UT03 -Metodología OO - UML - Diagramas

ÍNDICE

Índice de contenido

Versión del documento

19/11/11 Creación del documento
28/11/11 Notación básica diagramas
12/12/11 Diagrama de clases
08/01/12 Relaciones entre clases
23/01/12 Agregaciones, composiciones y compuestos
31/01/12 Interfaces,Visibilidad, Ejercicios 6 y 7
20/02/12 Ejercicio 8 y Casos de Uso
27/02/12 Diagrama de estados
04/03/12 Subestados, mensajes y señales

1 INTRODUCCIÓN

En la unidad anterior nos hemos centrado en las metodologías de desarrollo estructuradas. Éstas surgieron progresivamente según iban siendo necesitadas.

1. Programación *ad hoc.* El problema no era apenas analizado y se programa directamente sin pasos intermedios entre la aparición del problema a resolver y la elaboración del programa que lo resuelve.
2. Algoritmos. Se representa el problema a resolver y el proceso de resolución mediante pseudocódigo, diagramas de flujo u otras herramientas que representen el algoritmo que más tarde se programará. Estas técnicas aparecen tras los lenguajes de programación (que, a su vez, son evoluciones del lenguaje máquina, muy cercano al hardware) y están muy relacionadas con esos lenguajes y con su arquitectura y la del ordenador. Surge la programación estructurada.
3. Técnicas de diseño estructurado. Apoyando la programación estructurada y como estudio previo, aparecen técnicas y métodos basados en gráficos.
4. La preparación y organización de las fases de diseño conducen a la aparición del análisis estructurado.
5. Metodologías estructuradas. Cubren todos los aspectos del ciclo de vida, desde el surgimiento de la necesidad hasta la retirada del producto y los procesos relacionados con la generación del producto software incluida la elaboración de la documentación asociada, la planificación y organización del proyecto y otras.

Cada paso supone un avance que toma como base el anterior estado, lo que hace que las Metodologías estructuradas estén, de forma indirecta, condicionadas por la arquitectura hardware de los ordenadores y por la de los lenguajes estructurados típicos.

Hemos visto también que las metodologías estructuradas separan funciones y datos.

Sin embargo, las metodologías orientadas a objetos no tienes esos condicionantes ni hacen tal separación, sino que:

1. Son independientes de las arquitecturas
2. Aúnan o integran datos y procesos.

2 UNIFICACIÓN DE LOS MÉTODOS OO

Las ideas principales del análisis OO, presentes en la mayoría de los métodos son

1. La idea o concepto de clase
2. Las relaciones entre clases, particionando en subsistemas utilizando los conceptos de generalización, especialización y agregación
3. La importancia de recoger los requisitos del sistema a partir del estudio de la integración entre usuarios y sistema.

A mediados de los 90, había unos 50 métodos diferentes de análisis orientado a objetos. Sin embargo, las diferencias entre métodos en cada vez menores. Por citar alguno, el método de Boosch, el OMT (Object Modeling Technique) de Rumbaugh y el OOSE (Object-Oriented Software Engineering) de Ivar Jacobson son sólo tres de ellos.

