo diagrama que el resto, puesto que la presencia del caso de uso en el diagrama puede confundir, más que ayudar, a sus destinatarios.

Un error muy habitual es intentar representar el flujo de acontecimientos del caso de uso utilizando relaciones de inclusión para descomponerlo en objetivos más concretos:

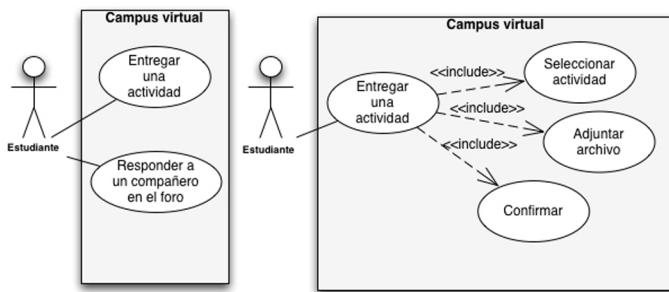


El problema con este tipo de descomposición es que se corre el riesgo de intentar sustituir la descripción textual, algo que no se puede hacer, puesto que es más difícil de leer y entender y, a la vez, no aporta toda la información que contiene la descripción. Por ejemplo, no tenemos ninguna manera de indicar la secuencia de los acontecimientos ni de relacionar el escenario principal con las extensiones: ¿primero adjuntamos un archivo y después seleccionamos la actividad o al revés?

Lo primero que hay que tener en cuenta es que el diagrama de casos de uso no muestra la secuencia de acontecimientos de un caso de uso; la distribución en el espacio de los casos de uso no tiene ningún significado y no implica, por lo tanto, que un caso de uso vaya antes que el otro.

Además, para asegurarnos de que los diagramas son fáciles de entender, no deberíamos tener grandes diferencias en cuanto al nivel de sus objetivos. Hay que evitar, pues, casos como el de la muestra, en el que tenemos, por ejemplo, un caso de uso de nivel de usuario "Entregar una actividad" y los casos de uso a escala de subtarea "Elegir aula" y "Elegir actividad".

En todo caso, si lo consideramos necesario, siempre podemos usar más de un diagrama y agrupar así los casos de uso según el criterio que consideremos más adecuado. Por otro lado, también hay que tener en cuenta que un caso de uso puede aparecer en más de un diagrama sin que esto implique ninguna duplicidad, puesto que la descripción textual sólo estará en un lugar.



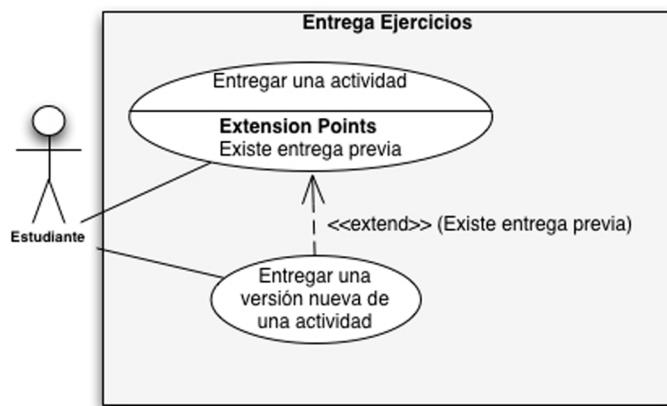
En este caso hemos creado dos diagramas. En el primero, hemos agrupado los casos de uso por el nivel de su objetivo mostrando sólo los de nivel usuario. En el segundo, en cambio, nos centramos en la descomposición de un caso de uso y mostramos, por lo tanto, el caso de uso y los de nivel inmediatamente inferior al suyo que están relacionados con él. Podríamos hacer un tercer diagrama que mostrara la descomposición de "Responder a un compañero del foro", otro para "Seleccionar actividad", etc.

2.1.6. Relaciones entre casos de uso: extensión

En el ejemplo inicial podemos encontrar un ejemplo de extensión entre los casos de uso "Entregar una actividad" y "Entregar una versión nueva de una actividad". En este caso, el sentido de la flecha nos indica que "Entregar una versión nueva de una actividad" es una extensión de "Entregar una actividad".

Como hemos visto antes, esta relación se usa poco. Habitualmente, en lugar de crear un nuevo caso de uso "Entregar una nueva versión de una actividad" habríamos añadido esta extensión al caso de uso "Entregar una actividad" modificando, por lo tanto, la descripción textual.

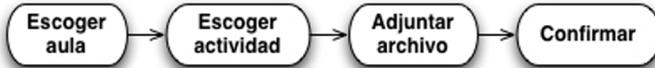
Opcionalmente, UML permite indicar el llamado *punto de extensión*, que es un texto que identifica en qué punto del escenario principal se produce el acontecimiento que provoca el comportamiento alternativo: de este modo podemos identificar el punto donde se inicia el nuevo comportamiento sin necesidad de hacer referencia a un número de paso concreto, lo que facilita el mantenimiento del modelo de requisitos. El problema, no obstante, es que se hace necesario identificar, a priori, todos los posibles puntos de extensión del caso de uso y, por ello, es una práctica en desuso.



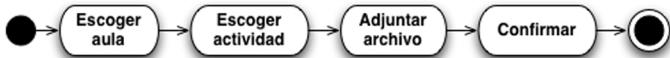
2.2. El diagrama de actividades

El diagrama de actividades combina diferentes ideas del mundo de la modelización de procesos y, por lo tanto, nos puede ayudar a documentar el comportamiento del sistema si lo vemos como un proceso. La modelización de actividades enfatiza más la secuencia y la coordinación de las posibles acciones por documentar que quién es el responsable de ejecutarlas.

El elemento básico de este diagrama es la actividad, que representa el hecho de que alguien hace a alguien. El otro elemento básico son las transiciones o flujos, que representan el hecho de que se pasa de una actividad a la siguiente. A continuación, tenemos un diagrama de actividades muy sencillo para el caso de uso "Entregar una actividad".



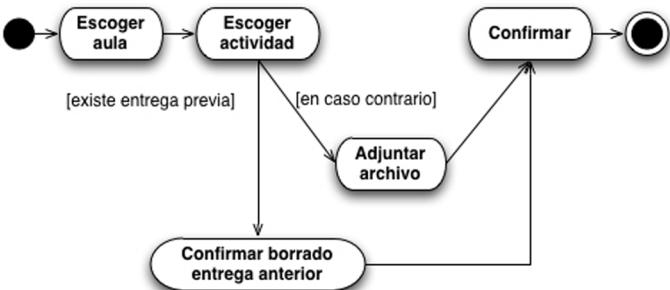
En este caso, cada paso del caso de uso lo hemos representado como una actividad y las diferentes transiciones nos dan una idea sobre cuál es la secuencia en la que tienen lugar las diferentes actividades. Se suele facilitar la lectura de la secuencia, indicando el inicio y el final:



En este ejemplo podemos ver fácilmente que el proceso empieza por "Elegir aula", tal como indica el punto negro, y el final se produce después de "Confirmar", tal como indica la transición hacia el punto negro dentro de un círculo. A pesar de que, en este caso, sólo hay uno, podemos poner en nuestro diagrama tantos símbolos de final como queramos.

2.2.1. Lógica condicional

Una de las ventajas del diagrama de actividades es que podemos representar la lógica condicional que nos lleva a la ejecución de los diferentes escenarios. Así pues, continuando con el caso anterior, podríamos hacer el diagrama siguiente:



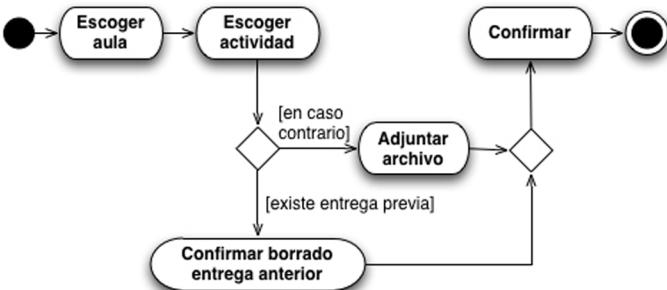
En este diagrama hemos introducido un nuevo elemento de notación, la condición, también llamada *condición de guarda*, que se representa mediante el texto entre corchetes en la transición. La condición *existe entrega previa* nos indica que esta transición sólo se produce cuando la condición es cierta. Por lo tanto, "Elegir actividad" tiene dos transiciones posibles: una se produce cuando hay una entrega previa y la otra cuando no es así.

Si queremos hacer más explícito el punto en el que hay una toma de decisión, lo podemos representar mediante el elemento gráfico de decisión, que se presenta mediante un rombo:



La decisión tiene un flujo de entrada y varios flujos de salida, cada uno con una condición. Sólo uno de los flujos de salida (aquel para el que la condición se cumpla) será el que se tome en un escenario concreto. Por ello, las diferentes condiciones deberían ser mutuamente excluyentes.

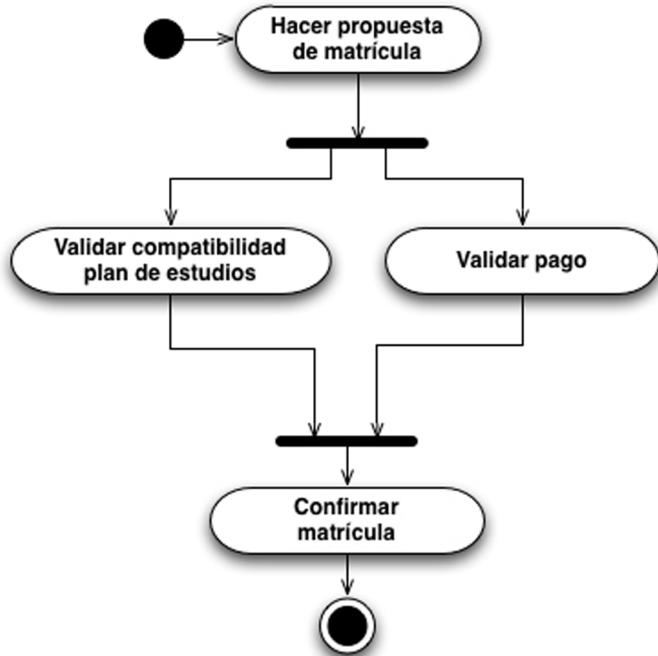
Otro elemento condicional que podemos utilizar es la fusión (Merge), que se representa como un rombo (igual que la decisión) pero que tiene varios flujos de entrada y sólo uno de salida, que se tomará cuando se llegue por alguno de los flujos de entrada:



Como hemos visto, las decisiones y fusiones son elementos opcionales que nos permiten enfatizar la lógica condicional. Habitualmente sólo usaremos, con este propósito, las decisiones.

2.2.2. Paralelización

Con los elementos de lógica condicional, los diferentes caminos son excluyentes: o bien se toma uno o bien se toma el otro, pero siempre hay un único camino (y, por lo tanto, una única actividad) ejecutándose en un momento concreto. A veces, no obstante, nos puede interesar tener la posibilidad de indicar que dos caminos se ejecutan de manera paralela:



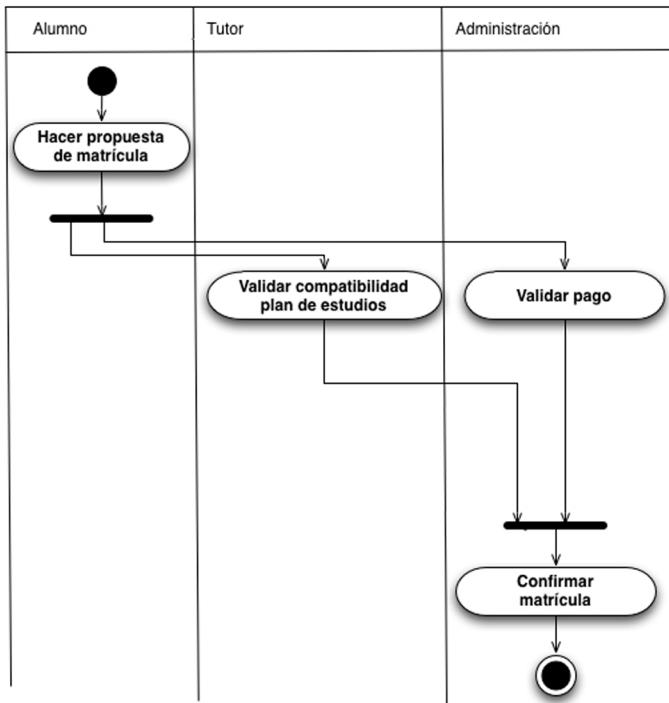
En este ejemplo, una vez se ha hecho la propuesta de matrícula, se inician dos flujos paralelos: uno que comprueba la compatibilidad con el plan de estudios y otro que comprueba que el pago se haga correctamente. Los dos se ejecutan en paralelo, puesto que no hay que esperar a que uno acabe para empezar el otro. La primera línea gruesa del diagrama es un ejemplo de bifurcación (*fork*); cada bifurcación tiene un flujo de entrada y varios flujos de salida.

El elemento contrario a la bifurcación es la unión (*join*). Este elemento (la otra barra gruesa) tiene varios flujos de entrada y uno de salida, que, a diferencia de la fusión, no se ejecuta hasta que no se han ejecutado todos los flujos de entrada.

Así pues, en nuestro ejemplo, no se confirma la matrícula hasta que no se ha validado la compatibilidad con el plan de estudios y se ha validado el pago.

2.2.3. Organización: carriles

Finalmente, hay un elemento de notación que nos puede ayudar a organizar las actividades. Los carriles (*swimlanes*) se usan, típicamente, para indicar qué actividades hace cada uno de los actores del proceso (o, en nuestro caso, del caso de uso) que estamos documentando. Se indican, gráficamente, de la manera siguiente:



En este caso, vemos que el alumno es el responsable de hacer la propuesta de matrícula, el tutor de validar la compatibilidad con el plan de estudios y el departamento de administración de validar el pago y confirmar la matrícula.

2.2.4. Otros elementos de notación

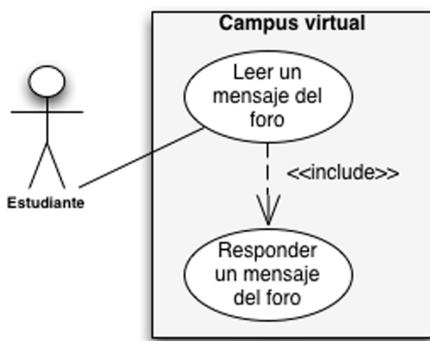
El diagrama de actividades soporta otros elementos de notación (como, por ejemplo, las señales o las subactividades) que no veremos en este módulo, puesto que sirven para modelizar comportamientos más complejos que los que encontramos típicamente en un caso de uso.

2.3. El modelo resultante

El modelo de casos de uso resultante debe contener una especificación basada en casos de uso especificados textualmente, tal como se explicó en el módulo "Requisitos". Ésta se complementará con los diagramas UML que sean oportunos para ayudar a comprender el modelo.

A continuación, veremos un ejemplo del modelo de casos de uso resultante para el ejemplo de las universidades. No obstante, por brevedad, nos centramos en sólo dos casos de uso del sistema.

El diagrama de casos de uso muestra visualmente los actores, los casos de uso, las relaciones de generalización/especialización entre casos de uso, qué actores intervienen en cada caso de uso y, si las hay, las relaciones entre casos de uso. En nuestro reducido ejemplo, mostraría los dos casos de uso:



Sin embargo, como se ha comentado, los diagramas de casos de uso sólo complementan la especificación textual de estos, que también debe formar parte del modelo de casos de uso.

Caso de uso: leer un mensaje del foro.

Actor principal: usuario.

Ámbito: campus virtual.

Nivel de objetivo: usuario.

Stakeholders e intereses:

Usuario: quiere leer el mensaje.

Profesor responsable del aula: quiere leer el mensaje y saber qué estudiantes lo han leído.

Precondición: el usuario se debe haber identificado en el sistema.

Garantías mínimas: el sistema grabará el intento de lectura del mensaje.

Garantías en caso de éxito: el sistema mostrará al usuario el contenido del mensaje y registrará su lectura.

Escenario principal de éxito:

1. El usuario indica qué mensaje quiere leer.
2. El sistema registra el intento de lectura del mensaje por parte del usuario.
3. El sistema valida que el usuario tenga acceso al foro.
4. El sistema muestra el tema y el contenido del mensaje.
5. El sistema registra que el usuario ha leído el mensaje.

Extensiones:

- 1a. El usuario apaga el ordenador.
 - 1a1. El sistema registra la lectura del mensaje aunque el usuario quizás no lo haya visto.
- 5a. El usuario quiere escribir una respuesta a un mensaje.

El usuario **responde al mensaje en el foro**

 - 5b. La base de datos no está disponible (error de servidor).
 - 5b1. El sistema indica al usuario que no ha sido posible recuperar el mensaje y le pide que lo vuelva a intentar pasado un rato.
 - 5c. El usuario quiere descargar documentos adjuntos.
 - 5c1. El usuario indica qué documento adjunto quiere descargar.
 - 5c2. El sistema descarga el documento adjunto en el ordenador del usuario.
 - 5c3. Si el usuario quiere descargar más documentos, volvemos al paso 5c1.

Caso de uso: responder a un mensaje del foro.

Actor principal: usuario.

Ámbito: campus virtual.

Nivel de objetivo: usuario.

Stakeholders e intereses:

Usuario: quiere responder a un mensaje que ha leído.

Profesor responsable del aula: quiere hacer el seguimiento de los debates que se producen en su aula.

Precondición: el usuario se debe haber identificado en el sistema.

Garantías mínimas: el sistema registrará la respuesta al mensaje.

Garantías en caso de éxito: el sistema registrará la respuesta al mensaje.

Escenario principal de éxito:

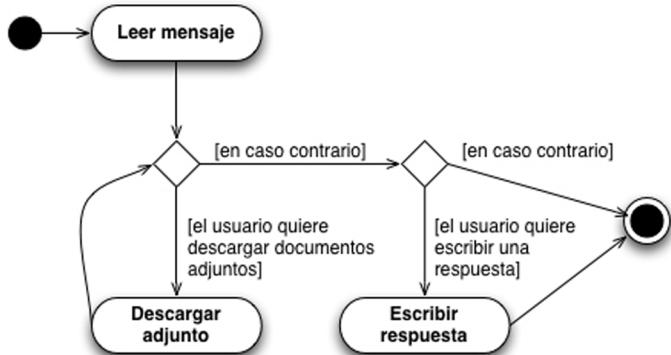
1. El usuario indica que quiere escribir una respuesta a un mensaje.
2. El sistema pide el tema y el contenido de la respuesta, sugiriendo *RE:<Tema del mensaje al que respondemos>* como tema del mensaje.
3. El usuario modifica el tema si quiere e introduce el contenido de la respuesta.
4. El sistema registra la respuesta con el tema y contenido introducidos, asigna la fecha actual como fecha de publicación del mensaje y la asocia al mensaje original.

Extensiones:

- 3a. El usuario introduce un tema o una respuesta vacíos.
 - 3a1. El sistema indica que no se puede dejar vacío el tema ni la respuesta y volvemos al punto 3.

En algunos casos, nos puede interesar documentar uno o más casos de uso vistos como procesos. Los diagramas de actividades nos permitirán mostrar visualmente la secuencia de pasos que hacen los actores, la lógica condicional,

el paralelismo, si lo hay, e incluso cómo diferentes actores intervienen en un mismo caso de uso (en forma de carriles). En nuestro ejemplo podríamos documentar los dos casos de uso mediante un diagrama de actividades:



Sin embargo, en realidad sólo vale la pena crear un diagrama de actividades cuando se quiere documentar como proceso un caso de uso (o un conjunto de casos de uso) más complejo que los de este ejemplo. De hecho, la mayoría de los casos de uso de nivel de usuario son lo suficientemente simples como para no necesitar ningún diagrama que los complemente. Otros casos de uso, no obstante, especialmente algunos de nivel de organización, pueden ser más complejos y hacerse más comprensibles si los complementamos con un diagrama de actividades.

3. Modelización de la interfaz

El modelo de casos de uso que hemos visto hasta ahora se centra en recoger los objetivos de los actores hacia el sistema. Por este motivo, no hemos incluido, en ningún momento, detalles sobre la interfaz de usuario (no queremos que los detalles de la interfaz de usuario nos distraigan de lo que nos interesa: los objetivos de los actores).

Por otro lado, tenemos toda una serie de requisitos, como por ejemplo la usabilidad o la arquitectura de información, que al ser los casos de uso independientes de la interfaz de usuario, no quedan recogidos en este modelo.

Necesitamos, por lo tanto, otro modelo que nos documente, antes de diseñar la interfaz de usuario, estos requisitos, y que nos ayude a entender cómo los diferentes usuarios del sistema interactuarán con él.

Este modelo de la interfaz de usuario nos debe dar indicaciones sobre lo siguiente:

- De qué manera el sistema presenta la información a los usuarios (mediante una interfaz gráfica, un sistema de voz, etc.).
- Cómo navega el usuario a través de la información que le muestra el sistema para conseguir sus objetivos.
- Cómo se relaciona el comportamiento indicado en el modelo de casos de uso con la interfaz de usuario.

3.1. Casos de uso concretos

Se denominan **casos de uso esenciales** los casos de uso que describen la interacción entre actores y sistema de manera independiente de la tecnología y de la implementación. En contraposición, denominaremos **casos de uso concretos** a aquellos que tienen en cuenta la tecnología y la implementación.