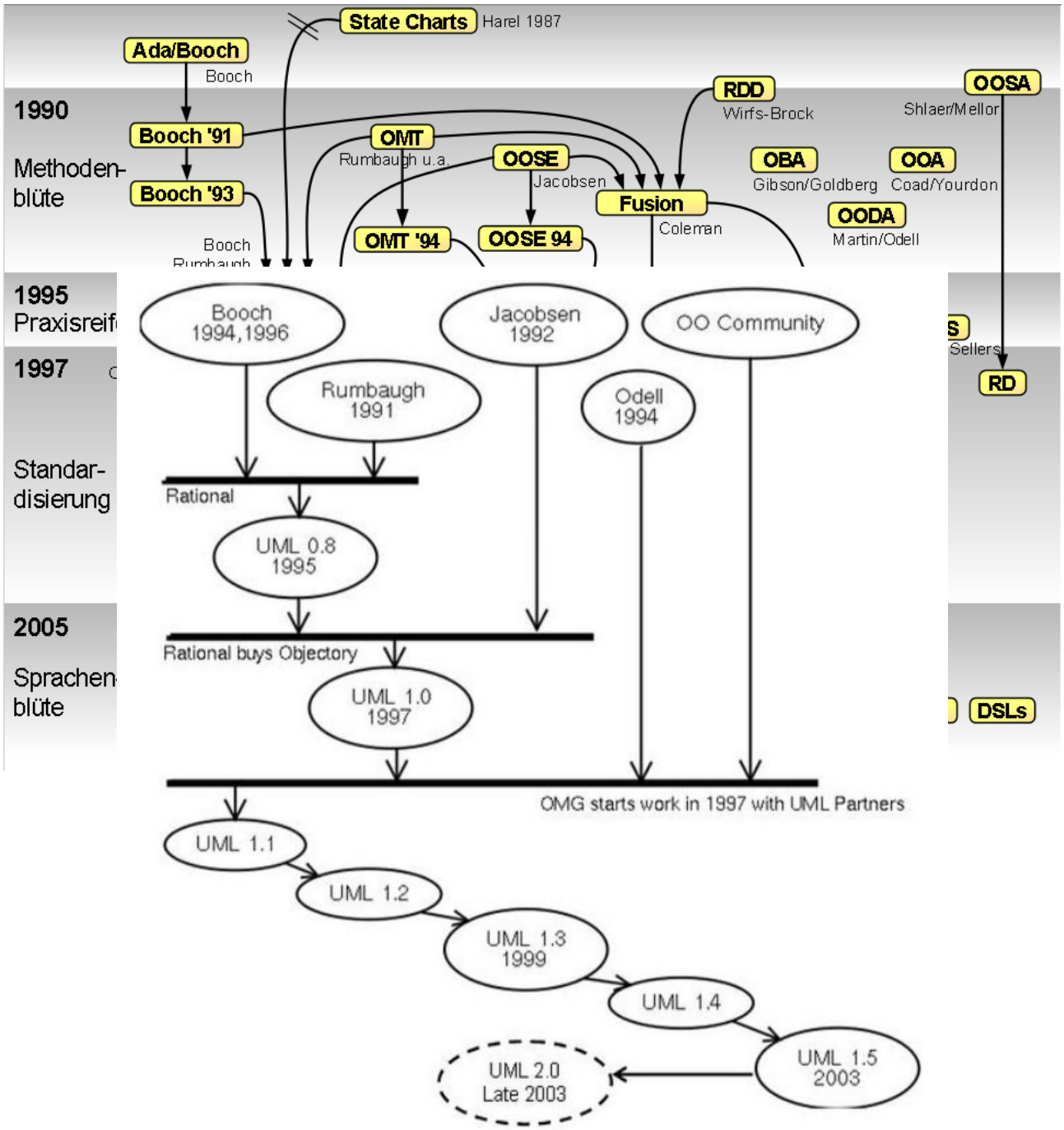

Ilustración 2: Evolución histórica de métodos OO

James Rumbaugh, Grady Boosch e Ivar Jacobson unieron sus fuerzas y con el patrocinio de la empresa Rational Software de Cupertino (California, USA) ofrecieron el UML o lenguaje unificado de modelado para el desarrollo Orientado a Objetos.

Se fijaron 4 objetivos:

1. Representar sistemas completos por conceptos de objetos.
2. Establecer una relación explícita entre los conceptos y los artefactos ejecutables.
3. Tener en cuenta los factores de escala inherentes a los sistemas complejos y críticos.
4. Crear un lenguaje de modelado utilizable tanto por los humanos como por las máquinas.

UML ha sufrido diversas modificaciones desde que fue propuesto por el OMG (Object Management Group) en septiembre de 1997, aunque ya fue respaldado por la ISO con el estándar ISO/IEC 19501 en 1995. La última versión es la 2.

UML es un lenguaje de modelado que incluye notación gráfica usada para crear un modelo abstracto de un sistema llamado “modelo UML”.

UML consiste en:

- *Diagramas de estructuras*: enfatiza QUÉ COSAS deben estar en el sistema que está siendo modelado.
- *Diagramas de comportamiento*: enfatiza QUÉ DEBE SUCEDER en el sistema que se esta modelando
- *Diagramas de Interacción*: un subconjunto de diagramas de comportamiento, enfatizan el FLUJO DE CONTROL Y DATOS ENTRE LAS COSAS que aparecen en el sistema que está modelándose.

UML no guía al desarrollador en la forma de realizar el análisis y diseño orientado a objetos ni le indica qué proceso de desarrollo adoptar, sólo proporciona un lenguaje común de representación. En consecuencia, los metodólogos definirán técnicas, modelos y procesos de desarrollo para la creación de software, que ahora pueden usar éste lenguaje.

UML describe LO QUE supuestamente hará el sistema (una vez desarrollado). NO dice CÓMO se implementará ese sistema.

A pesar de su status de estándar ampliamente reconocido y utilizado, UML siempre ha sido muy criticado por su carencia de una semántica precisa, lo que ha dado lugar a que la interpretación de un modelo UML no pueda ser objetiva. Otro problema de UML es que no se presta con facilidad al diseño de sistemas distribuidos. En tales sistemas cobran importancia factores como transmisión, serialización, persistencia, etc. UML no cuenta con maneras de describir tales factores. No se puede, por ejemplo, usar UML para señalar que un objeto es persistente o remoto, o que existe en un servidor que corre continuamente y que es compartido entre varias instancias de ejecución del sistema analizado. Sin embargo, UML sí acepta la creación de nuestros propios componentes para

este tipo de modelado

UML propone una serie de diagramas gráficos para representar el problema a resolver. Éstos diagramas son:

- Diagramas estructurales:
  - de clases
  - de componentes
  - de estructura compuesta (UML 2.0)
  - de "despliegue" (deployment).
  - de paquetes
  - de objetos
- Diagramas de comportamiento
  - de actividad y de interacción
  - de colaboraciones o de comunicación
  - de secuencias
  - de estados
  - de tiempos
  - de casos de uso

En general un modelo UML estará hecho de uno o más diagramas. Un diagrama representa cosas y las relaciones entre cosas de modo gráfico. Estas cosas pueden ser representaciones de objetos del mundo real, construcciones de software puro o la descripción del comportamiento de algún otro objeto. Es normal que una misma cosa se muestre en múltiples diagramas. En cada diagrama se presentará un aspecto particular o "vista" de la cosa que se modela.

A continuación veremos algunos aspectos teóricos básicos de la Orientación a Objetos.
Posteriormente pasaremos a describir los distintos diagramas que componen el UML.

3 LA ORIENTACIÓN A OBJETOS

El objetivo de la Orientación a Objetos es elaborar un programa o producto software mediante una serie de operaciones iterativas que se encaminan a:

1. Comprender la realidad del problema que se quiere resolver, qué parte de ese problema será resuelta por el producto. Será preciso descomponer el problema en subproblemas más sencillos, con objeto de facilitar su comprensión.
2. Buscar soluciones a los subproblemas parciales.
3. Composición de las diferentes soluciones parciales que se han encontrado a los distintos subproblemas en una solución general.

El método clásico consistía en identificar las funciones (procesos) principales del sistema y descomponerlas en subfunciones, hasta llegar a un nivel en el que los subproblemas son suficientemente pequeños como para ser resueltos por un lenguaje de programación, usando variables, secuencias, decisiones, iteraciones y funciones+procedimientos[1]. El método clásico aisla funciones. De modo que si en el mundo real cambia alguna de esas funciones, basta retocar esa función en el código. Calcular_nómina es una función del programa de gestión del departamento de recursos humanos de una empresa. Si la legislación relativa a las retenciones cambia tras la victoria

[1] Una función es un bloque de código que retorna un valor. Un procedimiento es un bloque de código que no retorna un valor.

del PP, basta retocar esa función "Calcular_nómina" para adaptarse a las nuevas leyes, sin afectar al resto del programa.