Así pues, a partir de un mismo modelo de casos de uso esenciales, es posible llegar a varios modelos de casos de uso concretos, según cuál sea la tecnología que utilizamos para la interfaz con los usuarios y cuál sea el intercambio de información entre usuarios y sistema. En adelante, supondremos que el sistema que vamos a desarrollar se comunica con sus usuarios mediante una interfaz gráfica basada en pantallas.

Para decidir el contenido de las diferentes pantallas se ha de tener en cuenta la opinión de varios especialistas. A continuación describimos algunos:

- **Especialista en interacción.** Su finalidad es conseguir que los usuarios del sistema consigan cumplir sus objetivos. Para hacerlo, estudia las diferentes tareas para conseguir un objetivo concreto (por ejemplo, qué hay que hacer para matricular un alumno en la universidad) y el modo como se llevan a cabo.
- **Especialista en usabilidad.** Aplica el método científico al estudio y la observación del modo como los usuarios utilizan el sistema para hacer que el uso sea cómodo e intuitivo.
- **Arquitecto de información.** Se encarga de organizar la información de manera que sea fácil de encontrar y de utilizar. Por ejemplo, es el encargado de asegurarse de que los estudiantes puedan acceder rápidamente al foro de su aula y de que no tengan que navegar por un montón de pantallas antes de llegar a él.

Así pues, podemos escribir una versión concreta (sin tener en cuenta las posibles extensiones) del caso de uso "Leer un mensaje del foro".

Caso de uso: leer un mensaje del foro.

Actor principal: usuario.

Escenario principal de éxito:

1. El sistema muestra la pantalla *Lista de mensajes* con los mensajes de la carpeta *Recibidos* del foro.
2. El usuario selecciona un mensaje de la lista.
3. El sistema registra el intento de lectura del mensaje por parte del usuario.
4. El sistema valida que el usuario tenga acceso al foro.
5. El sistema muestra la pantalla *Leer mensaje*.
6. El sistema registra que el usuario ha leído el mensaje.

Extensiones:

(...)

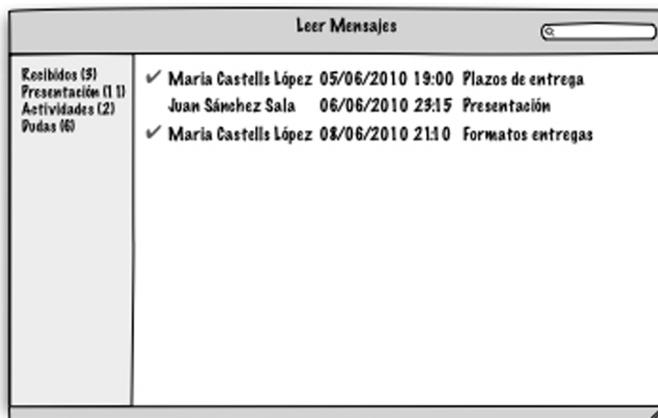
Pantallas:

Pantalla	Datos mostrados	Datos introducidos
Lista de mensajes	Lista de mensajes. Por cada mensaje: Marca leído/no leído Nombre autor Fecha y hora Tema	Mensaje seleccionado
Leer mensaje	Tema del mensaje Contenido del mensaje Autor Fecha	

3.2. Modelos de las pantallas

Una técnica muy útil en esta fase es crear modelos de las pantallas. Los modelos de las pantallas nos ayudan a hacernos una idea de cómo será la interfaz gráfica de usuario final sin necesidad de preocuparnos de los detalles concretos del diseño gráfico (colores, tipos de letra, etc.).

Los modelos de las pantallas se centran en los conceptos por mostrar y en la estructura de la pantalla más que en cómo será finalmente. Por ejemplo, no se tienen cuenta los colores ni los elementos exactos de interfaz gráfica que se usarán. Por ello muchas veces se hacen, incluso, manualmente (o simulando un dibujo a mano alzada).



Estos modelos nos ayudan a explorar posibles modelos de interacción y de organización de la información y también a detectar requisitos sobre qué información hay que mostrar en cada pantalla.

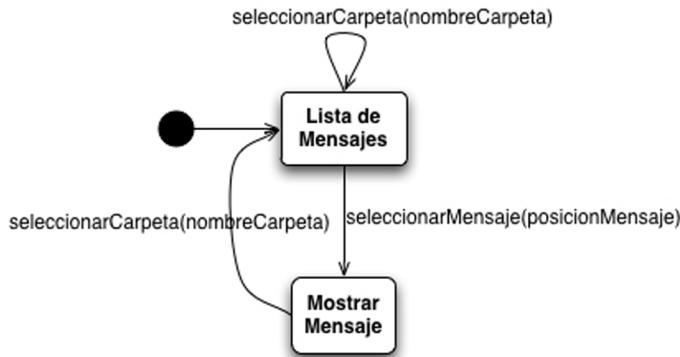
Así pues, el objetivo de los modelos de las pantallas es dejar claro:

- qué información se muestra,
- la distribución de la información en la pantalla,
- qué acciones puede realizar el usuario a partir de la información mostrada,
- cuál es el proceso que sigue el usuario para completar el caso de uso.

3.2.1. Diagrama de estado de las pantallas (mapa navegacional)

El mapa navegacional es un modelo que nos proporciona información sobre cuál es el flujo entre pantallas que puede seguir el usuario. Así pues, la finalidad del mapa navegacional es dar una visión general de qué acciones se pueden hacer en cada pantalla y en qué casos se pasa de una pantalla a otra.

Para hacer el modelo del mapa navegacional podemos usar una versión muy simplificada del diagrama de estados UML:



El diagrama de estados UML es muy similar al de actividades y, por lo tanto, es muy fácil entender su notación. Los estados representan las diferentes pantallas (y el punto negro indica el estado inicial) y las flechas representan acontecimientos ante los cuales el sistema ha de reaccionar.

En el diagrama del ejemplo, hay dos pantallas (*Lista de Mensajes* y *Mostrar Mensaje*) y tres acontecimientos:

- seleccionarCarpeta (nombreCarpeta) en Lista de Mensajes
- seleccionarMensaje (posicionMensaje) en Mostrar Mensaje
- seleccionarCarpeta (nombreCarpeta) en Lista de Mensajes

Cómo podemos ver, el mapa navegacional nos da una visión muy resumida de la interacción entre el usuario y el sistema, puesto que nos dice qué información aporta el usuario al sistema (el nombre de la carpeta o la posición del mensaje).

3.3. Contratos de las operaciones del sistema

En el supuesto de que necesitemos elaborar una documentación formal de los requisitos del sistema, los modelos vistos hasta ahora no serán suficientes, puesto que no formalizan cuál es el comportamiento del sistema ante cada acontecimiento; sólo indican qué transición de pantallas se produce.

Si necesitamos este grado de formalismo, podemos usar operaciones o acontecimientos de sistema. Lo que hacemos en este caso es identificar operaciones a partir de los acontecimientos del mapa navegacional y escribir, en un lenguaje más o menos formal, un contrato que establezca cuál es el comportamiento del sistema cuando se llama esta operación.

Un contrato consta de tres partes:

- **La firma.** Nos dice el nombre de la operación, la información que recibe de aquel que la solicita (los parámetros de entrada) y la información que

se devuelve a quien ha solicitado la operación (los parámetros de salida o el resultado).

- **Las precondiciones.** Las obligaciones que debe cumplir aquel que solicita la operación para que el sistema la ejecute correctamente. En el supuesto de que estas precondiciones no se cumplan, supondremos que el sistema rechaza el acontecimiento y que no se produce ningún cambio.
- **Las poscondiciones.** Las obligaciones a las que se compromete el sistema cuando se invoca la operación. El sistema se compromete a que sean ciertas estas condiciones una vez ejecutada la operación.

Ejemplo de contrato

```
obtenerNombreMensajesCarpeta(nombreCarpeta:String):Integer
```

pre: el nombre de la carpeta se corresponde a una carpeta del foro

pos: el resultado es el número de mensajes que hay en la carpeta con independencia de si se han leído o no

Este sencillo ejemplo nos permite ver cómo el contrato documenta:

- el nombre de la operación (obtenerNombreMensajesCarpeta).
- la información que aporta el usuario (nombreCarpeta de tipo String, es decir, un texto).
- la información que devolverá el sistema (en este caso, un número entero).
- las obligaciones del usuario (dar un nombre de carpeta que exista).
- las obligaciones del sistema (calcular el número de mensajes de la carpeta y devolverlo).

Como ya hemos mencionado, podemos escribir los contratos usando un lenguaje más formal, como por ejemplo OCL¹, el lenguaje estándar de restricciones de UML.

⁽¹⁾OCL son las siglas de *object constraint language*, en castellano, lenguaje de objetos de restricciones.

Ejemplo de contrato en OCL

```
contexto System::obtenerNombreMensajesCarpeta(nombreCarpeta:String):Integer
```

pre: Carpeta.allInstances()->exists(c|c.nombre=nombreCarpeta)

pos: result = Carpeta.allInstances()->select(c|c.nombre=nombreCarpeta).mensaje->size()

4. Modelo del dominio

Un modelo del dominio es una representación de clases conceptuales del mundo real en un dominio de interés. Este modelo ha de servir a las personas que intervienen en el desarrollo del software para entender mejor el dominio del sistema que deben desarrollar.

Para documentar un modelo del dominio en UML se usarán uno o más diagramas de clases, mediante los cuales se representarán las clases conceptuales (o clases del dominio), sus atributos y sus asociaciones.

4.1. Convenciones en los diagramas UML

Antes de empezar con la descripción de los diagramas del modelo conceptual, estableceremos algunas convenciones que usaremos con respecto al nombre de clases, atributos y asociaciones.

UML no pone ninguna restricción respecto a qué nombres son válidos y cuáles no, pero normalmente preferimos usar nombres que más adelante, cuando se pase a hacer tareas de diseño y de programación, se puedan conservar. Por otro lado, seguir una determinada convención de nombres ayuda a dar coherencia al conjunto de artefactos que elaboramos durante el desarrollo de software.

Por esta razón, en estos materiales seguiremos las convenciones siguientes:

- Usaremos nombres en minúsculas en general. Para las clases, las asociaciones y los tipos de atributo, la primera letra estará en mayúscula, como si se tratara de un nombre propio.

Ejemplo

No usaremos nombres de clase como *ALUMNO* o *alumno* sino *Alumno*. Los nombres de asociación serán como *Graba* o *SeMatricula* y los de los tipos de atributo, como *Integer* o *Boolean*.

Los nombres de atributo se escribirán en minúsculas, como nombre o apellidos.

- No usaremos nombres que contengan espacios; separaremos las palabras poniendo en mayúscula la primera letra de cada palabra.

Ejemplo

Como nombre de atributo usaremos, por ejemplo, *FechaNacimiento* en lugar de *fecha de nacimiento*.

- No usaremos caracteres fuera del rango de ASCII, como los signos de puntuación, los acentos, la ç, etc.

Ejemplo

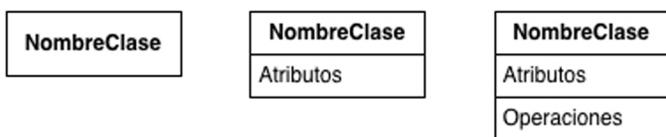
En lugar de *fecha de inscripción* usaremos *fechaInscripcion*.

4.2. Clases

4.2.1. Notación UML

En UML una clase se indica mediante una caja con tres compartimentos, de los que dos son opcionales.

Representación de las clases en el lenguaje UML



El primer compartimento es para el nombre de la clase y es el único compartimento obligatorio. El segundo compartimento contiene los atributos de la clase y, a pesar de que es opcional, lo usaremos en los modelos del dominio. El tercer compartimento contiene las operaciones y también es opcional; sin embargo, en este caso no lo usaremos en los modelos del dominio.

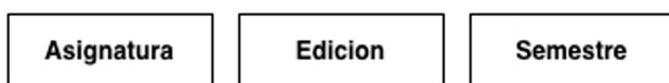
4.2.2. Técnicas de modelización

Identificación de clases del dominio

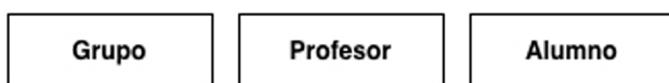
Para identificar clases del dominio hay que elaborar una lista de aquellos conceptos del espacio del problema que son relevantes para el sistema que se analiza.

Ejemplo de identificación de clases del dominio

En el ejemplo de partida de la universidad, nos dicen que la universidad ofrece una serie de asignaturas de las que se imparte una edición cada semestre.



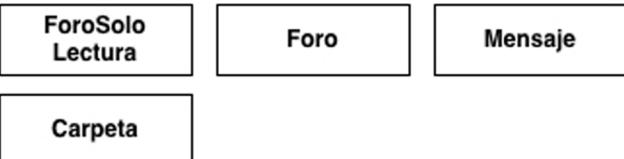
También sabemos que para cada edición de una asignatura se crean grupos y que cada uno tiene un profesor y una serie de alumnos matriculados.



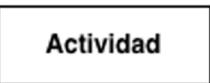
Algunos cursos son presenciales. De éstos queremos saber el horario semanal y, para cada franja horaria del curso, el aula en la que se imparten. Las aulas están situadas en edificios que, a la vez, pertenecen a un determinado campus.



Hay un campus virtual en el que cada grupo dispone de un tablero y un foro que contienen mensajes agrupados por carpetas.



Los profesores proponen actividades.



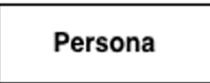
Una manera de completar la lista inicial de clases del dominio es la técnica, usada por varios autores, de usar categorías de clases conceptuales. Consiste en repasar una lista de categorías típicas de clases para identificar clases de aquella categoría que nos pueden haber pasado por alto.

Coad, Lefebvre y Luca (1999) proponen la siguiente lista de categorías:

- **Participantes, lugares y cosas.** Clases que representan personas u organizaciones del dominio (participantes), lugares y cosas. Los almacenes de los que dispone una empresa, los productos que vende o las personas que trabajan en ella o que le compran materiales, por ejemplo, pertenecen a esta categoría.

Ejemplo

En el ejemplo de las universidades, las personas que intervienen en ella (estudiantes, profesores y cualquier otro personal implicado) son ejemplos claros de participantes.



Por otro lado, aulas, edificios y campus son ejemplos claros de lugares.

- **Roles.** Clases que representan los roles que pueden tener las personas o las organizaciones en las actividades que se desarrollan en el dominio analizado. Los roles de comprador o vendedor que pueden tener las personas en muchos negocios son un buen ejemplo de esto.

Ejemplo

En nuestro ejemplo, los dos roles más importantes son el alumno y el profesor, que ya habíamos identificado.

- **Momentos o intervalos.** Clases que representan momentos en el tiempo o intervalos de tiempos, como un alquiler, una venta, etc. A menudo, estos momentos o intervalos tienen un conjunto de partes.

Ejemplo

En el ejemplo de las universidades, las distintas ediciones de una asignatura –ya identificadas– son un ejemplo claro de intervalo.

- **Descripciones.** Clases que representan la descripción habitual de tipo catálogo de una cierta "cosa", normalmente un producto. Así, por ejemplo, distinguimos un coche concreto, que tiene su matrícula y número de bastidor, del modelo de coche, que es una descripción de tipo catálogo asociada a aquel coche concreto.

Ejemplo

La Asignatura se puede ver como una descripción de las distintas ediciones que se hacen de ella, dado que nos indica el número de créditos, que es el mismo para todas las ediciones de una asignatura.

Larman (2005) propone una lista de categorías menos genérica:

- **Transacciones, líneas de transacción y productos o servicios:** las transacciones, normalmente de negocio. Un ejemplo clásico son las ventas de un supermercado. Cada venta tiene un conjunto de líneas de venta en la que cada línea tiene un producto vendido, un número de unidades y un precio total de línea.
- **Acontecimientos importantes,** a menudo en una fecha o lugar que es relevante, como un vuelo o el pase de una película.
- **Catálogos y descripciones de cosas:** las descripciones, en el sentido de Coad, se pueden agrupar en catálogos.
- **Contenedores físicos o lógicos y los elementos que contienen,** como almacenes y los productos que guardamos en ellos, o un tablero de ajedrez y las casillas que éste contiene.

Ejemplo

En nuestro ejemplo, el campus y los edificios son contenedores físicos de otros lugares.

- **Sistemas externos** con los que interactuará el sistema que estamos analizando.

- **Grabaciones** de trabajos, contratos, asuntos legales, etc., como los recibos.

Ejemplo

En nuestro ejemplo, los estudiantes pagarán la matrícula y tendremos un recibo de este pago.

ReciboMatricula

- Instrumentos financieros, como un cheque, efectivo, etc.
- Agendas, manuales, documentos, etc.

Ejemplo

En el ejemplo, el horario de los cursos presenciales, que ya hemos identificado como una clase, se puede ver como una agenda.

La lista de categorías de Larman incluye, además, una serie de categorías muy coincidentes con la lista de Coad: lugares, objetos físicos y roles de organizaciones y personas.

Otra técnica útil para identificar clases del dominio consiste en reutilizar modelos existentes. A pesar de que no podremos reutilizar tal cual el análisis realizado para un sistema existente, dado que, si pudiéramos hacerlo, probablemente, no necesitaríamos desarrollar un nuevo sistema a medida, sino que podríamos reutilizar directamente el sistema existente, sí podemos usar un análisis existente como punto de partida para identificar clases en nuestro modelo.

En este sentido, la obra de Coad y otros (1999) propone una serie de modelos genéricos que pueden ser útiles como punto de partida, mientras que Fowler (1997) propone usar el concepto de patrón para documentar patrones de análisis que podemos aplicar adaptándolos al dominio concreto que estamos analizando.

Ved también

Podéis encontrar más información sobre los patrones aplicados a la ingeniería del software en el módulo "Introducción a la ingeniería del software".

Nomenclatura de las clases

A la hora de dar nombre a las clases del dominio que se identifican conviene tener en cuenta la denominada **regla del cartógrafo**. Se trata de lo siguiente:

- Utilizar la terminología que usan las personas y los expertos del dominio que se está modelizando, al igual que un cartógrafo usa la toponimia local en el lugar que está cartografiando.

Ejemplo

En el ejemplo hemos usado un nombre de clase bastante extraño, que no tiene nada que ver con el que usa la gente que trabaja y estudia en la universidad de ejemplo: ForoSoloLectura. Conviene, pues, que nos fijemos en la nomenclatura que usa la gente que estudia y trabaja en la universidad para la que desarrollamos el proyecto. En este caso, los foros en los que sólo algunas personas autorizadas pueden escribir se denominan *Tablones*.



- Excluir aspectos irrelevantes del dominio que estamos analizando, al igual que los cartógrafos hacen una abstracción de la realidad y sólo representan aquellos rasgos del terreno que interesan en un mapa.

Ejemplo

En nuestro caso, hemos identificado una clase *ReciboMatricula* que no es relevante en el sistema que estamos analizando, dado que el alcance de este sistema no incluye el pago y la elaboración de recibos de las matrículas de los estudiantes.



- No añadir nada que no corresponda realmente al dominio que analizamos.

Error frecuente: clases de dominio, no de software

Hay que tener claro que el objetivo del modelo del dominio es documentar conceptos del mundo real, no clases de software. Por lo tanto, no tendremos clases que representen artefactos de software, como bases de datos, botones o ventanas.

Por la misma razón, existen ciertos conceptos de la orientación a objetos que no usaremos cuando realicemos el análisis. Así, por ejemplo, no asignaremos a las clases de este modelo responsabilidades de comportamiento y no les asignaremos operaciones. Tampoco usaremos el concepto de visibilidad.

4.3. Atributos

4.3.1. Notación UML

Dentro de la caja que representa una clase, en el compartimento de los atributos, cada atributo se indica textualmente escribiendo el nombre y, opcionalmente, dos puntos seguidos del tipo del atributo.

Ejemplos de atributos

Los siguientes son algunos ejemplos de atributos:

```
nombre: String
fechaNacimiento: Fecha13/01/2011 17:08:51
altura: Longitud
```

El lenguaje UML define una serie de tipos primitivos estándar que podemos emplear para indicar los tipos de los atributos. Pero, en general, podemos usar otros tipos de atributos siempre que quede bastante claro, por el nombre del tipo, a qué nos referimos.

UML permite, opcionalmente, indicar la multiplicidad de un atributo añadiendo, después del tipo y entre corchetes, un indicador de ésta, que puede ser:

- "0..1" para indicar que un atributo es univaluado pero opcional.
- "1" para indicar que un atributo es univaluado y obligatorio. Esta opción es la que consideraremos por defecto y, por lo tanto, no indicaremos esta multiplicidad de manera explícita.
- "*" para atributos multivaluados ("1..*" si como mínimo han de tener un valor). Una manera equivalente es indicar "0..*", pero dado que "*" es más corto, usaremos esta segunda manera.
- Otras multiplicidades, como "3..5" (entre 3 y 5 valores).

Ved también

En el subapartado 4.3.2 se da una lista de tipos de atributos típicos que nos pueden facilitar la identificación de atributos en las clases del dominio.

Multiplicidad de un atributo

La multiplicidad de un atributo indica qué cardinalidades son válidas para aquel atributo; es decir, qué restricción existe respecto al número de valores que puede tener un atributo.

Ejemplos de atributos teniendo en cuenta su multiplicidad

Los siguientes son algunos ejemplos de atributos teniendo en cuenta su multiplicidad:

- nombre: String
- fechaNacimiento: Fecha [0..1]
- telefono: Telefono[1..*]

En el primer caso, dado que no hemos indicado ninguna multiplicidad, entendemos que la multiplicidad es 1; es decir, que el nombre es obligatorio y univaluado.