El problema del método clásico es que a menudo una variación en la realidad representada, supone un cambio más profundo, que afecta a la estructura funcional del programa.

La orientación a objetos supone un avance en éste sentido. Acerca la representación informatizada de la realidad a la realidad misma haciéndola más fiel. No se basa únicamente en lo que hace el sistema, sino también en cómo es, cuáles son sus características definitorias.

Todo el modelo orientado a objetos se ajusta a los 4 objetivos que se fijaron en sus inicios y se basa en 5 elementos:

1. Objeto
2. Clase
3. Mensaje
4. Herencia
5. Polimorfismo

3.1 ELEMENTOS DEL MODELO DE OBJETOS

El desarrollo orientado a objetos considera a un programa como una colección de agentes autónomos llamados **Objetos**. Mediante la interacción de los objetos avanza la ejecución del programa.

3.1.1 Definición de Objeto

Un objeto es una representación de un ente del mundo real. Mediante un objeto el ente queda representado por:

1. El **estado** del objeto. El estado del objeto queda definido por el **valor de** una serie de **atributos o propiedades**.
2. El **comportamiento** del objeto. Representa las capacidades del objeto para actuar, para responder a la interacción con otros objetos, para operar. Estas capacidades se incluyen en el objeto con el nombre de **métodos u operaciones**.

Por ejemplo. Para representar un objeto triángulo "T", podrían incluirse

1. Como valores de **estado**: longitud del lado1, longitud del lado 2, longitud del lado3 y valor del ángulo1, 2 y 3. Por ejemplo: 3mm, 3mm y 3mm para los lados y 60º, 60º, 60º para un triángulo equilátero particular. Otro atributo puede ser la posición sobre la horizontal en grados (éste triángulo puede presentarse girado). Por ejemplo T podría estar girado en un instante dado 12º sobre la horizontal.
2. Como **comportamiento**, podría incluirse el método: girarTriángulo(grados). El método anterior indica que un triángulo puede girarse, mediante la invocación del método con un valor del argumento en grados. A esto se le llama también enviar un mensaje al triángulo.

El hecho de representar la realidad mediante objetos, permite aislar unos de otros, mientras su interfez externa (métodos) se mantenga. **Cualquier cambio en un objeto no repercutirá en otros**

objetos. De este modo se facilita la introducción de cambios y mejoras en el software. Sólo habrá que retocar determinados objetos, permaneciendo el resto inalterados.

En el mundo real, en el caso de un ordenador, por ejemplo, cuando se avería la pantalla, basta con reemplazarla por otra, no es necesario retocar o modificar el ordenador mismo.

Se llama **encapsulamiento** a la combinación de datos y comportamiento de los objetos, mientras que, por el contrario, se oculta su implementación.

Así, la forma en que un objeto desempeña una tarea concierne sólo a dicho objeto y no a otros objetos aunque interaccionen con éste. El objeto ofrece la misma imagen al exterior aunque se varíe su contenido.

La mayoría de la gente ve una televisión sin importarle demasiado como se hace para que la imagen llegue a él y se muestre en la pantalla. Simplemente hace lo que tiene que hacer y el cómo lo hace permanece oculto. Nosotros sólo vemos y conocemos "el interfaz" esto es, los botones del mando a distancia.

3.1.2 Comunicación entre objetos: mensajes

El avance de un programa depende de la colaboración entre los objetos que lo componen. Es necesario que los objetos se comuniquen.

Denominaremos **mensaje** hacia un objeto a la llamada o invocación de un método de ese objeto desde otro objeto o desde dentro de sí mismo.

Un objeto se activa mediante la invocación de un método propio al recibir un mensaje. Los métodos se suelen invocar usando su nombre y, opcionalmente, recibiendo unos valores concretos llamados argumentos; Ej: `girarTriángulo(10 grados)`. Se debe indicar a quién se envía el mensaje, esto es, a qué objeto concreto.

3.1.3 Clasificación

Cada objeto es un ejemplo que pertenece a una clase determinada. Se dice que un objeto concreto es una instancia de una clase. La clase es una generalización del concepto objeto.

En el ejemplo del triángulo anterior, el objeto triángulo con las medidas concretas propuestas, sería una instancia de la clase genérica "triángulos", a la que pueden pertenecer múltiples elementos o instancias.

Dos **instancias** de una misma clase se diferencian entre sí por el **valor de sus atributos**, esto es, por su **estado**.

En el lenguaje natural escrito o hablado puede deducirse cuál es la clase porque suele estar precedida por un artículo que determina que se trata de un concepto genérico aplicable a muchos casos particulares: Ej: "los triángulos".

A veces se hace referencia en el lenguaje natural a las clases usando el artículo indeterminado un(a). En lenguaje metemático también: "Sea 't' un triángulo..." "t" es la instancia y "un triángulo" es la clase.

En la definición de una clase se indica qué partes pueden ser vistas por otros objetos (**interfaz del objeto**) y cuáles quedan ocultas y forman parte de la **implementación** del mismo.

La ocultación de la implementación contribuye al "encapsulamiento".

3.1.4 Herencia

Las clase pueden organizarse en un árbol de herencia jerárquico. Las clases que se encuentran en los niveles bajos del árbol (clases descendientes) pueden tener asociados atributos (propiedades) y métodos que ya existen en las clases superiores o ancestros.

Para dos clases organizadas mediante una relación de herencia, una de ellas es la **clase padre** y la otra, la que "hereda" propiedades y comportamientos sería la **clase hija**.

En el ejemplo del triángulo T, éste pertenece (es un ejemplo o una instancia) de la clase triángulos, que es una clase hija de la clase padre polígonos.

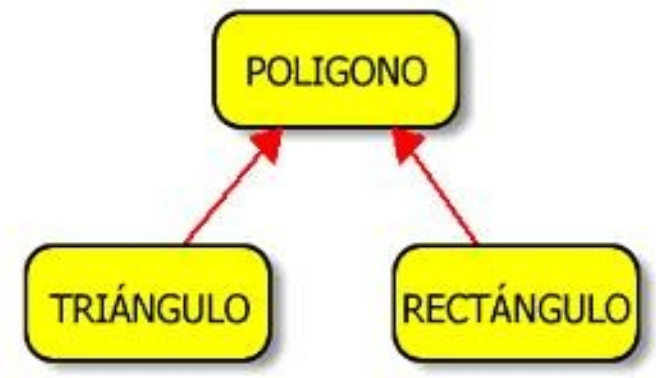

Ilustración 3: Ejemplo de relación de herencia

Ejercicio 1

Busca un ejemplo en el mundo real de clases y subclases relacionadas por herencia y represéntalos con rectángulos a modo del ejemplo de la Ilustración 3

Como se ha comentado, la clase hija "hereda" propiedades y comportamientos de la clase padre, lo que facilita la reutilización de desarrollo anteriores para aplicaciones completamente nuevas. La clase padre recibe a veces el nombre de "superclase" y "clase base" y la hija el de "subclase" o "clase derivada".

Las subclases aprovechan las características de sus superclases para construir las suyas propias. Además puede definir características (atributos o métodos) propias.

Una clase "Electrodoméstico" poseerá atributos y métodos comunes a cada una de sus clases derivadas. Contará por ejemplo con los atributos interruptor y cableElectrico. Con los métodos encender() y apagar(). Todas las clases de electrodomésticos, por ejemplo, Lavadora, poseerán

(heredarán) esos atributos y métodos. A partir de ahí, en Lavadora pueden definirse atributos y métodos propios de las lavadoras, aparte de esos comunes. Si los métodos comunes se redefinen en la clase derivada, dará lugar al polimorfismo, que se ve en el siguiente punto.