4.3.2. Técnicas de modelización

Identificación de atributos

A medida que vamos identificando clases de dominio, les podemos ir añadiendo atributos que indiquen la información relevante de los objetos de esa clase. Para identificar de manera más completa los posibles atributos de una clase, puede ser interesante repasar una lista de los tipos de atributos más frecuentes:

- **Números:** a menudo nos conviene saber si se trata de números naturales (enteros positivos, que denominaremos *Nat*), enteros (*Int*) o números no enteros en general (*Real*).

- **Textos:** *strings* (cadenas de caracteres sin formato) y textos con formato.
- **Booleanos:** cierto o falso, sí o no.
- **Información temporal:** como Fecha (1 de enero de 1980), Hora (12:43), Momento (1 de enero de 1980 a las 12:43), Duración (63 minutos).
- **Cantidades:** como Importe (por ejemplo 23 €), Longitud (por ejemplo 23,5 m), Temperatura (15°), Masa, etc.
- **Otros valores sin entidad propia:** Color (como el color RGB 70, 0, 130), Coordenadas (por ejemplo 41°23'N 2°11'E), CódigoPostal, NumeroTelefono, etc.

Ejemplo de identificación de atributos

Después de hablar de los detalles con los *stakeholders* en nuestro ejemplo, hemos identificado atributos en cada clase.

Las personas tienen nombre, apellidos y DNI. Dado que habrá un campus virtual, nos dicen que también quieren tener un texto con un pequeño currículo y una foto, así como una dirección de correo electrónico de contacto.

De los profesores queremos conocer también la fecha de contratación y de los alumnos, la de nacimiento.

Persona	Profesor	Alumno
dni: String nombre: String apellidos: String cv: String foto: Imagen direccion: EMail	fechaContratacion: Fecha	fechaNacimiento: Fecha

Las asignaturas tienen un nombre y un número de créditos. Cada semestre tiene el año y el número de semestre en el que éstas se imparten (primavera u otoño). De las ediciones que se realizan de una asignatura cada semestre, de momento, no tenemos atributos.

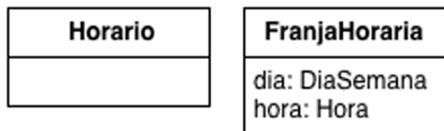
Asignatura	Semestre	Edición
nombre: String numCreditos: Nat	año: Año num: NumSemestre	

Una edición corresponde a la impartición de una asignatura en un semestre. Entonces, ¿por qué no ponemos los atributos *asignatura: Asignatura* y *semestre: Semestre* en la clase *Edición*? Lo veremos en el subapartado 4.4.

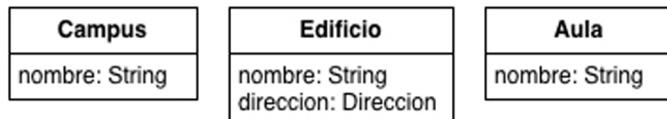
Los grupos de cada curso tienen un número.

Grupo
numero: Nat

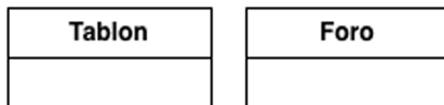
De los horarios no tenemos ningún atributo, pero de las franjas horarias que los forman, sí. En un horario, una franja horaria puede ser, por ejemplo, el lunes a las 15 horas o el miércoles a las 9 horas. Supondremos que si en un horario hay dos horas seguidas de clase, ello significa que hay dos franjas horarias.



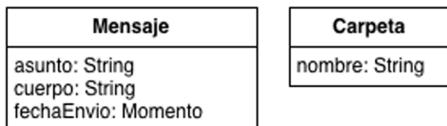
Las aulas, los edificios y el campus tienen un nombre (Aula C, edificio Shakespeare y campus Llull, por ejemplo). De los edificios querremos saber la dirección postal.



De tablones y foros no tenemos atributos.

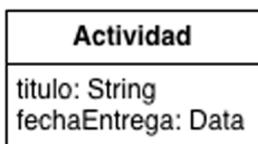


Los mensajes tienen un asunto, un cuerpo y una fecha y hora de envío. No tienen destinatario porque son mensajes enviados a un tablón o un foro y no mensajes de correo electrónico. Las carpetas en las que se organizan los mensajes tienen un nombre.



Para la fecha y hora de envío, en lugar de usar dos atributos separados, se ha decidido usar un atributo que represente el momento exacto (fecha y hora) en el que se envió el mensaje.

Las actividades de las asignaturas tendrán un título y una fecha de entrega:

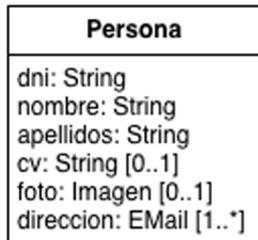


Atributos opcionales y atributos multivaluados

Para la mayoría de los atributos, habrá que señalar, tan sólo, si el atributo es **opcional** o no. Pero algunos atributos pueden ser, además, **multivaluados**.

Ejemplos de atributos opcionales y multivaluados

Supongamos, por ejemplo, que las personas pueden no indicar su currículum y pueden no tener fotografía. Y, además, que pueden tener múltiples direcciones de correo electrónico, pero que como mínimo deben tener una:



Por otro lado, podemos considerar que el cuerpo de un mensaje es opcional porque con el asunto, a veces, es suficiente.

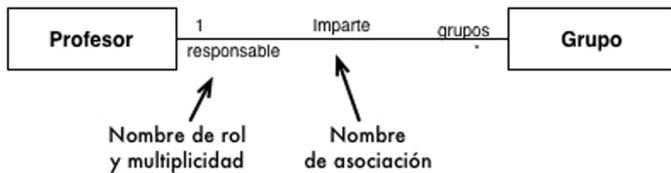
Mensaje
asunto: String
cuerpo: String [0..1]
fechaEnvio: Moment

4.4. Asociaciones

4.4.1. Notación UML

Una asociación, en UML, se indica mediante una línea continua entre las dos clases asociadas y puede incluir un nombre, nombres de rol y multiplicidades, entre otros elementos.

Notación UML de las asociaciones



Los nombres de rol de la asociación indican el rol que tiene cada clase en la asociación y, por lo tanto, es una manera de referirse a las instancias asociadas a una instancia desde ésta. Así, en el ejemplo de la figura, desde **Grupo**, el profesor asociado se denomina *responsable*, mientras que desde **Profesor** el conjunto de grupos que tiene asociados se denomina *grupos*.

La multiplicidad de los extremos de asociación se indica usando la misma nomenclatura que la de los atributos. A pesar de que UML considera, si no se indica lo contrario, que la multiplicidad por defecto es 1, en estos materiales siempre se indicará explícitamente la multiplicidad de cada extremo de asociación.

4.4.2. Técnicas de modelización

Nombres de asociación y de roles

Normalmente utilizaremos nombres de asociación que se puedan leer formando una frase del tipo *NombreClase-NombreAsociacion-NombreClase*, en el que el nombre de la asociación es un verbo que permite formar una frase con sentido.

Ejemplos de nombres de asociación

Algunos ejemplos de nombres de asociación y la frase que forman pueden ser:

- Alumno SeMatriculaDe Asignatura
- Profesor Imparte Grupo
- Estudiante Entrega Actividad

Para los nombres de los roles de asociación, en cambio, usamos nombres parecidos a los de los atributos.

Tanto el nombre de asociación como los de rol son totalmente opcionales y, de hecho, indicar el nombre de asociación y el de rol suele ser bastante redundante. Por esta razón, muchas veces no se indica el nombre de las asociaciones. En estos materiales, indicaremos un nombre de rol cuando sea especialmente interesante indicarlo.

En el caso de no indicar nombre de rol, UML asume que el nombre de rol es el nombre de la clase, pero empezando en minúscula. En nuestro caso, además, preferiremos pasarlo a plural si aquel extremo de la asociación es multivalorado, ya que los nombres de rol serán, así, más naturales.

Ejemplo de asociación con los nombres de rol implícitos

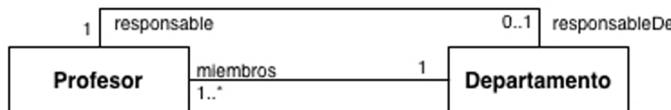


En este caso, los nombres de papel serían *profesor* y *grupos*.

No obstante, habrá que indicar, como mínimo, un nombre de rol, en aquellos casos en los que pueda haber confusión. Así, por ejemplo, será necesario indicar como mínimo un nombre de rol si tenemos dos asociaciones entre las dos mismas clases.

Ejemplo de dos asociaciones entre las mismas clases

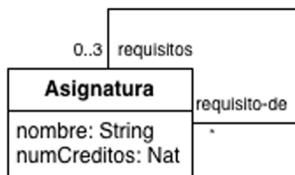
Imaginemos que queremos conocer el departamento al que pertenece un profesor y qué profesor es responsable de cada departamento. En este caso, debemos usar los nombres de rol para distinguir el profesor responsable de un departamento de aquellos que son simplemente miembros.



En el caso de las asociaciones recursivas, también es especialmente importante indicar claramente el rol de cada extremo de asociación.

Ejemplo de asociación recursiva

Supongamos, por ejemplo, que una asignatura puede tener requisitos: otras asignaturas que hay que haber cursado antes de cursar la asignatura en cuestión. Por ejemplo, una asignatura puede tener hasta tres requisitos, pero puede ser requisito de cualquier número de otras asignaturas.



Identificación de asociaciones

A medida que vamos identificando clases del dominio y poblándolas con sus atributos, también podemos ir identificando las asociaciones existentes entre éstas que, como siempre, sean relevantes para el sistema que analizamos.

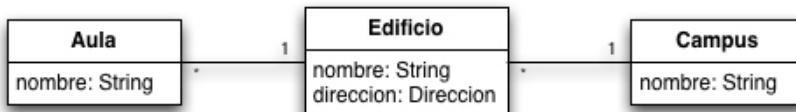
Identificaremos una asociación siempre que descubramos que en el espacio del problema las instancias de dos clases pueden estar asociadas, de tal manera que recordar la asociación entre éstas es relevante.

Para identificar las asociaciones en el modelo del dominio, un buen punto de partida es estudiar las clases ya identificadas y plantear qué pueden ser las asociaciones entre éstas.

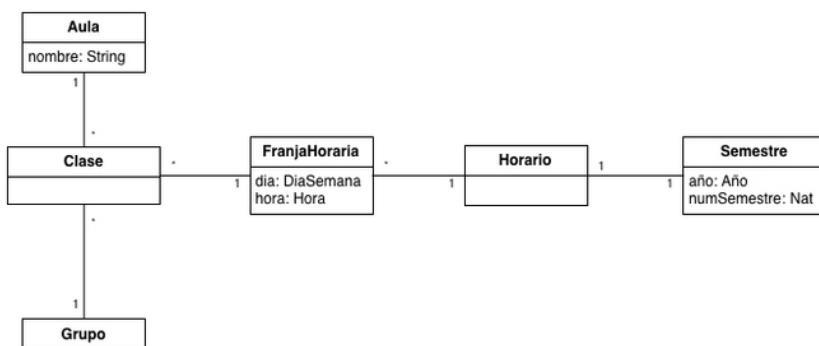
Ejemplo de identificación de asociaciones

Algunas asociaciones de nuestro caso de ejemplo se pueden identificar de entrada.

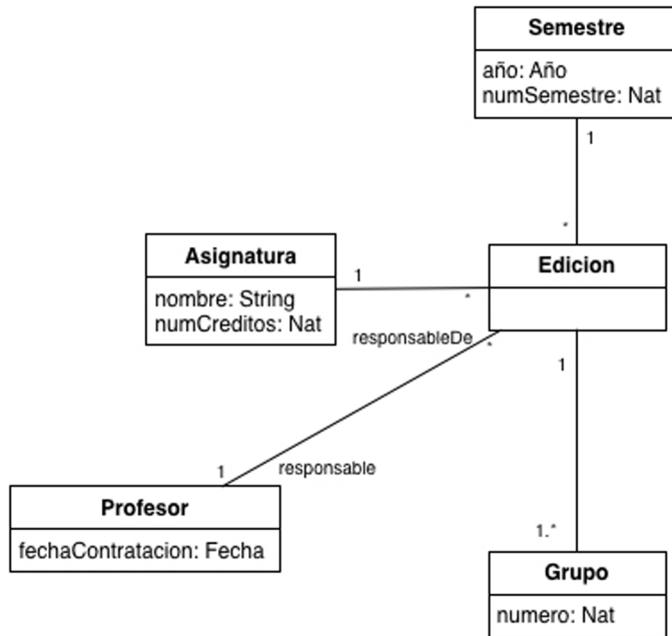
Cada aula está en un edificio que, a la vez, está en un campus. Así, por ejemplo, podríamos tener un aula de nombre *112*, situada en un edificio denominado *B* que, a su vez, está ubicado el campus *Llull*.



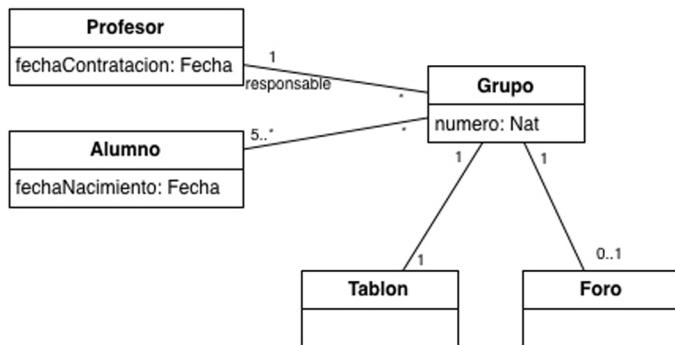
Cada semestre tiene un horario que está formado por un conjunto de franjas horarias. En cada franja horaria se programan cualquier número de clases. Así, por ejemplo, en un determinado horario, la franja horaria del lunes a las 9 horas podría tener programada una clase del grupo 2 de Ingeniería del software en el aula 112 antes mencionada. Dado que no la habíamos detectado antes, introducimos ahora la clase *Clase*, que representa una de estas clases programadas y que tiene asociadas el aula y el grupo que se han programado.



Cada edición de una asignatura corresponde a un semestre, tiene un profesor responsable y uno o más grupos.



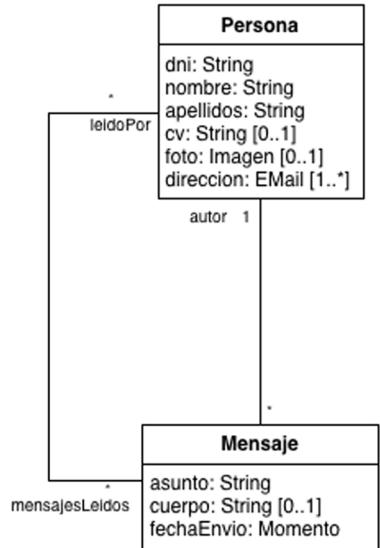
Cada grupo tiene un profesor y algunos alumnos. Supongamos, por ejemplo, que en un grupo debe haber, como mínimo, cinco alumnos. A cada grupo asignamos, también, un tablón y, opcionalmente, un foro.



Cada edición de una asignatura tiene una serie de actividades:



De cada mensaje querremos saber el autor y quién lo ha leído y quién no:

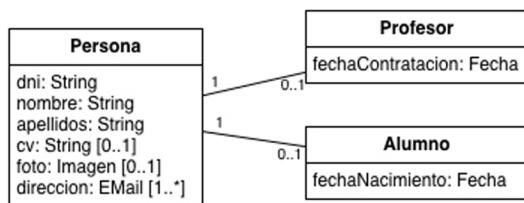


A partir de aquí, se puede mejorar la identificación usando, como ya se ha mencionado en el caso de las clases, listas de categorías de asociación.

La lista de categorías de clases del dominio de Coad, Lefebvre y Luca (1999) propone las asociaciones típicas siguientes:

- Los participantes, cosas o lugares pueden tener asociada una descripción.
- Cada participante, pero también las cosas o los lugares, puede tener asociados cualquier número de roles, en los que cada instancia de rol corresponde a un único participante.

En nuestro ejemplo, las personas pueden tener el rol de Profesor o de Alumno (o los dos). Una persona puede ser profesor o no serlo y, por lo tanto, la multiplicidad de la asociación de *Persona* con el rol *Profesor* tendrá multiplicidad 0..1. Lo mismo sucederá con la clase *Alumno*.



- Los momentos o intervalos pueden tener asociados varios roles correspondientes a los participantes, los lugares y las cosas que intervienen en ellos (bajo un rol determinado, como comprador o vendedor).

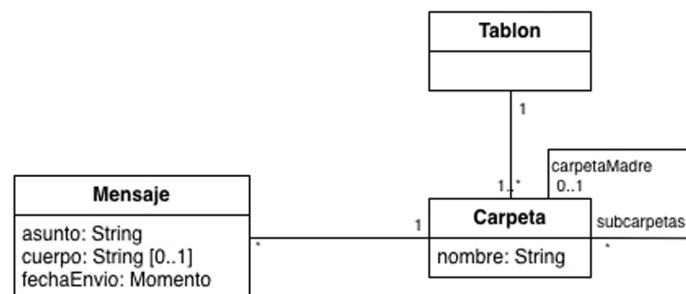
Otra lista de categorías de asociación nos la propone Larman (2005). Algunas de las categorías más interesantes de esta lista son:

- Las transacciones pueden tener asociadas otras transacciones, como la asociación entre una venta y su pago.

- Algunas transacciones tienen asociadas líneas de transacción, como las líneas de venta de una venta de un supermercado.
- Las transacciones o líneas de transacción pueden tener asociado un producto o descripción de producto, como la asociación entre la línea de venta del supermercado y la descripción del producto vendido.
- Las transacciones pueden tener asociados participantes por medio de sus roles.
- Algunas clases representan objetos que están compuestos lógica o físicamente por objetos de otra clase, como los asientos de una sala de cine o los pisos de un edificio.
- Nos puede interesar la relación de contenedor-contenido entre instancias, como es el caso de un coche y la plaza de aparcamiento en la que está aparcado.

Ejemplo

Éste es el caso de los tablones. Cada tablón contendrá un conjunto de carpetas; como mínimo una, a la que irán a parar los mensajes por defecto. Éstas no serán compartidas, sino que cada carpeta estará en el tablón en el que se creó y sólo en él. Dentro de cada carpeta habrá mensajes; de nuevo, un mensaje sólo podrá estar en una carpeta. Sin embargo, las carpetas pueden contener otras carpetas. Por lo tanto, cada carpeta tiene una carpeta madre (sacada de las carpetas raíz) y puede tener cualquier número de subcarpetas.



Los foros funcionarán de manera parecida a los tablones.

- Si tenemos productos pero también descripciones de producto, tendremos una asociación entre éstos, como la que puede existir entre una edición de una asignatura y su asignatura, que ya hemos identificado previamente.

Clases asociativas

Las clases asociativas permiten añadir atributos y asociaciones a las instancias de asociación. La representación en UML de una clase asociativa es la de una clase con una línea discontinua sin flechas que la conecta con la asociación correspondiente.

Véase también

Las clases asociativas se estudian en el módulo "Orientación a objetos".

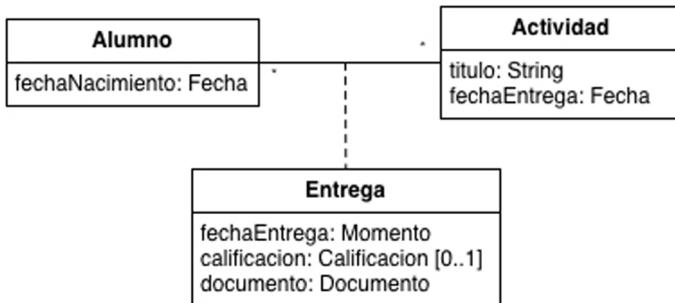
Ejemplo de clases asociativas

Supongamos, por ejemplo, que queremos saber qué alumnos entregan una determinadas actividades.



Pero, además, queremos hacer un seguimiento de la fecha y hora de entrega, y de las calificaciones que se pone a cada alumno de cada actividad, así como tener el documento entregado por el alumno; es decir, de cada entrega, queremos tener más información en forma de atributos. Una manera de representar esta información es añadir una clase asociativa para las entregas.

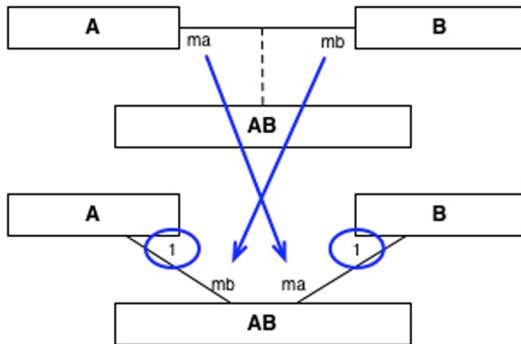
Representación de las entregas como una clase asociativa



Hemos indicado que la calificación es de tipo Calificación porque no queremos entrar en detalles todavía sobre si es un número del 0 al 10, un número del 0 al 100, una letra (A, B, C, D), etc. Por otro lado, hemos indicado la calificación como opcional porque sólo aquellas entregas ya corregidas tendrán calificación.

Algunos autores prefieren no usar clases asociativas, dado que introducen más conceptos y símbolos gráficos de los estrictamente necesarios.

Toda clase asociativa se puede representar, de hecho, como una clase normal que tenga, a su vez, dos asociaciones con las dos clases que asociaba.