La herencia es **múltiple** cuando la clase derivada hereda de varias clases base.

3.1.5 Polimorfismo

El **polimorfismo** consiste en elaborar diferentes operaciones en respuesta a un mismo mensaje.

Como hemos visto en la herencia, las clases hijas "heredan" métodos de sus clase padre. Un mismo método (con el mismo nombre) existirá en las clases padre e hija, pero su implementación puede ser diferente.

Supongamos la clase "polígonos" y su subclase "Triángulos". El método "girar()" se llamará igual, pero el mecanismo finalmente empleado para girar un triángulo puede ser distinto al usado para otros polígonos.

Resumiendo:

1. La subclase **puede modificar** la implementación de las operaciones heredadas.
2. En ese caso, el mismo nombre de operación puede designar **implementaciones distintas**.
3. Así, objetos de distintas clases pueden tener operaciones con el mismo nombre, pero con una implementación diferente, por lo tanto: **ante un mismo mensaje, responderán realizando operaciones diferentes**. (El código fuente o definición es distinto, pero el nombre de función o método es el mismo).

Por poner otros ejemplos: puede "abrirse()" una puerta, una ventana, un periódico, un regalo o una cuenta de banco. En cada caso se realiza una operación diferente. En la orientación a objetos cada clase sabrá cómo realizar cada operación según el caso. Esto es el polimorfismo.

4 DIAGRAMAS DE CLASES

4.1 Representando clases

Una clase se representa por un cuadrilátero dividido en secciones apiladas verticalmente. La superior, contiene el nombre de la clase, la siguiente los nombres de los atributos o propiedades que determinarán el estado de cada instancia y la siguiente los métodos de la clase, lo que puede hacer.

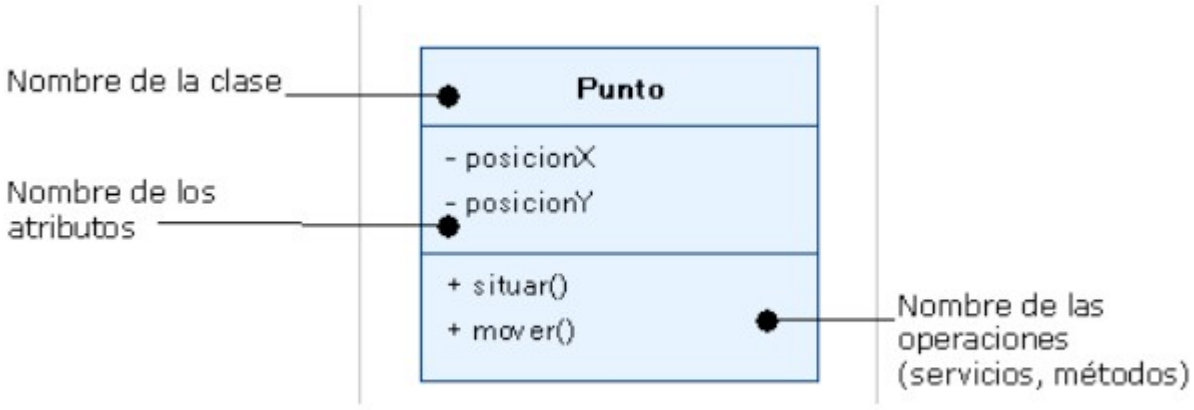

Ilustración 4: Notación básica para la clase "Punto"

Ejercicio 2

En el ejercicio anterior, ilustra la jerarquía de objetos con un diagrama. Para cada objeto indica posibles atributos y métodos.

4.1.1 Convenios de nomenclatura

En la clase "**`Coche`**":

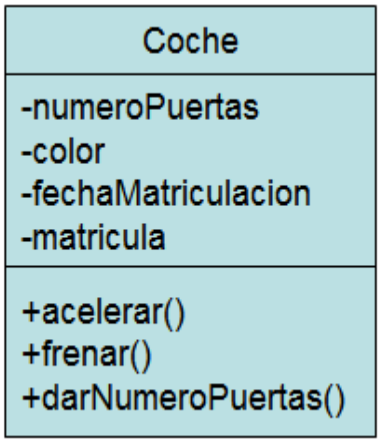

Ilustración 5: Clase "Coche"

Los **métodos públicos**, lo que forman parte del Interfaz del objeto, se indican precedidos por el signo "+". Los que forman parte de la implementación son precedidos por el signo "-".

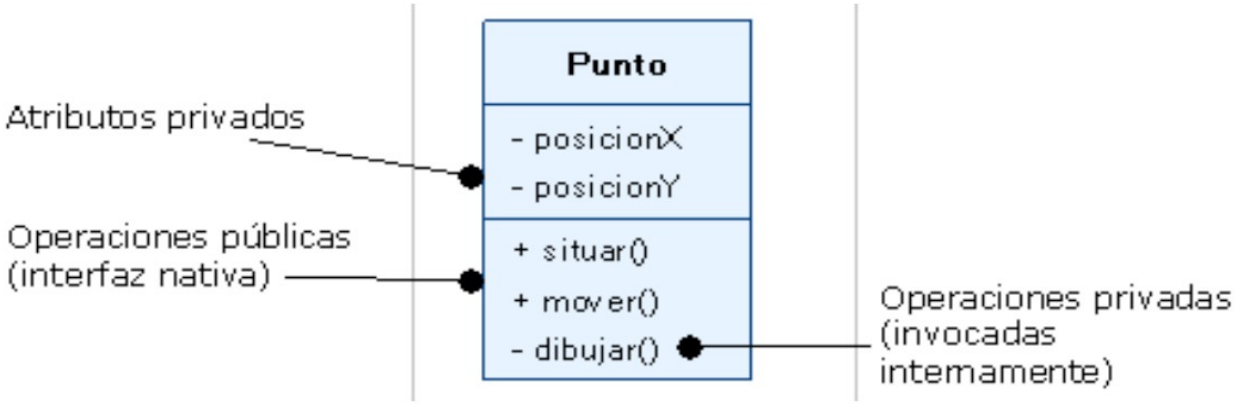

Ilustración 6: Métodos públicos y privados

Por convenio las clases se representan con la primera letra en mayúsculas y el resto en minúsculas. Si tienen varias palabras se representan seguidas. Para no convertirlo en un vocablo ilegible, se pone en mayúscula la primera letra en la segunda y sucesivas palabras. No se usarán caracteres regionales como eñe o vocales acentuadas.

`LavadoraIndustrial` es un nombre de una clase.[2]

Los atributos se representan de idéntica manera pero comenzando en en minúsculas la primera palabra.

`numero` es un atributo de una sola palabra

`numeroMatricula` es un atributo de dos palabras

`numeroMatriculaHomologado` es un atributo de tres palabras

UML da la opción de indicar el **tipo de datos** para cada atributo. Existen los tipos:

- `string` para cadenas de caracteres,
- `integer` para números enteros
- `float` para números con decimales
- `boolean` para variables que sólo toman los valores verdadero o falso.
- tipos enumerados

Para indicar el tipo, se pone éste tras el nombre de la variable separado por dos puntos.

```
numeroMatricula:string
```

Un atributo puede tomar un valor predeterminado. Se separa por el símbolo "=".

```
numeroMatricula:string="0000AAA"
```

Las operaciones o métodos se nominan de la misma manera en cuanto al uso de mayúsculas y minúsculas. Pueden recibir valores como parámetros, que se indican entre paréntesis. Los

[2] UML sugiere que se escriba el nombre de clase en negrita.

parámetros se definen como los atributos, con su tipo. Si no tiene parámetros se mantienen los paréntesis tras el nombre del método. Hay métodos estilo "función" que retornan un valor al finalizar. En los métodos puede mostrarse el tipo del valor retornado al final.