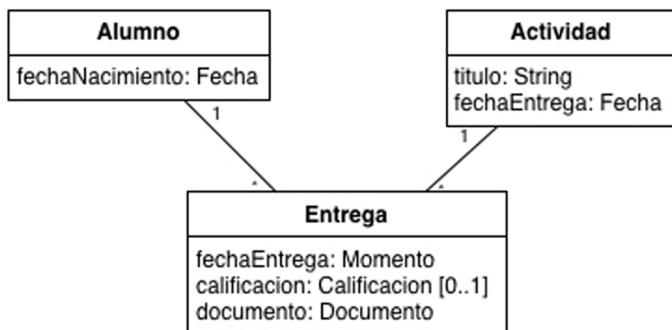


Las multiplicidades a los dos extremos de asociación más alejados de la clase originalmente asociativa siempre tienen multiplicidad 1, puesto que, por definición, cada instancia de la clase **AB** asocia una única instancia de la clase **A** y una y sólo una de la **B**.

Las multiplicidades en los extremos de asociación cercanos a la clase AB son las que aparecían en la asociación original: cada instancia de A tiene asociadas mb instancias de B y, para cada asociación, tendrá una instancia de AB; por definición, pues, cada instancia de A tendrá asociadas mb instancias de AB.

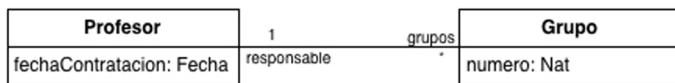
Representación de las entregas como una clase no asociativa con dos asociaciones

En nuestro ejemplo, las entregas se pueden representar también sin clases asociativas.



Asociaciones o atributos de tipos de clases del dominio

Tanto las asociaciones como los atributos acaban definiendo propiedades de las clases. Formalmente, en UML, una clase tiene un conjunto de propiedades formado por los atributos de la clase y los extremos de asociación de las asociaciones de la clase. Tomemos el ejemplo de asociación entre un profesor y los grupos que imparte:

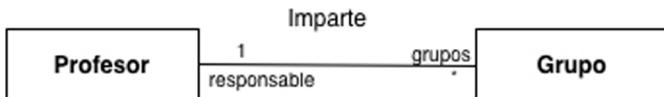


Esta asociación define dos propiedades nuevas que no son atributos:

- La clase *Grupo*, además de la propiedad *numero* de tipo *Nat*, tiene una propiedad *responsable* de tipo *Profesor* con multiplicidad 1.
- La clase *Profesor*, además de la propiedad *fechaContratacion* de tipo *Fecha*, tiene una propiedad *grupos* de tipos *Grupo* con multiplicidad *.

Por lo tanto, en realidad, los dos modelos siguientes son muy parecidos:

Representación más adecuada: como asociación



Representación menos adecuada: como atributos



Nombres de rol de asociación identificativos

Los nombres de las propiedades de las clases deben ser únicos. Por lo tanto, del mismo modo que no podemos tener dos atributos de una misma clase con el mismo nombre, tampoco podemos tener dos nombres de rol en dos asociaciones de una misma clase que definen propiedades con el mismo nombre o un nombre de rol que defina una propiedad con el mismo nombre que un atributo.

La diferencia radica en el hecho de que la primera representación es más visual y nos indica que las propiedades *grupos* y *responsable* son inversas. Es decir, que si cambiamos el responsable de un grupo, los grupos de este responsable y del que había antes habrán cambiado.

Por lo tanto, para relacionar clases del dominio preferiremos siempre asociaciones y no atributos.

Error frecuente: uso de claves foráneas

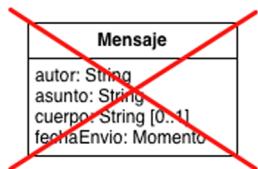
Un caso particular y muy típico de uso de atributos cuando habría que emplear asociaciones es el de las claves foráneas. Cuando se diseña una base de datos, las asociaciones se almacenan como atributos que contienen identificadores de registros de la tabla relacionada.

Por ello, algunos analistas usan atributos parecidos en sus modelos UML, ya sea además de la asociación correspondiente o en lugar de ésta.

Como hemos visto anteriormente, los dos casos son un error, dado que hay que representar la asociación como tal y no como atributo.

Ejemplo

Si, por ejemplo, hubiéramos dicho que el autor de un mensaje es un atributo de tipo String que contiene el DNI del autor, estaríamos cometiendo este error:



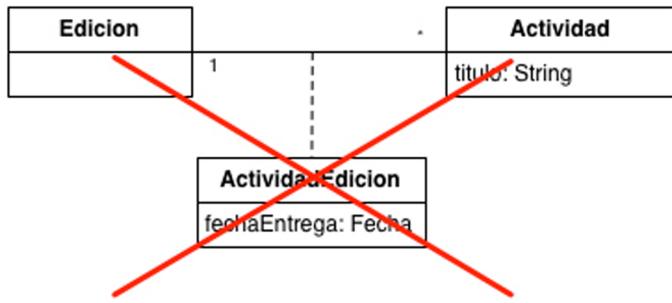
Representación incorrecta de una asociación usando claves foráneas

Error frecuente: clases de asociación con atributos erróneos

Un error habitual es usar clases asociativas para añadir atributos a una asociación que, en realidad, pertenecen a una de las clases asociadas. Esto sucede, especialmente, en asociaciones con una multiplicidad de 1 en uno de los extremos.

Ejemplo

Un ejemplo de este error sería pensar que la fecha de entrega de una actividad es un atributo de la asociación entre Actividad y Edicion. Dado que cada actividad sólo se lleva a cabo en una edición, esta fecha depende de la actividad pero no de la edición.



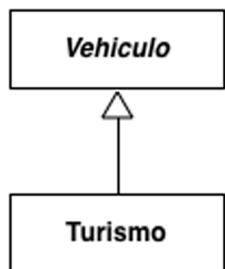
4.5. Herencia

4.5.1. Notación UML

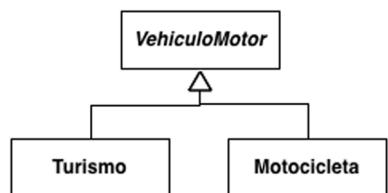
La notación UML de la herencia, que UML denomina *generalización*, es una línea continua desde la subclase hacia la superclase con un triángulo en la punta.

Ejemplo

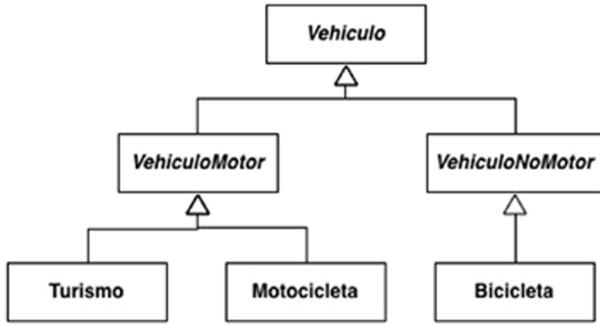
Ejemplo de notación UML de la generalización



En el supuesto de que una clase tenga más de una subclase, solemos agrupar las líneas de generalización en una única línea.



En UML, las clases abstractas (aquellas en las que todas las instancias deben ser instancia de alguna de sus subclases) se indican poniendo el nombre en cursiva.



4.5.2. Técnicas de modelización

Identificación de la herencia: generalización y especialización

Los procesos que nos permiten identificar la herencia son la generalización y la especialización.

En un momento dado, examinando una determinada clase, podemos detectar una posible subclase por **especialización** cuando nos encontramos alguno de los casos siguientes:

- La subclase candidata tiene atributos adicionales de interés (que no tienen todas las instancias de la clase examinada).
- La subclase candidata tiene asociaciones adicionales de interés (que no tienen todas las instancias de la clase examinada).
- La subclase candidata actúa de manera diferente en un sentido relevante. Puede ser que se use o se opere de manera diferente o que represente a una persona, animal o sistema con un comportamiento propio diferente.

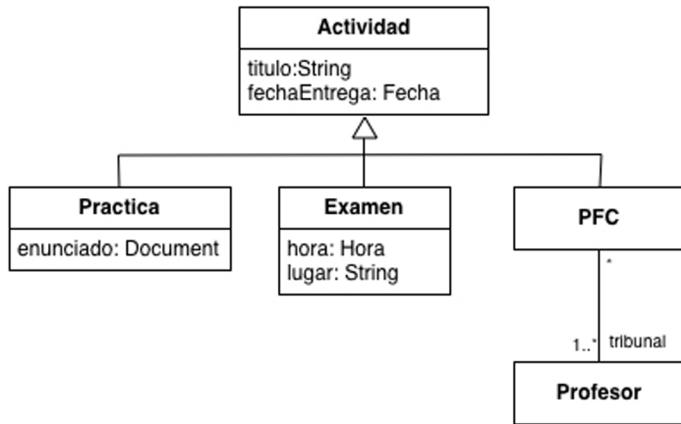
Ejemplo de especialización

En nuestro caso, nos podemos fijar en las actividades. Supongamos que tenemos actividades que son prácticas (se deben entregar en el campus virtual), otras que son exámenes presenciales y, finalmente, proyectos de final de carrera.

A parte de la información común a toda actividad, de las prácticas, dado que se hacen por el campus virtual, querremos tener el enunciado. De los exámenes deberemos conocer la hora y el lugar. Los proyectos de final de carrera (que se pueden considerar la única actividad de cada edición de la asignatura de proyecto de final de carrera) tienen un tribunal formado por uno o más profesores. Por lo tanto, podemos especializar las actividades de la manera siguiente:

Vé tambiéñ

Los procesos de generalización y especialización se estudian en el módulo "Orientación a objetos".

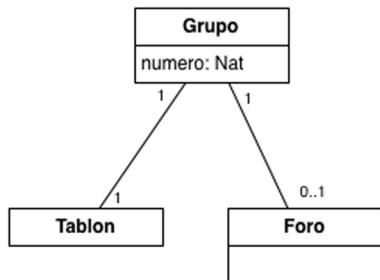


En cuanto a la **generalización**, podemos detectar una oportunidad si se da alguno de los casos siguientes:

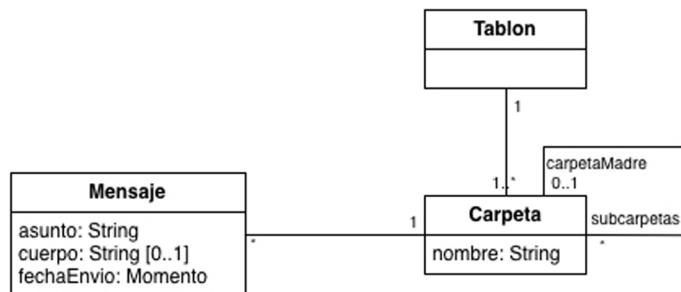
- Las subclases potenciales representan variaciones de un mismo concepto.
- Las subclases potenciales tienen uno o más atributos en común que se pueden expresar como atributos de la nueva superclase.
- Las subclases potenciales tienen una o más asociaciones en común que se pueden representar como asociaciones de la nueva superclase.

Ejemplo de generalización

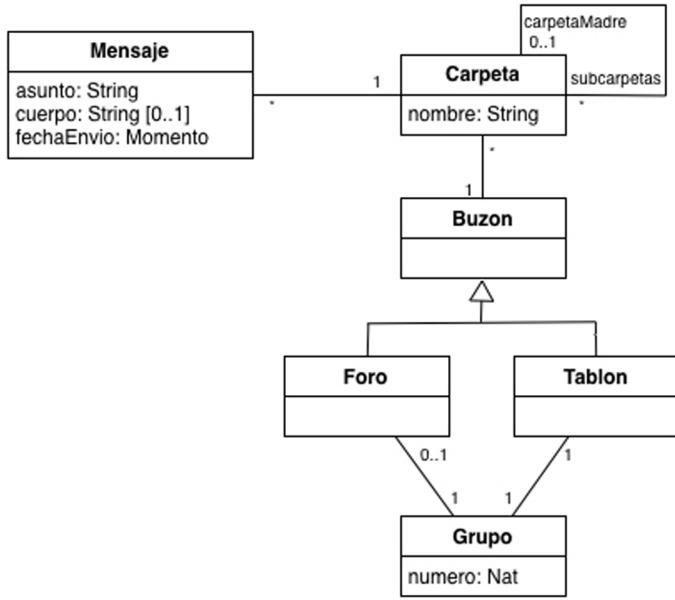
Siguiendo con nuestro ejemplo, hasta ahora hemos identificado que cada grupo de un curso tiene un tablón y, opcionalmente, un foro:



También hemos visto que el tablón contiene mensajes organizados en carpetas:



Pero los foros también tienen mensajes organizados en carpetas. Por lo tanto, foros y tablones son buzones de mensajes con algunas características particulares de funcionamiento y algunas asociaciones particulares (el tablón es obligatorio en un curso, pero el foro no) y podemos hacer una generalización para abstraer las partes en común a los dos:



Error frecuente: herencias falsas

La relación de herencia se puede estudiar bajo la teoría de conjuntos como la pertenencia de determinadas instancias a determinados conjuntos. Si representamos cada clase como un conjunto de las instancias de esa clase, la relación de herencia entre Actividad y Examen, por ejemplo, se puede representar de la manera siguiente:

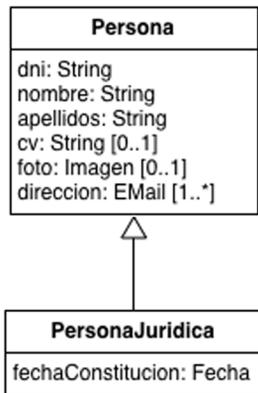


Por lo tanto, toda instancia de *examen* debe ser también una instancia de *actividad* y cuanto hemos definido de las actividades (asociaciones y atributos) ha de ser 100% cierto también para *exámenes*.

Hay que evitar, pues, las falsas herencias, en las que alguna instancia de una pretendida subclase no lo es de la superclase o en la que algún atributo o asociación de una pretendida superclase no es aplicable a todas sus subclases.

Ejemplo

Supongamos, por ejemplo, que nos interesa modelizar las personas jurídicas. Dado que tenemos ya una clase *persona*, todo parece indicar, a priori, que una persona jurídica será una subclase de *persona*:



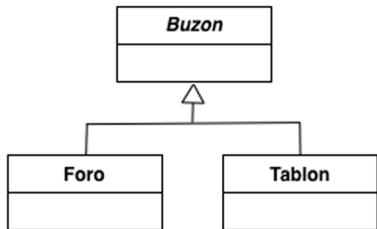
El problema es que las personas jurídicas no tienen apellidos, CV, ni foto y, en lugar de DNI, tienen un CIF. Por lo tanto, en realidad, no es cierto que cuanto podamos decir de las personas también sea cierto de las personas jurídicas.

Distinción entre clases abstractas y clases concretas

Cuando modelizamos una herencia, hay que plantearse si toda instancia de la superclase debe ser, forzosamente, instancia de alguna subclase o, por el contrario, puede ser que alguna instancia de la superclase no lo sea de ninguna subclase. En el primer caso, la superclase se definirá como abstracta.

Ejemplo de clase abstracta

En el caso de la generalización de tablones y foros en buzón de mensajes, observamos que no tendremos nunca un buzón de mensajes que no sea un tablón o un foro, dado que estamos suponiendo que el sistema que modelizamos no ofrece buzones de ningún otro tipo. Por lo tanto, habría que indicar que la clase *Buzon* es abstracta poniendo el nombre en cursiva.

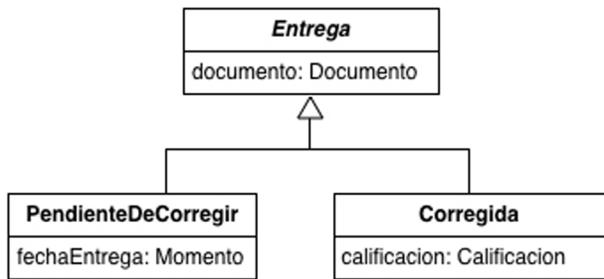


De manera parecida, la clase *Actividad* también será abstracta.

Modelización del estado de los objetos

Muy a menudo nos encontramos con una clase cuyos objetos pueden aparecer en varios estados. Así, por ejemplo, las entregas de actividades por parte de los alumnos se pueden considerar pendientes, hechas y corregidas. Para las entregas ya hechas nos puede interesar la fecha en la que se entregó, mientras que para las ya corregidas nos interesaría la calificación otorgada.

Una manera de representar estos posibles estados es usar la herencia:



Con este modelo, estamos diciendo que las instancias de entrega, cuando se crean, son instancias de *PendienteDeCorregir* que, una vez corregidas, pasan a ser instancias de *Corregida*. Esta herencia es, por lo tanto, dinámica.

Decimos que una herencia es dinámica cuando una instancia de la superclase a lo largo de su vida puede cambiar de subclase.

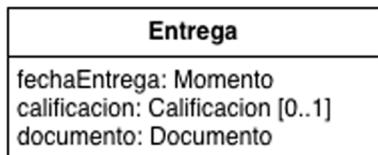
Este enfoque tiene el inconveniente de que el concepto de herencia dinámica es poco natural y, por lo tanto, poco entendible.

Otro problema habitual en este uso de la herencia es que puede inducir a errores respecto a qué atributos son relevantes en cada estado. El modelo anterior, por ejemplo, parece bastante natural, pero hemos cometido el error de considerar que para las entregas corregidas no nos interesa saber la fecha en la que se entregaron.

Una manera alternativa de representar esta situación es usar un atributo que indique el estado del objeto (discriminador) y hacer opcionales aquellos atributos que sólo toman valor en determinados estados, tal y como ya habíamos hecho.

Ejemplo

Representación de la clase *Entrega* sin reflejar, en el diagrama de clases, sus estados posibles



En este caso, el atributo *calificación* puede actuar como discriminador, dado que, por definición, una entrega corregida es aquella que tiene calificación. Otro caso habitual sería usar un atributo booleano (*corregida: boolean*) o de un tipo específico para casos con más de dos estados (*estadoEntrega: EstadoEntrega*).

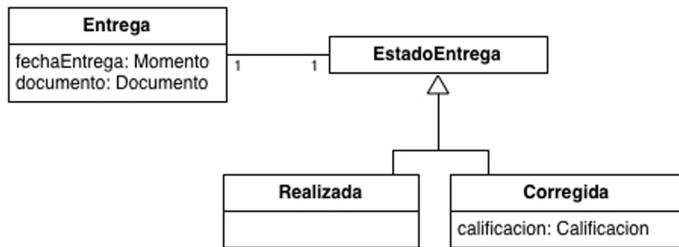
Forma más natural de entender la herencia

La herencia es un concepto de la orientación a objetos y, por lo tanto, es originaria de los lenguajes de programación. Dado que casi ningún lenguaje de programación permite que una instancia cambie de clase a lo largo de su vida, la manera más natural de entender la herencia es como herencia estática.

Otra solución consistiría en añadir una clase que represente el estado. En este caso habrá que tener cuidado de no cometer el mismo error respecto al atributo *fechaEntrega*:

Ejemplo

Representación de la clase *Entrega* que refleja sus posibles estados mediante una clase asociada



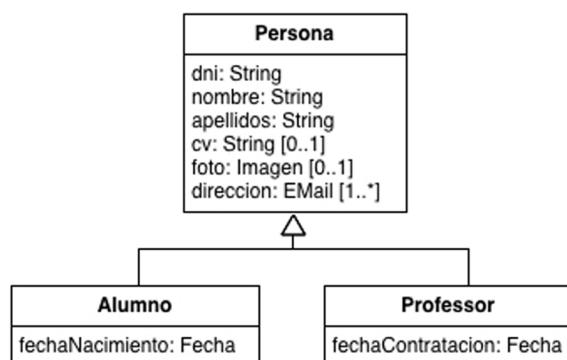
Modelización del rol de objetos y personas

Una situación habitual es que un objeto, especialmente si representa a una persona del mundo real, pueda tener un rol determinado dentro del dominio analizado. Es el caso, por ejemplo, de los profesores y alumnos de una universidad.

Supongamos, por ejemplo, que se han identificado profesores y alumnos de la facultad como dos clases de objetos:

Profesor	Alumno
dni: String nombre: String apellidos: String cv: String [0..1] foto: Imagen [0..1] direccion: EMail [1..*] fechaContratacion: Fecha	dni: String nombre: String apellidos: String cv: String [0..1] foto: Imagen [0..1] direccion: EMail [1..*] fechaNacimiento: Fecha

En esta situación, tal y como se ha explicado, encontramos una oportunidad muy clara de generalizar las partes comunes en una clase *Persona*:



Sin embargo, esta solución presenta el problema ya planteado de las herencias dinámicas. Una misma persona podría pasar de alumno a profesor y, por lo tanto, la herencia se puede considerar dinámica.

Pero, además, presenta una problemática adicional: supongamos que una persona puede ser, a la vez, Alumno y Profesor (por ejemplo, porque es alumno de unos estudios y profesor de otros). En tal caso, un mismo objeto de clase *Persona* debería pertenecer a las dos subclases a la vez.

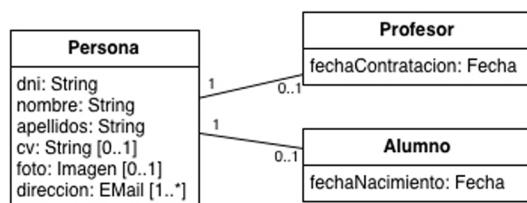
Decimos que una herencia es *overlapping* cuando una misma instancia de la superclase puede pertenecer a más de una subclase a la vez.

Por razones parecidas al caso de las herencias dinámicas, consideramos que las herencias *overlapping* no son naturales; son confusas, difíciles de entender y, por lo tanto, no consideramos su uso muy recomendable.

Una solución mejor consiste en modelizar los roles como clases propias asociadas a la clase que puede tener estos roles. Es, de hecho, lo que proponen Coad, Lefebvre y Luca (1999) cuando nos proponen los roles como una categoría de clases a la hora de identificarlas, tal y como se ha explicado en el subapartado 4.2.2.

Ejemplo

Modelización de los posibles roles de las personas en nuestro sistema como clases asociadas. En este caso, una persona puede tener el rol de Profesor, el de Alumno, los dos o ninguno de los dos.



4.6. Información derivada y reglas de integridad

4.6.1. Reglas de integridad

Una de las funcionalidades básicas del software para sistemas de información consiste en guardar información de manera persistente que permita hacer consultas y guiar la toma de decisiones en el día a día de la empresa, organización o persona que usa el sistema de información.

Ejemplo

El software para la universidad, por ejemplo, permitirá consultar los expedientes de los alumnos y guiar la evaluación de éstos, dado que, por ejemplo, puede proporcionar al profesor, a la hora de introducir las calificaciones, una lista de los estudiantes matriculados en su grupo.

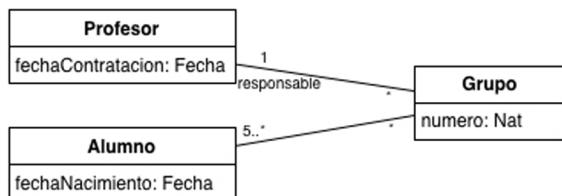
Para que esta información sea útil será importante mantener su integridad: habrá que evitar situaciones en las que la información es contradictoria o incompleta.

En el supuesto que nos ocupa será importante evitar, por ejemplo, situaciones en las que se califica una entrega de una actividad a un alumno que no está matriculado en ese curso o situaciones en las que no se dispone de la calificación de una actividad que se corrigió.

El modelo del dominio, por lo tanto, debe documentar qué restricciones de integridad han de satisfacer los datos que gestiona el sistema. Los diagramas de clases UML nos permiten representar gráficamente algunas de estas restricciones de integridad. Es el caso, por ejemplo, de las multiplicidades.

Ejemplo

En el fragmento de diagrama mostrado, que es una parte del que ya hemos analizado, hemos documentado varias restricciones de integridad. Decimos, por ejemplo, que un grupo debe tener como mínimo cinco alumnos y que, por lo tanto, si se diera el caso de contar con un grupo inferior a cinco alumnos, el sistema debería considerarlo una situación errónea. De manera parecida, indicamos que cada grupo tiene un único profesor y que por ello no se puede dar el caso de que un grupo tenga dos profesores o que carezca de representante.



No obstante, otras muchas restricciones de integridad no pueden ser representadas gráficamente en el lenguaje UML. Para éstas habrá que redactar un documento que las recopile.

Claves de las clases del dominio

Un subconjunto especialmente importante de las restricciones de integridad que habrá que documentar son las claves de las clases del dominio. Casi cada clase del dominio tendrá, típicamente, un atributo o conjunto de atributos que identifica las instancias de manera única. Por lo tanto, el sistema deberá garantizar que no existen dos instancias de una misma clase con idéntica clave.

Ejemplo

En el supuesto que nos ocupa, si, por ejemplo, el sistema detectara que se está dando de alta una persona con el mismo DNI que una persona existente, se debería indicar que se trata de un error: o bien la persona ya se ha dado de alta en el sistema con anterioridad o bien existe un error en el DNI de la nueva persona que se está dando de alta. En cualquiera de los dos casos, permitir el alta de dos personas con el mismo DNI implicaría romper la integridad de los datos.

Decimos, pues, que la clave de persona es el DNI. Así, en el documento en el que aparece la lista de las claves de las clases del dominio, indicaremos:

Las clases del dominio tendrán las claves siguientes:

- Persona: dni
- Asignatura: nombre
- Campus: nombre

En el caso de los semestres, el identificador serán el año y el número de semestre, cosa que indicaremos como:

- Semestre: año + numero

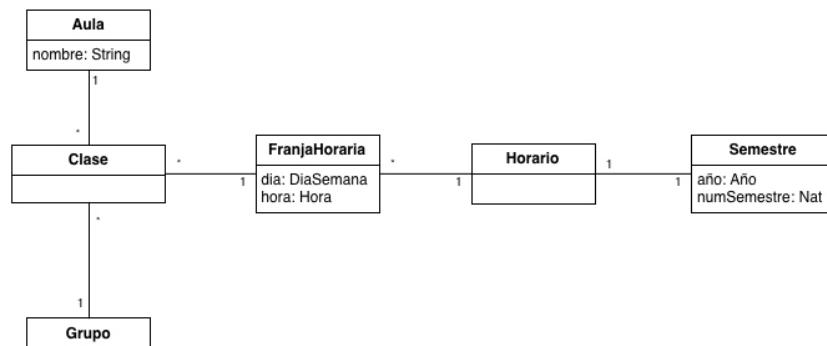
Los edificios, en principio, se identificarán por *nombre*. Pero podría haber dos edificios que se denominaran igual, siempre que estuvieran en campus diferentes, dado que continuaría sin haber confusión posible. Por lo tanto, el nombre del edificio lo identifica dentro del campus. Asimismo, podríamos considerar que la clave de edificio es nombre del campus + nombre del edificio. No obstante, para simplificar diremos que la clave de edificio es campus + nombre (es decir, una vez identificado un campus, el edificio se identifica por *nombre*). De manera parecida, la clave de la clase *Aula* estará formada por el edificio y el nombre.

- Edificio: campus + nombre
- Aula: edificio + nombre

Cada semestre tiene su propio horario. Por lo tanto:

- Horario: semestre

Pero el caso de las franjas horarias merece una atención especial:



En un horario, la franja horaria se identificará por el día de la semana y la hora:

- FranjaHoraria: horario + dia + hora

En cuanto a las clases, en una misma franja puede haber más de una clase, siempre que se cumpla una serie de condiciones: no puede haber dos clases del mismo grupo en la misma franja horaria y no puede haber dos clases en la misma aula y en la misma franja horaria. Por lo tanto, en este caso, tenemos dos identificadores:

- ClaseProgramada: franjaHoraria + aula, franjaHoraria + grupo

Cada edición de una asignatura se corresponde a un semestre de una asignatura. Y, en una edición, los grupos se identifican por número. Por lo tanto:

- Edicion: semestre + asignatura
- Grupo: edicion + numero

DNI como clave

A pesar de que el DNI parece una clave obvia para identificar a personas en el Estado español, en realidad existe una intensa problemática asociada a los números repetidos que nosotros no tendremos en cuenta pero que nos podría afectar en sistemas reales.

Cada tablón tiene asociado sólo un grupo y es el único tablón del grupo. Lo mismo sucede con los foros, en el caso de que el grupo los tenga. Por lo tanto:

- Tablon: grupo
- Foro: grupo

Tablones y foros tienen una superclase *Buzon*, pero no tenemos ninguna manera de identificar los buzones por sí mismos. Por lo tanto, la clase *Buzon* no tiene clave.

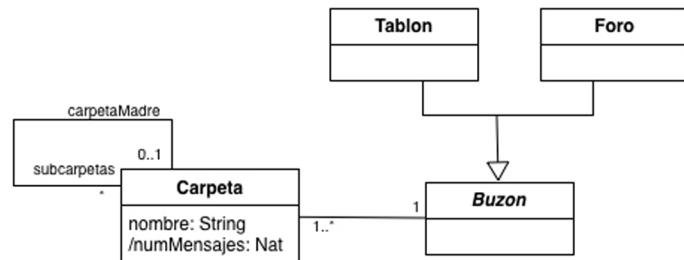
Las actividades de una edición de una asignatura deben ser identificadas por *título*. Obviamente, podría haber dos actividades con el mismo título siempre que sean de diferentes ediciones de asignatura.

- Actividad: edicion + titulo

Cada instancia de rol de profesor o de alumno se corresponde con una única persona y esta persona es la que la identifica:

- Profesor: persona
- Alumno: persona

El caso de las carpetas de los buzones también hay que estudiarlo con atención:



Es evidente que no puede haber dos carpetas con el mismo nombre dentro de una misma carpeta. Por lo tanto, parece que la clave de carpeta debe ser carpetaMadre + nombre. No obstante, sí puede haber dos carpetas con el mismo nombre que no tengan madre, siempre que estén en buzones diferentes. Por lo tanto, ¿la clave es buzón + nombre? No, porque si se pueden tener dos carpetas diferentes, del mismo buzón, que tengan el mismo nombre, siempre que tengan madres diferentes.

En este caso, por lo tanto, no podemos encontrar ninguna clave.

Por último, en cuanto a los mensajes, podríamos tener dos exactamente iguales (con el mismo asunto, cuerpo y momento de envío) dentro de la misma carpeta. Por lo tanto, la única manera de identificar un mensaje es mediante la posición que ocupa dentro de la carpeta.

- Mensaje: carpeta + posición en carpeta

Otras restricciones de integridad

Aparte de las restricciones de clave, conviene documentar de manera textual cualquier otra restricción de integridad que el lenguaje UML no nos permita expresar gráficamente.

Ejemplo de restricciones

En nuestro ejemplo encontraremos algunas restricciones que deberemos documentar:

- Para toda actividad, la fecha de entrega debe ser una fecha dentro del semestre de la edición a la que está asociada la actividad.
- Las carpetas forman una jerarquía válida de madres-subcarpetas (no se pueden formar ciclos en los que una carpeta tenga como madre una hija, o una hija de su hija, etc.).
- No puede haber más de una carpeta sin madre con el mismo nombre dentro del mismo buzón.
- No puede haber más de una carpeta con la misma carpeta madre y el mismo nombre.

4.6.2. Información derivada

A veces nos encontramos con atributos o asociaciones que pueden ser de interés pero que, en realidad, derivan de otros atributos y asociaciones; es decir, que su valor se puede deducir a partir de información ya modelizada.

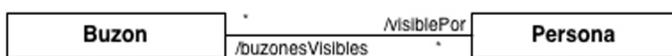
Ejemplo de información derivada

Supongamos, por ejemplo, que de las personas nos interesa saber su fecha de nacimiento y edad. Diremos que la edad es derivada porque se puede calcular a partir de la fecha de nacimiento.

Otro ejemplo de atributo derivado podría ser el número de mensajes en una carpeta. A pesar de que todavía no lo hemos puesto como atributo de la clase *Carpeta*, sabemos que es relevante aunque se puede derivar de la asociación entre Mensaje y Carpeta.

Las asociaciones también pueden ser derivadas. Como ejemplo, podemos pensar en la lista de buzones visibles por una persona. Si la introducimos como asociación entre Persona y Buzon, obtendremos una asociación que nos dará información relevante pero que es derivada, dado que una persona tiene visibles los foros y tablones de los grupos en los que participa como profesor o como alumno.

UML permite indicar que un atributo es derivado poniendo una barra (/) delante de su nombre. De manera parecida, podemos indicar que una asociación es derivada poniendo una barra delante de su nombre o el nombre de los roles.



De manera parecida a lo que sucede con las restricciones de integridad, UML no ofrece una manera gráfica de expresar cómo se deriva la información derivada. Por lo tanto, conviene documentarlo textualmente.

En nuestro ejemplo, deberíamos documentar:

- *Carpeta::numMensajes* es el número de mensajes asociados a la carpeta.
- *Persona::buzonesVisibles* es el conjunto formado por el tablón y el foro (para quienes tengan) de todos los grupos a los que la persona está asociada como profesor o como alumno.

4.7. Clases y atributos: elementos avanzados

4.7.1. Tipos de datos

Hasta ahora hemos usado, en los atributos, tipos como String, Nat, Data, pero también cantidades u otros valores sin entidad propia, como Direccion, NumSemestre o Calificacion. Estos tipos los denominamos *tipos de datos*.

Un tipo de datos es un conjunto de valores para los que la identidad única no tiene sentido; es decir, un conjunto de valores que se comparan por valor y no por identidad.

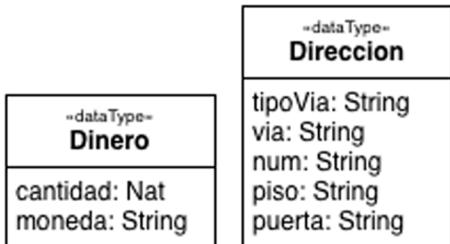
Si pensamos en el número 15 o en la hora 13:46, no podemos pensar en términos de identidad. No tiene sentido distinguir entre dos instancias diferentes del número 15 o de las 13:46 horas, ni pensar en términos de borrar un número 15 o tener 25 copias; y tampoco tiene sentido pensar en una lista de todas las horas. Por lo tanto, los números y las horas son dos ejemplos de tipos de datos.

En cambio, aunque dos mensajes de una carpeta tengan el mismo autor, asunto, cuerpo y fecha de envío, los dos mensajes no son el mismo, ya que tienen una identidad propia. Si borramos uno de los mensajes, el otro no se borrará; no es lo mismo tener 2 copias del mensaje que tener 25; si elaboramos una lista de los mensajes de una carpeta nos aparecerán dos mensajes iguales, uno debajo del otro.

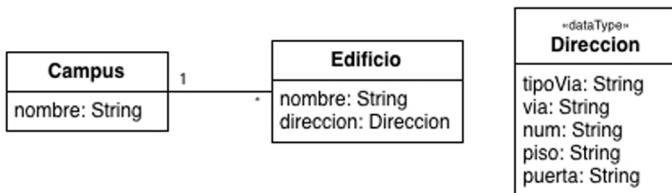
Otra diferencia es que los valores pertenecientes a tipos de datos son, desde un punto de vista conceptual, inmutables. No tiene sentido modificar las 13:46 horas. En todo caso, si teníamos una reunión entonces, la podemos sustituir por las 14:00 horas cambiando el valor 13:46 por el valor 14:00. Si realmente modificáramos el valor 13:46, todas las reuniones que empiecen o acaben a las 13:46, los mensajes enviados a las 13:46, etc., sufrirían un cambio en alguno de sus atributos.

Los tipos de datos pueden ser primitivos (como los Strings, los números, los booleanos o las fechas) o pueden ser estructurados, constituidos, a su vez, por varios campos, como las cantidades (23 €), los colores (RGB 70, 0, 130), etc. En el caso de los tipos de datos estructurados, nos puede interesar representarlos visualmente en el diagrama de clases UML para indicar cuáles son los campos que los componen.

Para representar un tipo de datos en UML se representa un clasificador con el estereotipo *dataType*.



Incluso cuando representamos un tipo de datos de manera gráfica, preferiremos modelizar las propiedades de este tipo como atributos en lugar de como asociaciones.



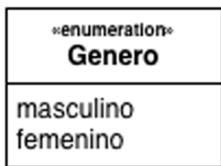
Enumeraciones

Una enumeración es un tipo de datos especial en el que damos la lista exacta de valores que forman el tipo de datos.

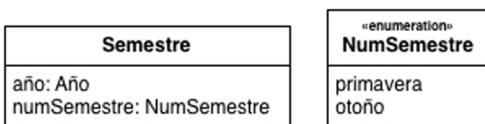
Ejemplo de enumeración

Si estuviéramos interesados en el género de las personas, definiríamos, en la clase *persona*, un atributo *género*. Éste no es, obviamente, un String ni un booleano (aunque sólo pueda tomar dos valores, éstos no son *cierto* o *falso*).

Para definir una enumeración sólo hay que elaborar una lista de los valores válidos. UML ofrece una notación gráfica a modo de clasificador con el estereotipo *enumeration* y con la lista de valores en lugar de los atributos.



En el ejemplo del campus sí teníamos un atributo para el número de semestre. Teniendo en cuenta que éste sólo puede ser 1 o 2 o –si usamos la nomenclatura propia de la universidad de ejemplo– primavera u otoño, podríamos haber definido el semestre de la manera siguiente:



4.7.2. Atributos: notación UML avanzada

A pesar de que la mayoría no se utilizan en el modelo del dominio, los atributos UML tienen otros elementos que se pueden indicar según muestra la sintaxis completa:

```
visibilidad nombre: tipo multiplicidad = valor-por-defecto  
{propiedades}
```

Los elementos de los que todavía no habíamos hablado son:

- **visibilidad.** Un único carácter que indica la visibilidad del atributo y que puede ser "+" para visibilidad pública, "-" para privada, "#" para protegida y "~" para visibilidad de paquete. Normalmente, durante el análisis, no indicaremos ninguna visibilidad en los atributos de nuestros modelos.
- **valor-por-defecto.** El valor por defecto que toma el atributo cuando se crea un objeto en el caso de que no se indique un valor inicial para éste.
- **propiedades.** Permite indicar propiedades adicionales al atributo. Un ejemplo de propiedad es {readOnly}, que indica que un atributo no puede ser modificado una vez se ha creado el objeto, u {ordered}, que, para un atributo multivalorado, indica que el orden de los valores es relevante y que, por ejemplo, no es lo mismo tener los valores [933455667, 654433221] que los valores [654433221, 933455667].

Modelo de dominio

Para realizar el modelo del dominio, no usaremos la visibilidad ni el valor por defecto, que, por lo tanto, no se indicarán; y muy raramente se usarán propiedades.

Otro concepto de orientación a objetos que no usamos durante el análisis es el ámbito. En UML los atributos con ámbito de clase aparecen subrayados, al contrario que los atributos con ámbito de instancia.

4.8. Asociaciones: elementos avanzados

4.8.1. Composición

Algunas asociaciones representan una asociación parte-todo más fuerte de lo habitual. Para representar este hecho, UML define el concepto de composición.

Una composición UML es una asociación entre dos clases (la compuesta y la clase componente), de manera que:

- Si destruimos una instancia compuesta, todos sus componentes también son destruidos.
- No puede haber instancias de la clase componente que no formen parte de una única instancia compuesta.
- No podemos tomar una instancia componente y cambiarla de compuesto.

UML permite representar las composiciones con un rombo negro en el extremo de la clase compuesta:



Ejemplo de composición

Cada edición representa, como ya se ha visto, un semestre de una asignatura y está formada por uno o más grupos. Si eliminamos una edición, todos sus grupos se deben eliminar; no tiene sentido tener un grupo que no sea de ninguna edición; y, finalmente, no tiene sentido tomar un grupo y cambiarlo de edición. Por lo tanto, podemos decir que la asociación entre Edición y Grupo es una composición en la que Edición es la clase compuesta y el Grupo, la clase componente, dado que una edición está compuesta por varios grupos.

UML también define un tipo de asociación que denomina *agregación*. Una agregación es una asociación con un cierto significado parte-todo, pero sin ninguna restricción o semántica especial (a diferencia de la composición). Por ello, los propios autores de UML y Larman recomiendan no usar agregaciones en nuestros modelos. No obstante, UML permite representarlas (al igual que las composiciones, pero con el rombo blanco).

4.8.2. Asociaciones con repeticiones o con orden

UML considera que los extremos de asociación son, por defecto, sin repeticiones y sin orden.



Así, en este ejemplo, un departamento puede tener un cierto número de profesores asociados, pero no existe ningún orden en este conjunto; dicho de otro modo, no tiene sentido distinguir entre el conjunto de profesores [Maria, Joan] y el conjunto [Joan, Maria].

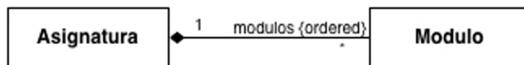
Además, en este ejemplo, el conjunto de los miembros de un departamento no puede tener repeticiones. Un mismo profesor no puede ser miembro más de una vez del mismo departamento; o, lo que es lo mismo, no tiene sentido

decir que los miembros de un departamento son [Maria, Maria, Joan] (si entendemos que esto significa que una única profesora Maria es miembro del departamento dos veces).

Si en un extremo de asociación el orden de los objetos es relevante, hay que indicarlo, en UML, usando la palabra clave *ordered* entre llaves.

Ejemplo

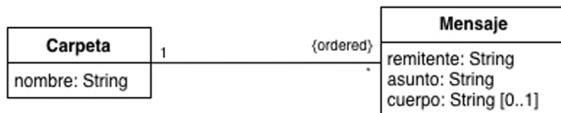
Si queremos conocer el temario de cada asignatura, podemos decir que una asignatura tiene un conjunto de módulos. Pero el orden de los temas de una asignatura es relevante, dado que no es lo mismo un temario formado por los módulos 2, 1 y 3 que decir que está formado por los módulos 1, 2 y 3.



Observación

Se ha considerado que la asociación de la asignatura con sus módulos es una composición porque si eliminamos una asignatura, debemos eliminar los módulos que la forman; no podemos tener un módulo que no pertenezca a ninguna asignatura. Además, no tiene sentido hablar de mover un módulo a otra asignatura. Podéis ver el subapartado 4.8.1.

Un último ejemplo. En una carpeta los mensajes tienen un orden que es relevante. A pesar de que se ordenan por fecha de llegada, si dos mensajes llegaran exactamente al mismo tiempo, es importante saber cuál va antes.



Normalmente, durante el análisis, no encontraremos la utilidad de usar asociaciones que permitan repeticiones. Sin embargo, UML permite indicarlo usando una notación parecida al caso anterior, pero con la palabra clave *non-unique*.

4.8.3. Notación UML avanzada

Los otros elementos de las asociaciones UML son:

- **Visibilidad de los extremos de asociación.** Un solo carácter ante el nombre de rol, que indica la visibilidad del extremo de asociación y que puede ser "+" para visibilidad pública, "-" para privada, "#" para protegida y "~" para visibilidad de paquete.
- **Propiedades.** Como en el caso de los atributos, permiten indicar propiedades adicionales de los extremos de asociación, como las ya vistas *readonly* (que también es aplicable a los extremos de asociación), *non-unique* u *ordered*.

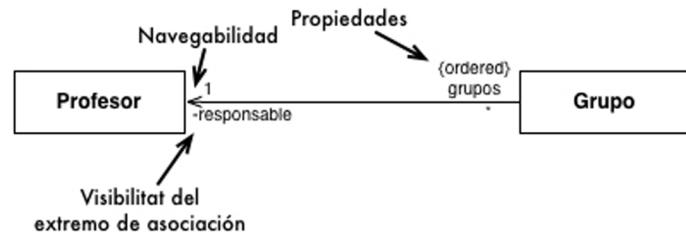
- Navegabilidad.** A veces queremos que una asociación sólo sea navegable en uno de sus sentidos, es decir, que sólo una de las clases tenga una propiedad ligada a la asociación. En el caso del ejemplo hemos indicado que la asociación es navegable de Grupo a Profesor –y, por lo tanto, la clase *Grupo* tiene una propiedad *Profesor-*, pero no lo es de Profesor a Grupo –y, por lo tanto, la clase *Profesor* no tiene ninguna propiedad denominada *grupos*.

Navegabilidades dobles

A pesar de que UML no indica ningún valor por defecto para las navegabilidades, durante el análisis, cuando elaboramos el modelo del dominio, supondremos que las navegabilidades son siempre dobles y, por lo tanto, no las indicaremos en los diagramas.

Ejemplo

Notación UML completa de las asociaciones

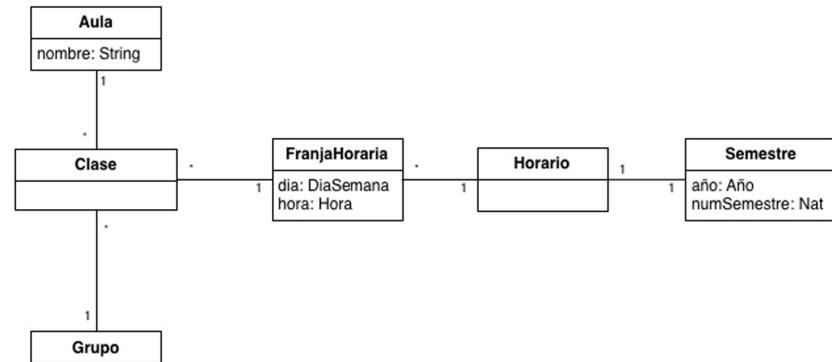


4.8.4. Asociaciones ternarias (y *N*-arias en las que *N* > 2)

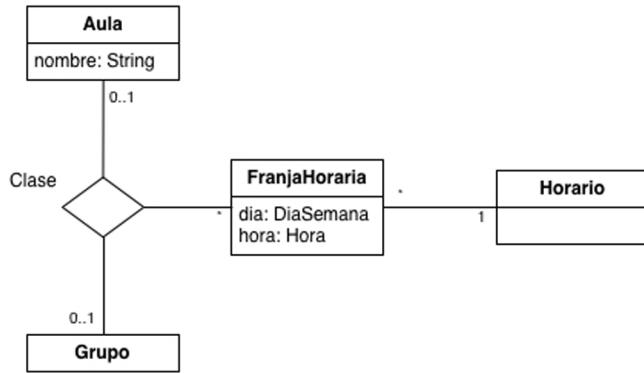
En UML, una asociación puede tener más de dos extremos de asociación. Denominamos **asociación ternaria** a las asociaciones con tres extremos de asociación y **asociación cuaternaria**, a las que tienen cuatro.

Ejemplo de asociaciones ternarias

Modelizamos los horarios como un conjunto de franjas horarias. En cada franja se programaban un conjunto de clases, que a su vez se asociaba a un grupo.



Este mismo concepto se habría podido expresar como una asociación ternaria que representara las clases.



La asociación *Clase* es una asociación ternaria, dado que tiene tres extremos de asociación: el grupo, el aula y la franja horaria (cada instancia de la asociación asocia un aula, un grupo y una franja horaria determinadas). En este sentido, la solución es bastante parecida a la anterior.

Las multiplicidades de una asociación ternaria funcionan de manera un poco especial. El 0..1 junto a aula indica que no puede haber más de una clase programada para el mismo grupo y la misma franja horaria. Esto no significa, por lo tanto, que podamos tener una clase que tenga grupo pero no aula.

De manera parecida, el 0..1 junto a grupo indica que no puede haber más de una clase programada en la misma aula y para la misma franja horaria.

En casos muy determinados las asociaciones ternarias (y otras asociaciones no binarias) permiten expresar de manera gráfica las restricciones de clave. En este sentido, son más expresivas que usar sólo clases y asociaciones binarias.

Ahora bien, se debe tener en cuenta que el uso de ternarias es menos comprensible, sobre todo respecto a sus multiplicidades, hasta el punto de que hay artículos de investigación dedicados a intentar aclarar su definición, como el de Génova, Lloréns y Martínez (2001).

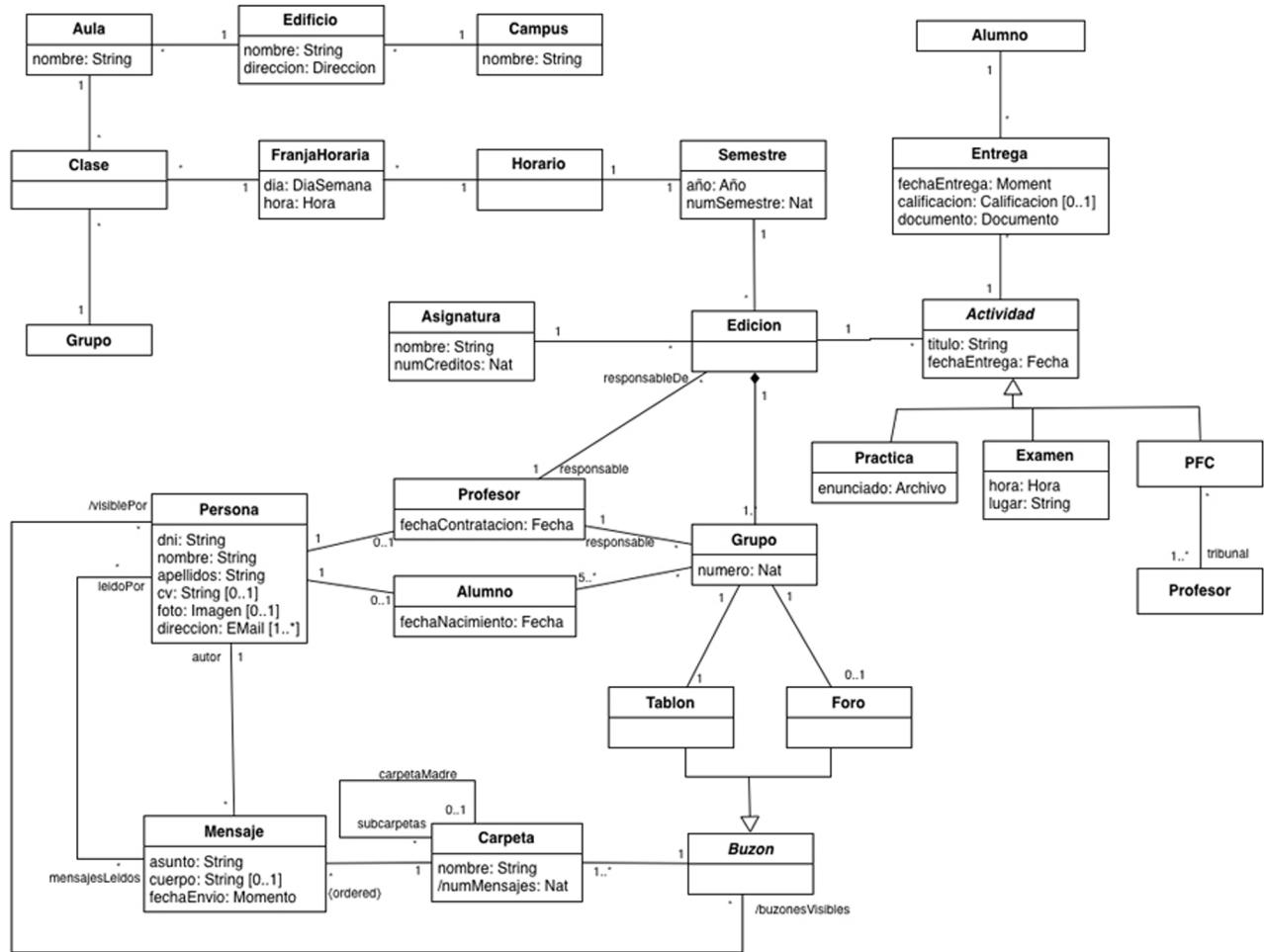
Dado que nuestra finalidad al hacer modelización del dominio es ayudar a entender el problema pendiente a las personas implicadas en el desarrollo de software, salvo que todas las partes implicadas entiendan muy bien el uso correcto de este tipo de asociaciones, no es aconsejable usarlas.

Por otro lado, la mayoría de las herramientas de modelización no soportan las asociaciones no binarias o lo hacen sólo parcialmente.

4.9. El modelo resultante

A continuación se muestra el modelo del dominio resultante del análisis del caso de ejemplo de la universidad. Algunas clases se han representado más de una vez para facilitar la lectura del diagrama; en tal caso, sólo una de las representaciones de la clase tiene el comportamiento de los atributos para evitar repetir innecesariamente esta información.

Éste no pretende ser un modelo fiel a la realidad, sino un ejemplo de modelo UML que representaría el modelo conceptual del campus virtual que hemos discutido. No hemos tenido en cuenta, por ejemplo, los requisitos ni los módulos que forman el temario de las asignaturas, a pesar de haber hablado de ello.



<code><dataType></code>	<code>Direccion</code>	<code><enumeration></code>	<code>DiaSemana</code>	<code><enumeration></code>	<code>NumSemestre</code>	<code><enumeration></code>	<code>Calificacion</code>
	tipoVia: String via: String num: String piso: String puerta: String						
			lunes martes miércoles jueves viernes sábado domingo		primavera otoño		excelente notable aprobado suspendido noPresentado

Modelo conceptual del campus virtual

Restricciones de clave:

- Persona: dni
- Asignatura: nombre
- Campus: nombre
- Semestre: año + numSemestre
- Edificio: campus + nombre
- Aula: edificio + nombre
- Horario: semestre
- FranjaHoraria: horario + dia + hora
- Clase: franjaHoraria + aula, franjaHoraria + grupo
- Edicion: semestre + asignatura
- Grupo: edicion + numero
- Tablon: grupo
- Foro: grupo
- Actividad: edicion + titulo
- Profesor: persona
- Alumno: persona
- Mensaje: carpeta + posición en carpeta::mensajes

Otras restricciones de integridad:

- Para toda actividad, la fecha de entrega debe ser una fecha del año del semestre del curso al que está asociada la actividad.
- Las carpetas forman una jerarquía válida de madres-subcarpetas (no se pueden formar ciclos cuya carpeta tenga como madre una hija, o una hija de su hija, etc.).
- No puede haber más de una carpeta sin madre con el mismo nombre dentro del mismo buzón.
- No puede haber más de una carpeta con la misma carpeta madre y el mismo nombre.

Información derivada:

- *Carpeta::numMensajes* es el número de mensajes asociados a la carpeta
- *Persona::buzonesVisibles* es el conjunto formado por el tablón y el foro (para quienes tengan) de todos los grupos a los que la persona está asociada como profesor o como alumno.

Resumen

En este módulo hemos visto cómo podemos utilizar los casos de uso y la orientación a objetos para realizar análisis en UML de software para sistemas de información.

El lenguaje UML es el lenguaje estándar que nos permite crear modelos visuales del software; se puede usar desde varias perspectivas, de las que nosotros usamos la conceptual.

El modelo de casos de uso consiste en elaborar el análisis basado en casos de uso, realizar la descripción textual tal y como hemos visto en el módulo "Requisitos" y hacer diagramas de casos de uso UML que muestren los casos de uso, los actores y las relaciones entre éstos. Hay que elegir con cuidado cómo agrupamos los casos de uso en diagramas para elaborar un documento lo más inteligible posible.

El diagrama de actividades UML nos permite documentar de manera visual el comportamiento de un caso de uso visto como un proceso. Nos permite indicar la secuencia de actividades que se llevan a cabo, ramas condicionales, paralelismo e incluso el modo como varios participantes interactúan en el proceso (mediante carriles).

Pero a menudo también querremos modelizar la interfaz del sistema para un determinado caso de uso. Para ello podemos redactar casos de uso concretos, que tienen en cuenta la tecnología y la implementación. Podemos representar las pantallas de un caso de uso haciendo modelos que muestren los elementos principales que las forman; el diagrama de estados UML, además, nos permite representar visualmente las transiciones que se producen entre éstas.

El modelo del dominio es un diagrama de clases UML que representa las clases conceptuales del mundo real en un dominio de interés usando la orientación a objetos. Para elaborarlo, primero hay que identificar las clases y añadirles atributos y asociaciones. Podemos usar la relación de herencia para especializar clases (cuando encontramos casos concretos que se apartan de la definición general que teníamos hasta el momento) o para generalizar (cuando encontramos dos clases con elementos en común). El diagrama de clases permite, además, usar otros elementos, como la información derivada, las enumeraciones o la composición, para representar algunos conceptos particulares de manera bastante precisa y formal.

Actividades

1. Elaborad una posible especificación textual breve para los casos de uso "Entregar una actividad", "Entregar una versión nueva de una actividad" y "Adjuntar archivo" del ejemplo del subapartado 2.1. Tened especialmente en cuenta que hay que reflejar las relaciones entre casos de uso mostradas en el diagrama.

2. Rehaced el diagrama del ejemplo del subapartado 2.1 para no utilizar la relación *extend*, tal y como se recomienda en el subapartado 2.1.6. Repetid la actividad anterior teniendo en cuenta el nuevo diagrama.

3. Suponed que disponemos del caso de uso siguiente:

Preparar inicio curso (nivel usuario)

- El profesor de un grupo quiere preparar el inicio de curso. Primero debe elegir el grupo cuyo inicio de curso quiere preparar y, después, ha de escribir un mensaje de bienvenida en el tablón. Si su asignatura tiene foro, debe organizar las carpetas de éste. El mensaje de bienvenida en el tablón y la organización de las carpetas se pueden realizar en cualquier orden, pero no se puede dar el caso de uso por finalizado hasta que el profesor no haya hecho las dos cosas (salvo que el grupo no tenga foro, en cuyo caso basta con escribir el mensaje de bienvenida).

Realizad el diagrama de actividades de este caso de uso.

4. Suponed que disponemos del caso de uso siguiente:

Preparar la edición de la asignatura (nivel usuario)

- Un miembro del personal académico, por orden del jefe de estudios, quiere dar de alta una nueva edición de una asignatura. Elige la asignatura y a un profesor responsable, y el sistema registra la nueva edición. A continuación, el usuario indica si en esta asignatura se usa foro o no y crea uno o más grupos de la asignatura. El sistema asigna, a cada grupo, un número de grupo consecutivo (que el usuario puede cambiar). Para cada grupo, el usuario asigna un profesor. El sistema graba los nuevos grupos y crea un tablón para cada uno y, si se ha indicado que se usan, un foro también para cada uno de ellos.
 - a) Escribid una especificación concreta y detallada de este caso de uso.
 - b) Realizad el diagrama de actividades de este caso de uso.
 - c) Proponed un modelo de interfaz gráfica de usuario que incluya esbozos de las pantallas y mapa navegacional.

5. Queremos desarrollar un sistema para llevar a cabo la gestión de proyectos mediante la metodología ágil Scrum (o, cuando menos, una versión simplificada).

Cada proyecto, que se identificará mediante un nombre, tendrá un equipo de al menos dos personas identificadas, también por un nombre, y estará formado por un conjunto de iteraciones (como mínimo una) que se identificarán, dentro del proyecto, por un número consecutivo. Además, de cada iteración querremos conocer la fecha de inicio, la fecha de la reunión de retrospectiva (con la que finaliza la iteración) y la velocidad prevista (un número correspondiente al número de puntos de valoración que se prevé, al iniciar la iteración, que pueda cumplir).

Cada proyecto tiene también un conjunto de historias de usuario llamado *backlog*. Las historias se identifican (dentro del proyecto) por un título, tienen un estado (pendiente, en desarrollo o finalizada) y una valoración (uno de los números del *planning poker*, no se incluye el carácter interrogante). Este conjunto de historias se ordenará, dentro del proyecto, por prioridad.

A medida que avanza el proyecto, en las iteraciones se van asignando historias de usuario no finalizadas, de tal manera que algunas iteraciones tendrán un conjunto de historias de usuario (de las del proyecto) ordenadas por prioridad.

Durante el desarrollo, cada día laborable, para cada proyecto, se organizará una reunión (el *daily scrum*) en la que, entre otras cuestiones, se decidirá cuántos puntos de valoración se considera que faltan para acabar la iteración. El sistema debe guardar el valor proporcionado cada día y, según éste, se elaborará un gráfico de seguimiento denominado *burndown chart*.

Al finalizar una iteración, las historias de usuario de la iteración que no se hayan acabado se asociarán a una nueva iteración y se dará la iteración por acabada. La velocidad real de una

iteración se calcula como la suma de las estimaciones de las historias de usuario que, para aquella iteración, se han finalizado.

Realizad el modelo del dominio del sistema descrito. Más concretamente:

- Elaborad el diagrama de clases del modelo del dominio.
- Indicad la información derivada y las reglas de integridad adicionales que sean necesarias.

6. Queremos desarrollar un sistema de gestión del catálogo para una cadena de ropa con varias tiendas distribuidas por todo el país.

Cada tienda tiene un nombre, que la identifica, y una dirección. El catálogo consiste en un conjunto de modelos que se identifican por un nombre y una marca, como el modelo 309 de Tangerine Jeans. De cada modelo hay varias variantes identificadas, dentro del modelo, por un nombre y una talla, como la variante Deep blue, talla 32, del modelo mencionado antes. De las marcas, además del nombre que las identifica, debemos conocer una dirección de correo electrónico de contacto.

Los precios de los modelos pueden variar según la temporada y la tienda. Por ello, cuando se indica el precio de un modelo en una tienda se indica el periodo de validez del precio. Un precio se considera vigente si la fecha actual está dentro del periodo de validez y no debe ocurrir que dos precios de un mismo modelo en la misma tienda coincidan. Si en una tienda y una fecha dada un modelo no tiene precio, se considera que no está disponible para la venta.

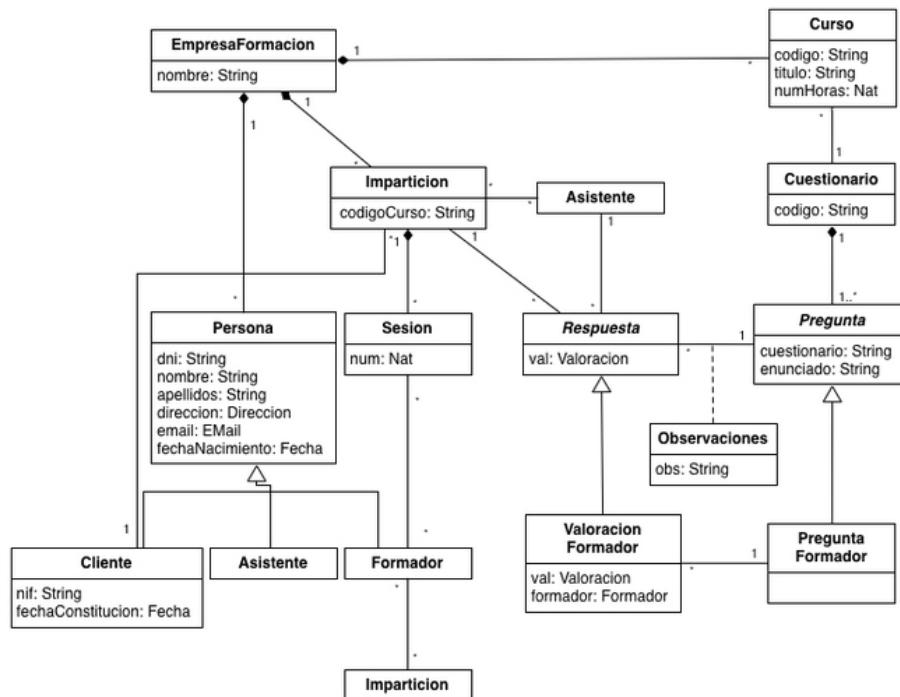
Algunos precios son de descuento. En tal caso, es necesario guardar un precio, como siempre, y un porcentaje de descuento.

Realizad el modelo del dominio del sistema descrito. Más concretamente:

- Elaborad el diagrama de clases del modelo del dominio.
- Indicad la información derivada y las reglas de integridad adicionales que sean necesarias.

7. Queremos desarrollar el sistema informático de la empresa de formación Training SAN para gestionar los cursos que ésta imparte, sus formadores y los cuestionarios de valoración de curso que pasan a los asistentes.

Tenemos un modelo del dominio ya elaborado:

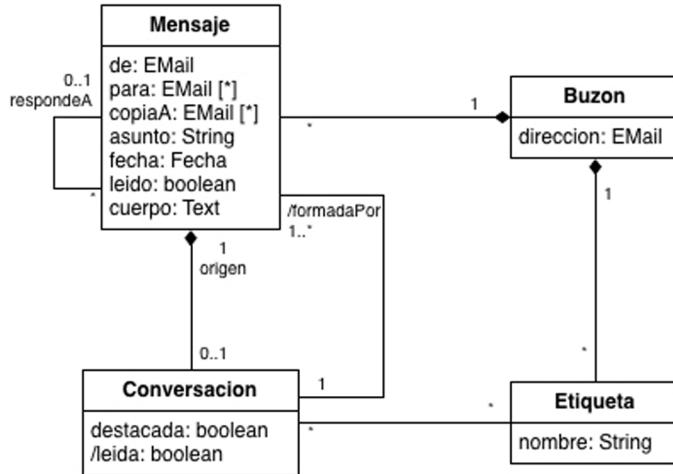


Este modelo muestra que cada curso tiene un cuestionario asociado que está compuesto por un conjunto de al menos una pregunta; algunas preguntas versan sobre el conjunto del curso, pero otras sólo lo hacen sobre el formador. Cuando se imparte un curso se quiere saber qué formadores lo imparten (y, en concreto, cuáles imparten cada sesión) y qué asistentes han asistido a cada sesión (se pasa lista). Al finalizar el curso se pasa el cuestionario y hay que

conocer las respuestas (valoración y observaciones) que hacen los asistentes; para las preguntas sobre el formador debe haber una respuesta por cada uno que haya impartido el curso.

Escribid una crítica del modelo del dominio propuesto e indicad cómo se pueden corregir los defectos que encontréis.

8. Disponemos de un sistema de correo electrónico que agrupa los correos por conversaciones y en el que una conversación es un correo que no responde a ningún otro correo ni al resto de las respuestas directas o indirectas que dependen de él; además, el sistema permite etiquetar las conversaciones. También disponemos de un modelo del dominio para el sistema en cuestión:



Claves de las clases:

- Buzon: direccion, Etiqueta:nombre

Restricciones de integridad e información derivada:

- Los mensajes que no son una respuesta, y sólo estos, deben ser origen de una conversación.
- Para una conversación *leida* es falso si y sólo si *leido* es falso para todos los mensajes que forman la conversación.
- Una conversación está formada por su mensaje origen y por las respuestas a cualquiera de los mensajes que forma parte de la conversación.
- Un mensaje sólo puede ser respuesta de otro que se halle en el mismo buzón.

Responded a las preguntas siguientes:

- ¿Puede ser que un mensaje ni tenga destinatarios (paraA) ni tenga copia a nadie (copiaA)?
- ¿Puede ser que un mensaje no tenga cuerpo?
- ¿Qué sucede si borramos un buzón?
- ¿Y si borramos una etiqueta?
- ¿Cómo podríamos representar la primera restricción de integridad de manera gráfica en UML?
- La multiplicidad del rol de asociación *formadaPor* es 1..*. ¿Por qué no es *?
- ¿Qué diferencias habría si en lugar de modelizar la etiqueta como una clase la hubiéramos modelizado como un atributo *etiqueta:String* de la clase *Conversacion*?

9. Queremos desarrollar un sistema para la gestión personal de finanzas. Los usuarios se registran proporcionando toda clase de datos y, una vez registrados, pueden dar de alta cuentas, que pueden ser cuentas corrientes, tarjetas de crédito o préstamos hipotecarios. El sistema les permite gestionar los movimientos de cada cuenta.

Disponemos de los modelos de dos de las pantallas:

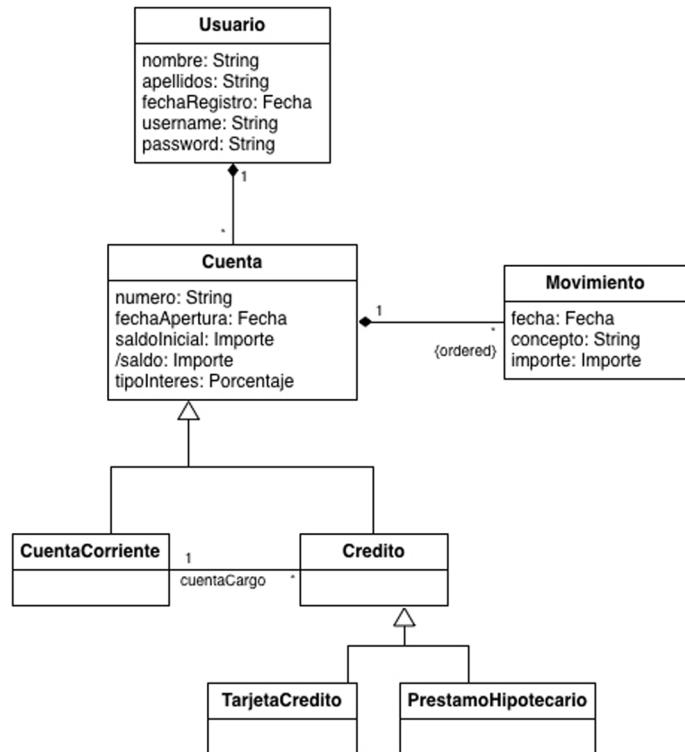
Posición global

Cuentas	Balances	Saldo	Opciones
1245789892	Débitos	999900€	Ver detalles
9898741258	Caja	999900€	Ver detalles
Tarjetas			
159482673	Personal	999900€ -999900€	Ver detalles
5261594473	Trabajo	999900€ -999900€	Ver detalles
Préstamos			
0124578369	Hipoteca	999900€ -999900€	Ver detalles
0926159487	Coches	999900€ -999900€	Ver detalles

Movimientos

Fecha	Efecto	Valor	Concepto	Importe	Saldo
20/06	21/06	La Terrada		999900€	999900€
12/06	15/06	Rentropago cajero		999900€	999900€
11/06	11/06	Café de Pisa		999900€	999900€
10/06	10/06	Newaditios Lorenzo		999900€	999900€
09/06	10/06	Rentropago cajero		999900€	999900€
06/06	06/06	Café de Pisa		999900€	999900€

También tenemos un modelo del dominio parcial que queríramos completar y corregir:



Claves de las clases:

- Usuario: nombreUsuario, Cuenta: usuario + numero, Movimiento: orden dentro de la cuenta

Restricciones de integridad

- El usuario de una cuenta de crédito debe ser el mismo que el usuario de la cuenta de cargo asociado.
- El saldo de una cuenta se calcula como saldoInicial + suma de todos los importes de todos los movimientos asociados a la cuenta.

Comprobad la validez del modelo del dominio asumiendo que los modelos de las pantallas son correctos. Indicad los errores y las carencias que encontréis.

Ejercicios de autoevaluación

1. ¿Qué es el lenguaje UML?
2. ¿Cuáles son los tres usos principales para el lenguaje UML?
3. ¿Cuál es la relación entre el diagrama de casos de uso y la descripción textual de éstos?
4. ¿Qué son los puntos de extensión de un caso de uso?
5. ¿Cómo podemos indicar la lógica condicional en un diagrama de actividades?
6. ¿Por qué hay que hacer un modelo de la interfaz de usuario?
7. ¿Qué son los casos de uso esenciales?
8. ¿Cuál es el objetivo de los modelos de las pantallas?
9. ¿Qué es el mapa navegacional?
10. ¿Qué es el modelo del dominio?
11. Indicad tres categorías típicas de clases conceptuales.
12. ¿Qué indicaciones nos da "la regla del cartógrafo"?
13. ¿Qué indica una multiplicidad "*" en un atributo?
14. ¿Por qué no usamos atributos para representar relaciones entre clases?
15. ¿En qué casos debemos plantearnos si hay que llevar a cabo un proceso de especialización de una clase?
16. ¿Qué son las restricciones de integridad?
17. ¿Qué es la información derivada?
18. ¿Cuál es la principal diferencia entre un tipo de datos y una clase?

Solucionario

Actividades

1.

a) **Caso de uso: entregar una actividad**

El estudiante pide entregar una actividad. El sistema le muestra una lista con las asignaturas que está cursando, de las que el estudiante elige una. A continuación, el sistema muestra una lista de actividades de la asignatura seleccionada y el usuario selecciona una y adjunta el archivo de la entrega. El sistema graba la entrega e incluye la fecha y la hora en la que se ha realizado.

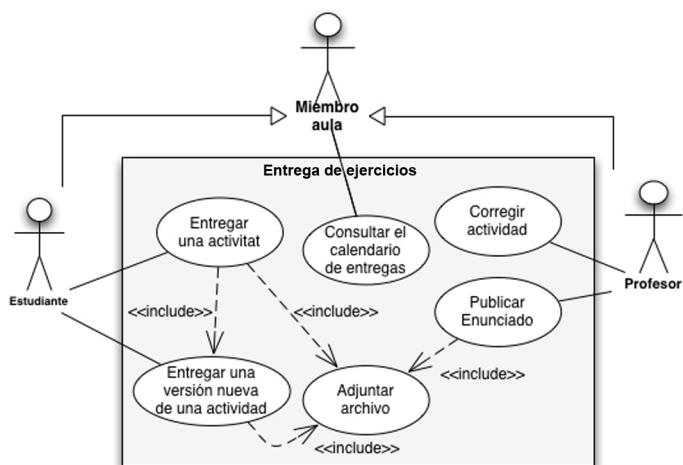
b) **Caso de uso: entregar una versión nueva de una actividad**

Este caso de uso extiende el caso de uso Entregar una actividad cuando el estudiante adjunta el archivo de entrega y resulta que ya había hecho la entrega previamente. El sistema pide confirmación y el usuario confirma que quiere entregar una nueva versión y adjunta un nuevo archivo. El sistema registra la nueva versión y la nueva fecha y hora de entrega.

c) **Caso de uso: adjuntar archivo**

El usuario quiere adjuntar un archivo y el sistema le muestra un navegador de su espacio de carpetas, situado, inicialmente, en su carpeta de usuario. El usuario navega por el espacio de carpetas hasta que encuentra el archivo que quiere adjuntar, lo selecciona y lo confirma. El sistema guarda el archivo adjuntado.

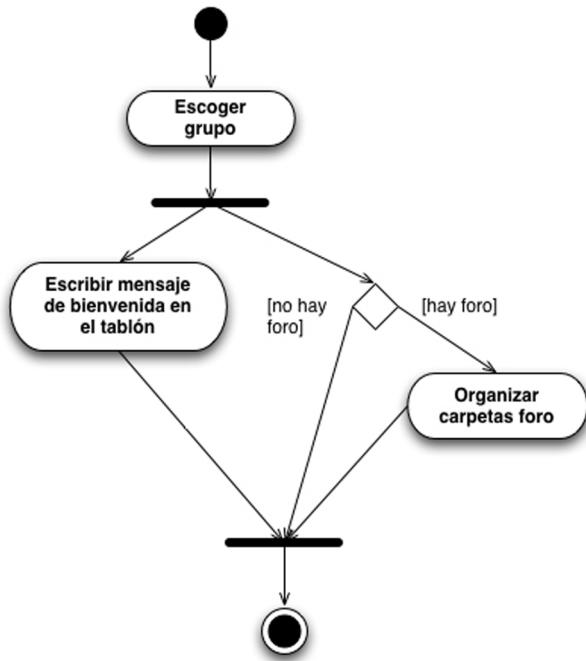
2.



La especificación textual del caso de uso *Adjuntar archivo* no cambia. La de los otros dos casos de uso sí se modifica:

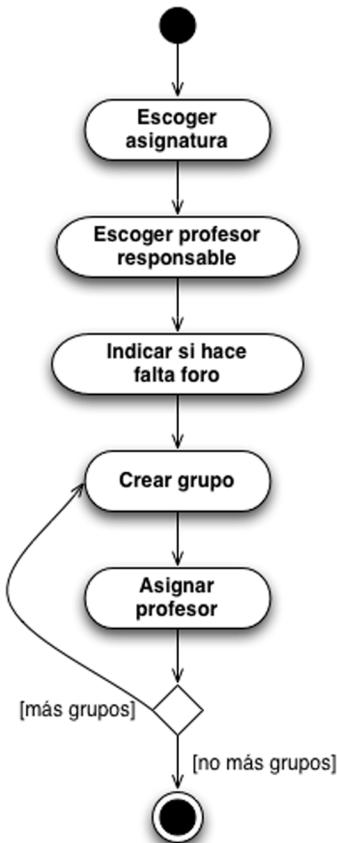
- a) **Caso de uso: entregar una actividad.** El estudiante pide entregar una actividad. El sistema le muestra una lista con las asignaturas que está cursando, de las que el estudiante elige una. A continuación, el sistema muestra una lista de actividades de la asignatura seleccionada y el usuario selecciona una y adjunta el archivo de la entrega. El sistema graba la entrega, incluyendo la fecha y hora en la que se ha hecho. Si el estudiante ya había hecho la entrega, el estudiante puede entregar una versión nueva de una actividad.
- b) **Caso de uso: entregar una versión nueva de una actividad.** El sistema pide confirmación y el usuario confirma que quiere entregar una nueva versión y adjunta un nuevo archivo. El sistema registra la nueva versión y la nueva fecha y hora de entrega.

3.



Se ha usado una bifurcación para indicar que escribir el mensaje de bienvenida y organizar las carpetas en el foro (si es necesario) se hacen en paralelo y una unión para indicar que el proceso no puede acabar hasta que no se han llevado a cabo las dos actividades. Asimismo, se ha usado una decisión para indicar que sólo se deben organizar las carpetas si hay foro; en caso contrario, se acude directamente a la unión y, por lo tanto, el proceso se acaba una vez escrito el mensaje de bienvenida.

4. El diagrama de actividades del caso de uso se muestra a continuación:



Un posible caso de uso concreto podría ser el siguiente:

Caso de uso: preparar la edición de la asignatura

Actor principal: usuario

Escenario principal de éxito:

1. El sistema muestra la pantalla *Elegir asignatura* con un desplegable de las asignaturas que aún no tienen edición en el curso actual.

2. El usuario selecciona una asignatura y pulsa *Siguiente*.

3. El sistema muestra la pantalla *Responsable y buzones* con un desplegable de profesores que pueden hacer de responsable de la asignatura.

4. El usuario selecciona un profesor, elige una configuración de buzones y pulsa *Siguiente*.

5. El sistema muestra la pantalla *Grupos* con un desplegable de profesores que pueden llevar un grupo de la asignatura para la creación de nuevos grupos.

6. El usuario selecciona un profesor para el nuevo grupo y pulsa *Añadir grupo*.

7. Los pasos 5 y 6 se repiten hasta que el usuario no quiere crear más grupos, momento en el que pulsa *Siguiente*.

8. El sistema muestra la pantalla *Confirmación* con todos los datos de la nueva edición de la asignatura.

9. El usuario pulsa *De acuerdo* y el sistema registra la nueva edición de la asignatura y sus grupos.

Extensiones:

2a. El usuario pulsa *Cancelar* y el caso de uso finaliza sin cambios.

*a En cualquier momento, a partir del paso 3, el usuario puede pulsar *Atrás* y el caso de uso vuelve a la pantalla anterior. (*a indica extensiones múltiples; en este caso, es válido para todos los pasos del 3 al 9).

Pantallas:

Pantalla	Datos mostrados	Datos introducidos
Elegir asignatura	Lista de nombres de asignatura	Asignatura seleccionada
Responsable y buzones	Asignatura seleccionada Lista de profesores que pueden ser responsables	Profesor seleccionado Sólo tablón/tablón y foro
Grupos	Asignatura, responsable y opciones de buzones Lista de grupos. Para cada uno: <ul style="list-style-type: none"> • Número • Profesor responsable Lista de profesores que pueden llevar un grupo	Profesor seleccionado como responsable de un grupo
Confirmación	Asignatura, responsable, opciones de buzones Lista de grupos. Para cada uno: <ul style="list-style-type: none"> • Número • Profesor responsable 	-

Como se puede ver, se debe decidir que el profesor responsable y la configuración de buzones de esta edición se realizan en la misma pantalla. Asimismo, se ha introducido una pantalla de confirmación al final.

Los esbozos de las pantallas son los siguientes:

Preparar edición de una asignatura

Asignatura: Ingeniería del Software

Empiece a teclear el nombre o parte del nombre de la asignatura para localizarla más fácilmente

Cancelar **Siguiente**

Preparar edición de una asignatura

Asignatura: Ingeniería del Software

Responsable: Mario Dener Lombart

Empiece a teclear el nombre o parte del nombre del profesor para localizarlo más fácilmente

Buzones de la asignatura:

- Sólo tablón
- Tablón y foro

Atrás **Siguiente**

Preparar edición de una asignatura

Asignatura: Ingeniería del Software

Responsable: Mario Dener Lombart

Grupos: 1 (Marta Sabio Romea)
2 (Juan Berraz Sortigo)

Nuevo grupo: Milena García Ferrero **Añadir grupo**

Empiece a teclear el nombre o parte del nombre del profesor para localizarlo más fácilmente

Atrás **Siguiente**

Preparar edición de una asignatura

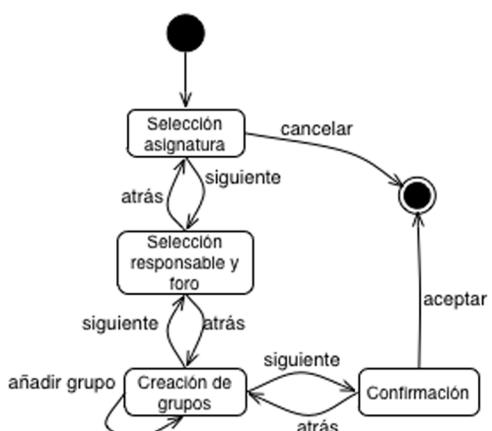
Asignatura: Ingeniería del Software

Responsable: Mario Dener Lombart

Grupos: 1 (Marta Sabio Romea)
2 (Juan Berraz Sortigo)
3 (Milena García Ferrero)

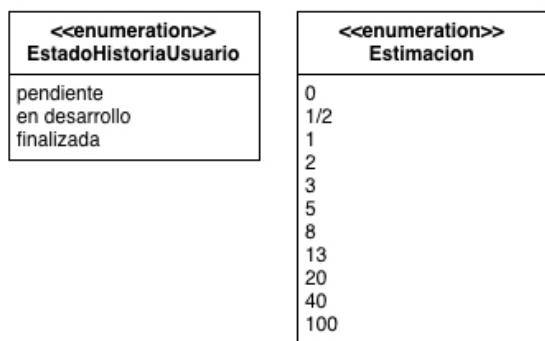
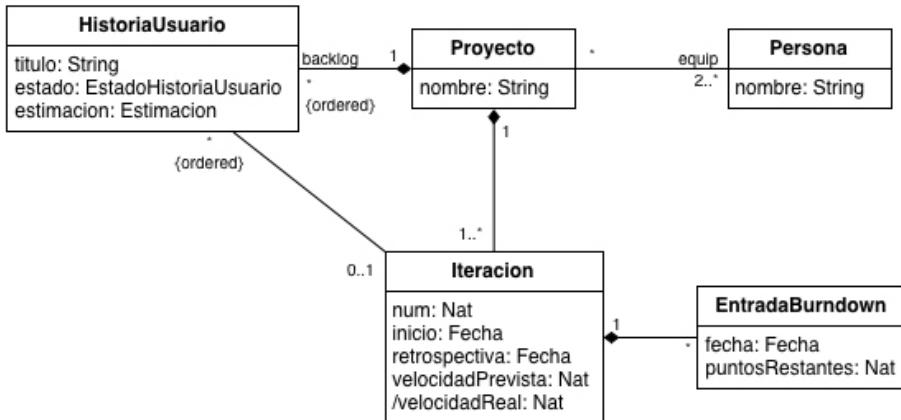
Atrás **Aceptar**

El mapa navegacional correspondiente es el siguiente:



A cada pantalla le corresponde un estado del diagrama. Las transiciones entre estados están etiquetadas con el acontecimiento que las provoca, que es una acción del usuario en la pantalla, como pulsar un botón.

5.



Restricciones de integridad:

- Claves: Proyecto: nombre, Persona: nombre, Iteracion: proyecto + nombre, HistoriaUsuario: proyecto + título, EntradaBurndown: iteración + fecha.
- La fecha de inicio de una iteración debe ser anterior a la fecha de retrospectiva.
- La fecha de todas las entradas de *burndown* de una iteración debe estar entre las fechas de inicio y de retrospectiva de ésta.
- Las historias de usuario asociadas a una iteración deben pertenecer al proyecto al que pertenece la iteración.

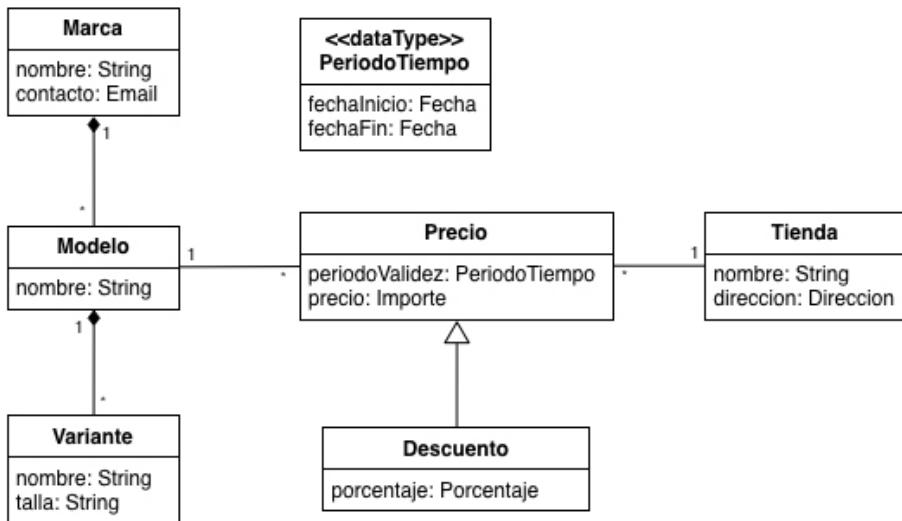
Información derivada:

- La velocidadReal de una iteración es la suma de las valoraciones de las historias de usuario finalizadas asociadas a esa iteración.

Notas:

- Se ha considerado que hay varias asociaciones que son composiciones. Así, un proyecto estará compuesto por su conjunto de historias de usuario y su conjunto de iteraciones. Éstas, a su vez, estarán compuestas por un conjunto de entradas de *burndown*. Por lo tanto, entre otras cosas, podemos afirmar que, si borramos un proyecto, también borraremos las historias de usuario, las iteraciones y las entradas de *burndown*.
- Se ha usado {ordered} para indicar que la lista de historias de usuario de un proyecto está ordenada y que el orden importa. También para la lista de historias de usuario de una iteración.

6.



Restricciones de integridad:

- Claves: Tienda: nombre, Marca: nombre, Modelo: marca + nombre, Variante: modelo + nombre + corta.
- No puede haber más de un precio del mismo modelo en la misma tienda con períodos de validez que coincidan.

Notas:

- Se ha modelizado el precio con descuento como una subclase. Sin embargo, a causa de su simplicidad, se habría podido indicar, sencillamente, el porcentaje de descuento como un atributo opcional de precio.
- Los tipos de datos Email, Dirección, Porcentaje e Importe no se han modelizado con más detalle o bien porque eran bastante evidentes (Email, Porcentaje e Importe) o bien porque en el enunciado no tenemos más información (Dirección).

7.

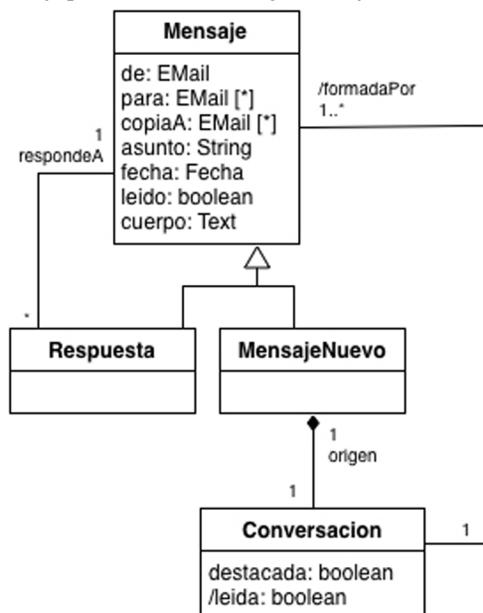
- De entrada, en el modelo mostrado no se ha indicado ningún tipo de restricciones de integridad adicionales, ni siquiera las de clave. Sería necesaria una nota adicional que informara sobre estas restricciones.
- La clase *ValoracionFormador* tiene un atributo de tipo Formador. Éste, en todo caso, debería ser una asociación.
- La clase *Imparticion* tiene un atributo *codigoCurso* de tipo String que parece un tipo de clave foránea hacia Curso. Habría que indicar que una *Imparticion* tiene asociado un Curso mediante una simple asociación y no un atributo que contenga el código del curso asociado.
- El modelo presentado tiene una clase *EmpresaFormacion* que no es necesaria. La presencia de esta clase parece indicar que las empresas de formación y su nombre son relevantes para nuestro sistema informático. Sin embargo, dado que es para una única empresa de formación, el hecho de conocer el nombre y de tener la posibilidad de tener más de una empresa de formación no es relevante.
- La clase asociativa *Observaciones* no tiene mucho sentido. Si, al responder, un asistente añade observaciones a la respuesta, éstas serían un atributo de la clase *Respuesta*. No tiene sentido modelizarlas como un atributo de una clase asociativa, sobre todo porque la multiplicidad junto a Pregunta es 1 y, por lo tanto, cada instancia de *Respuesta* tendrá una única instancia de *Observaciones* asociada.
- La clase *Pregunta* se ha indicado como abstracta, cuando en realidad sólo tiene una subclase. Excepto que todas las preguntas sean, efectivamente, de formador, la clase *Pregunta* debe ser concreta. Lo mismo sucede con la clase *Respuesta*.
- La clase *Persona* no se ha indicado como abstracta, a pesar de que parece que todas deban ser clientes, asistentes o formadores. Si, efectivamente, no puede haber personas relevantes en el problema que no pertenezcan a alguna de las 3 subclases, se deberá indicar que la clase *Persona* es abstracta (poniendo el nombre en cursiva).
- La clase *Cliente* no parece una subclase de *Persona* correctamente modelizada. Si un cliente tiene un NIF y una fecha de constitución, es porque debe ser una empresa o algún otro tipo de organización. Por lo tanto, no tiene sentido que digamos que es una subclase de *Persona*.
- La clase *Formador* tiene una asociación con la clase *Sesion* que indica las sesiones en las que participa cada formador. También tiene una asociación con la clase *Imparticion* que debe indicar los formadores de una impartición de un curso. Estas dos asociaciones deberían

estar relacionadas: o bien los profesores de una impartición derivarán de los profesores de cada una de las sesiones, o bien habrá una restricción de integridad que nos indique que todos los profesores asociados a una sesión también deben pertenecer a la impartición correspondiente.

- De manera parecida al caso anterior, una ValoracionFormador tiene asociada una PreguntaFormador y, como toda Valoracion, también una Pregunta. La asociación con PreguntaFormador sobra (y se debe indicar mediante una restricción de integridad textual que la pregunta asociada a una ValoracionFormador será una PreguntaFormador).

8.

- Según el modelo del que disponemos, sí.
- No. Para el cuerpo no se ha indicado ninguna multiplicidad y, por lo tanto, asumimos que la multiplicidad es 1; es decir, que es un atributo obligatorio y univaluado.
- Que todas las etiquetas y mensajes del buzón también se borrarán, dado que un buzón está compuesto por etiquetas y por mensajes. Al borrar aquellos mensajes que sean origen de una conversación, también se borrarán las conversaciones.
- Nada, dado que una etiqueta no es un objeto compuesto. Dependiendo del caso de uso y de cómo se defina, probablemente sólo se desasociará la etiqueta de las conversaciones en las que estuviera asociada y del buzón en el que se creó, y se borrará.
- Podríamos haber usado la especialización para distinguir entre los mensajes que son respuesta (y, por lo tanto, tienen 1 en la multiplicidad de respondeA) y los que no son respuesta (y, por lo tanto, son origen de 1 y sólo una conversación):



- Porque según la definición de esta asociación derivada, cualquier conversación está formada por su mensaje origen y por las respuestas a cualquier mensaje de la conversación. Por lo tanto, como mínimo existirá el mensaje que es el origen de la conversación.
- De entrada, el atributo debería ser multivaluado: `etiqueta:String[*]`. Además, no tendríamos las etiquetas como entidades; así, por ejemplo, estaríamos dando a entender que no tiene sentido gestionar las etiquetas: crearlas, enumerarlas, etc. Dos conversaciones podrían tener una misma etiqueta sólo por coincidencia.

9.

- En las pantallas figura la dirección de correo electrónico del usuario. Habría que añadirla como atributo de la clase *Usuario*.
- En las pantallas, cada cuenta, además del número, tiene un nombre. Por lo tanto, habría que añadir el atributo `nombre:String` a la clase *Cuenta*.
- En las pantallas, las tarjetas tienen, además del saldo, un límite. Habría que añadirlo como atributo de la clase *TarjetaCredito*.
- Los préstamos de las pantallas pueden ser hipotecas, pero también otros tipos. Seguramente, la clase *PrestamoHipotecario* del modelo del dominio se debería denominar *Prestamo*, ya que no sólo representa préstamos hipotecarios. Si más adelante se viera que hay motivos para una mayor especialización, se podrían tener subclases de préstamo.
- Los préstamos tienen, además del saldo, una cuota, que falta como atributo de la clase *Prestamo* mencionado antes.
- Los movimientos de las pantallas tienen dos fechas: una de operación y otra de valor. Por lo tanto, en lugar del atributo `fecha`, la clase *Movimiento* debería tener los atributos

- fechaOperacion* y *fechaValor*. Aunque las pantallas enumeran un saldo después de cada movimiento y éste se podría representar en el modelo del dominio, este saldo se puede derivar y, por lo tanto, no se puede considerar incorrecto el hecho de que no aparezca explícitamente como atributo de la clase *Movimiento*.
- Hay mucha información en el modelo del dominio que no aparece en las pantallas. Seguramente no es incorrecta, pero se debería verificar con el resto de las pantallas. Es el caso, por ejemplo, de la mayoría de los atributos de la clase *Usuario*, de varios atributos de *Cuenta*, de la cuenta de cargo de los créditos, etc.

Ejercicios de autoevaluación

1. El lenguaje UML es el lenguaje estándar para crear modelos visuales de software. (Podéis ver el subapartado 1.2.1).
2. Los tres usos principales para el lenguaje UML son: como lenguaje para diseñar un croquis, como lenguaje para realizar un plano y como lenguaje de programación. (Podéis ver el subapartado 1.2.1).
3. El diagrama de casos de uso complementa, pero en ningún caso sustituye, la descripción textual de los casos de uso, dado que no incluye información sobre cuál es el comportamiento del sistema. (Podéis ver el subapartado 2.1).
4. El punto de extensión es un texto que identifica en qué punto del escenario principal se produce el acontecimiento que provoca el comportamiento alternativo. (Podéis ver el subapartado 2.1.6).
5. Lo podemos hacer mediante la condición de guarda y, opcionalmente, una decisión (que se representa mediante un rombo). (Podéis ver el subapartado 2.2.1).
6. Tenemos toda una serie de requisitos, como la usabilidad o la arquitectura de información, que, al ser los casos de uso independientes de la interfaz de usuario, no quedan recogidos en este modelo. (Podéis ver el subapartado 3).
7. Se denominan *casos de uso esenciales* los casos de uso que describen la interacción entre actores y sistema de manera independiente de la tecnología y de la implementación. En contraposición, denominaremos *casos de uso concretos* a aquellos que tienen en cuenta la tecnología y la implementación. (Podéis ver el subapartado 3.1).
8. El objetivo de los modelos de las pantallas es dejar claro qué información se muestra, la distribución de la información en la pantalla, qué acciones puede tomar el usuario a partir de la información mostrada, cuál es el proceso que sigue el usuario para completar el caso de uso. (Podéis ver el subapartado 3.2).
9. El mapa navegacional es un modelo que nos da información sobre el flujo entre pantallas que puede seguir el usuario. (Podéis ver el subapartado 3.2.1).
10. Un modelo del dominio es una representación de clases conceptuales del mundo real en un dominio de interés. (Podéis ver el subapartado 4).
11. Coad propone: participantes, lugares y cosas, roles, momentos o intervalos y descripciones. (Podéis ver el subapartado "Identificación de clases del dominio" dentro del 4.2.2).
12. Usar la terminología que emplean las personas y los expertos del dominio, excluir aspectos irrelevantes del dominio y no añadir nada que no esté realmente en el dominio que analizamos. (Podéis ver el subapartado "Nomenclatura de clases" dentro del 4.2.2).
13. Indica que es opcional (mínimo 0 valores) y multivalorado (máximo * valores). (Podéis ver el subapartado 4.3.1).
14. A pesar de que tanto los atributos como los extremos de asociación son propiedades de la clase, la notación de las asociaciones es más visual y, por lo tanto, la preferimos a la de los atributos para representar la relación entre dos clases. (Podéis ver el subapartado 4.4.2).
15. Debemos plantearnos la especialización si hay subconjunto de las instancias de la clase tal que tiene atributos o asociaciones adicionales de interés que el resto de las instancias de la clase examinada no tienen, o se comportan diferente de manera relevante. (Podéis ver el subapartado "Identificación de la herencia: generalización y especialización" dentro del 4.5.2).

16. Son el conjunto de reglas que deben cumplir los datos que almacena un sistema para evitar situaciones en las que la información es contradictoria o incompleta. (Podéis ver el subapartado 4.6.1).

17. Los atributos o asociaciones derivados son aquellos cuyo valor se puede deducir a partir de información ya modelizada. (Podéis ver el subapartado 4.6.2).

18. Los valores de un tipo de datos, a diferencia de los objetos, no tienen identidad única. Por lo tanto, se comparan por valor y no por identidad. (Podéis ver el subapartado 4.7.1).

Glosario

bifurcación *f* Elemento gráfico del diagrama de actividades de UML representado como una línea horizontal gruesa con un flujo de entrada y múltiples flujos de salida. Cuando se llega a una, todos los flujos de salida se producen de manera paralela.

cardinalidad (de un atributo) *f* Número de valores que una determinada instancia tiene en un atributo.

cardinalidad (de una asociación) *f* Número de instancias que una determinada instancia tiene asociadas por medio de una asociación concreta.

carril *m* Elemento gráfico del diagrama de actividades de UML que permite organizar las actividades según qué actor del proceso las lleva a cabo.

caso de uso concreto *m* Caso de uso que describe la interacción entre actores y sistema teniendo en cuenta la tecnología y la implementación. Pueden contener, por ejemplo, detalles sobre el modo como el actor usa la interfaz ofrecida por el sistema.

caso de uso esencial *m* Caso de uso que describe la interacción entre actores y sistema de manera independiente de la tecnología y de la implementación.

clave (de una clase de dominio) *f* Atributo o conjunto de atributos que identifica las instancias de la clase de manera única, de tal manera que no puede haber más de una instancia con los mismos valores en ese u otros atributos.

composición *f* Asociación con un fuerte sentido todo-parte, de manera que si destruimos una instancia compuesta, todos sus componentes también se destruyen, no puede haber instancias de la clase componente que no formen parte de una única instancia compuesta y, finalmente, una instancia componente no puede cambiar de un compuesto a otro.

contrato (de una operación) *m* Documentación detallada de una operación que incluye su firma y las precondiciones y poscondiciones.

decisión *f* Elemento gráfico del diagrama de actividades UML representado como un rombo y que representa una toma de decisión.

diagrama de actividades *m* Diagrama estándar de UML utilizado para representar procesos.

diagrama de estados *m* Diagrama estándar de UML utilizado para modelizar estados y transiciones entre éstos.

diagrama de estructura compuesta *m* Diagrama estándar de UML utilizado para modelizar la estructura interna en tiempo de ejecución de una clase o componente.

diagrama de objetos *m* Diagrama estándar de UML utilizado para representar objetos.

diagrama de casos de uso *m* Diagrama estándar de UML utilizado para modelizar los casos de uso del sistema, los actores y las relaciones entre éstos.

diagrama de clases *m* Diagrama estándar de UML utilizado para modelizar las clases de objetos y sus relaciones.

diagrama de colaboración *m* Nombre del diagrama de comunicación en la versión 1 de UML.

diagrama de componentes *m* Diagrama estándar de UML utilizado para modelizar los componentes que forman parte de un sistema.

diagrama de comunicación *m* Diagrama estándar de UML utilizado para modelizar la interacción entre varios objetos haciendo énfasis en el aspecto estructural.

diagrama de despliegue *m* Diagrama estándar de UML utilizado para modelizar la distribución física de los diferentes artefactos de software en tiempo de ejecución.

diagrama de paquetes *m* Diagrama estándar de UML utilizado para representar los paquetes y las dependencias entre éstos.

diagrama de secuencia *m* Diagrama estándar de UML utilizado para modelizar la interacción entre varios objetos haciendo énfasis en la secuencia temporal.

diagrama de tiempo *m* Diagrama estándar de UML utilizado para modelizar el comportamiento de los objetos a lo largo del tiempo, poniendo énfasis en las restricciones temporales.

diagrama de visión general de interacción *m* Diagrama estándar de UML que combina los diagramas de actividades y los de secuencias.

e26eración *f* Tipo de datos en el que el conjunto de los valores es una lista finita de valores que se e26era en la definición del tipo de datos.

firma (de una operación) *f* Nombre e información sobre los datos que recibe (parámetros de entrada) y la información que devuelve.

fusión *f* Elemento gráfico del diagrama de actividades de UML representado como un rombo con varios flujos de entrada y sólo uno de salida. El flujo de salida se produce cuando se llega por alguno de los flujos de entrada.

herencia dinámica *f* Herencia en la que una instancia de una clase, a lo largo de su vida, puede cambiar de una clase de herencia a otra.

herencia overlapping *f* Herencia en la que una misma instancia de la superclase puede pertenecer a más de una subclase a la vez.

mapa navegacional *m* Modelo que documenta los flujos entre pantallas de la interfaz gráfica de usuario. Se representa mediante un diagrama de estados UML en el que cada pantalla es un estado y cada transición representa la transición entre pantallas producida por un acontecimiento, típicamente un gesto del usuario.

multiplicidad *f* Restricción que indica qué cardinalidades son válidas para un atributo o asociación.

navegabilidad *f* Propiedad de una asociación binaria que provoca que sólo esté definida en un sentido y que, por lo tanto, sólo defina una propiedad en una de las dos clases que participen en ella. La otra clase continuará ignorando la existencia de la asociación.

object constraint language *m* Lenguaje formal estándar de restricciones: lenguaje estándar de UML para la expresión de restricciones que se puede usar, entre otras cosas, para escribir precondiciones y poscondiciones de los contratos de las operaciones.

Sigla **OCL**

Object Management Group *m* Consorcio americano sin ánimo de lucro, creado en 1989, que tiene por objetivo la estandarización y promoción de la modelización orientada a objetos.

Sigla **OMG**

poscondición (de una operación) *f* Obligación a la que se compromete el sistema cuando se invoca una operación a modo de condición que se compromete a ser cierta una vez ejecutada la operación.

precondición (de una operación) *f* Condición que se debe satisfacer necesariamente en el momento de invocar la operación. Si la condición no es cierta, el sistema rechaza la petición y la operación no se ejecuta.

propiedad (de una clase) *f* Nombre genérico para referirse a atributos y a extremos de asociación de la clase.

punto de extensión *m* Punto concreto en la secuencia de pasos de un escenario (principal o extensión) de un caso de uso al cual se da un nombre para que los casos de uso que lo extienden puedan definir aquel punto como el punto en el que entra en juego la extensión. En desuso.

tipos de datos *m* Conjunto de valores para los que la identidad única no tiene sentido, sino que se comparan por valor.

unión *f* Elemento gráfico del diagrama de actividades de UML representado como una línea horizontal gruesa con múltiples flujos de entrada y un flujo de salida. El flujo de salida sólo se produce cuando han llegado todos los flujos de entrada.

Bibliografía

Larman, C. (2005). *Applying UML and patterns: an introduction to object-oriented analysis and design and iterative development*. Prentice Hall.

Larman cubre, en esta obra, el análisis y el diseño de sistemas informáticos usando el lenguaje UML. Da un enfoque ágil y otro de UP, y el libro está escrito de manera iterativa e incremental. Cubre todo lo que se explica en este módulo y bastante más, puesto que entra en detalles de diseño e implementación.

Fowler, M. (2004). *UML distilled: A brief guide to the standard object modeling language*. Addison-Wesley.

Esta obra es una muy buena introducción corta al UML. Tiene un enfoque más pragmático que normativo y resulta muy aclaratoria. Muestra el uso de todos los diagramas, con ejemplos, y da consejos personales del autor sobre cómo y cuándo hay que utilizar cada parte de la especificación. Evidentemente, no entra en detalles.

Bibliografía complementaria

Cockburn, A. (2001). *Writing Effective Use Cases*. Addison-Wesley.

A pesar de que no profundiza en el uso del UML, Cockburn continúa siendo la bibliografía de referencia básica en cuanto a casos de uso. Respecto a este módulo, es especialmente interesante la discusión que plantea sobre relaciones entre casos de uso y sobre el alcance y nivel de los casos de uso.

VV.AA. (2010). *Unified Modeling Language™ (UML®)*. Object Management Group.

Ésta es la especificación de UML. A pesar de su lenguaje formal y normativo, es, evidentemente, la fuente más fiable en caso de duda sobre el lenguaje UML.

Booch, G.; Rumbaugh, J.; Jacobson, I. (2005). *The unified modeling language user guide*. Addison-Wesley.

En esta obra los autores de UML escriben una guía de uso del lenguaje de carácter didáctico, más que de referencia. La obra se centra en el lenguaje UML, pero nos da consejos y sugerencias sobre cuándo hay que usar una parte del estándar u otra.

Coad, P.; Lefebvre, E.; De Luca, J. (1999). *Java Modeling in Color With UML: Enterprise Components and Process*. Prentice Hall.

A pesar de que ya hace bastantes años que fue publicado y de que su título da a entender que se trata de un libro técnico, esta obra trata, básicamente, sobre la modelización de sistemas de información usando orientación a objetos. Propone un enfoque basado en un modelo de análisis que se reutiliza adaptándolo a las necesidades concretas de cada problema.

Referencias bibliográficas

Fowler, M. (1997). *Analysis Patterns. Reusable Object Models*. Addison-Wesley.

Fowler propone el uso del concepto de patrón para reutilizar soluciones de análisis sobre todo en cuanto a la modelización del dominio. Aparte de proporcionar algunos patrones realmente útiles a la hora de modelizar sistemas de información, también es interesante ver los ejemplos de modelos que elabora aplicando estos mismos patrones.

Génova, G.; Lloréns, J.; Martínez, P. (2001). "Semantics of the minimum multiplicity in ternary associations in UML". *The 4th International Conference on the Unified Modeling Language*.

<http://www.ie.inf.uc3m.es/ggenova/pub-uml2001.html>

En este artículo los autores señalan que la especificación de UML es ambigua a la hora de explicar la cardinalidad de las asociaciones no binarias y proponen una clarificación.