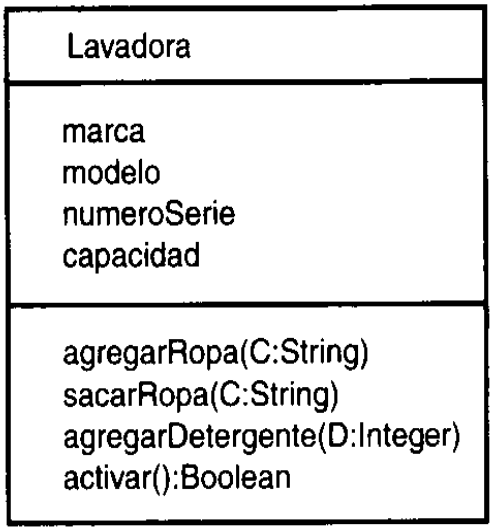

Ilustración 7: Atributos y métodos con sus tipos.

En la práctica una misma clase no se dibuja sólo una vez. Suelen dibujarse muchas veces. Para no repetir cada vez todos los atributos y métodos, se permite que se represente con los huecos de atributos y métodos vacíos.

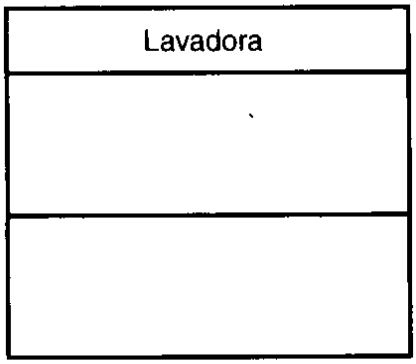

Ilustración 8: Clase Lavadora

En ocasiones será bueno mostrar sólo algunos atributos o métodos. Se muestra "..." para indicar que hay más, que no se desean representar en ese momento.

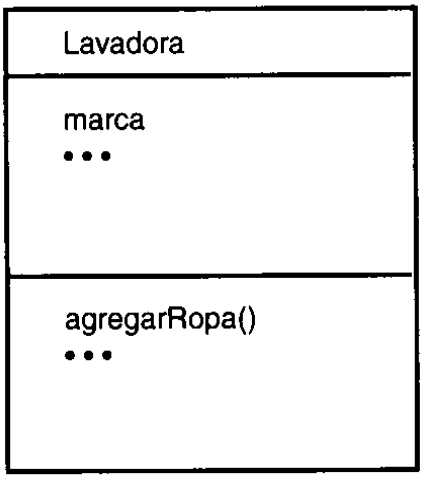

Ilustración 9: Clase Lavadora

4.1.2 Estereotipos, responsabilidades, restricciones y notas adjuntas

Pueden comentarse atributos y métodos usando "**estereotipos**" englobados por los símbolos << y >>. Las clases pueden contener una descripción en un apartado propio bajo los métodos llamada "**responsabilidad**".

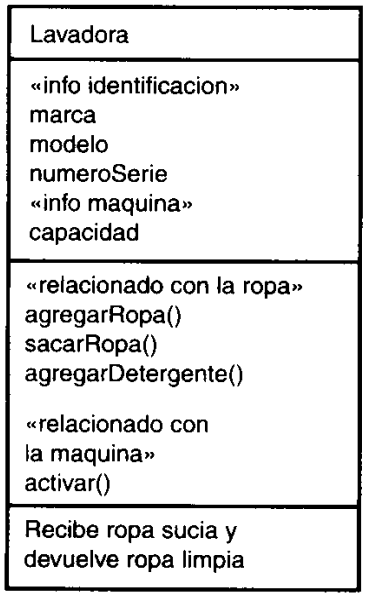

Ilustración 10: Estereotipos y responsabilidad de la clase "Lavadora"

Una manera más formal es la "**restricción**". Se pone con un texto libre y entre llaves a un lado del rectángulo de la clase.

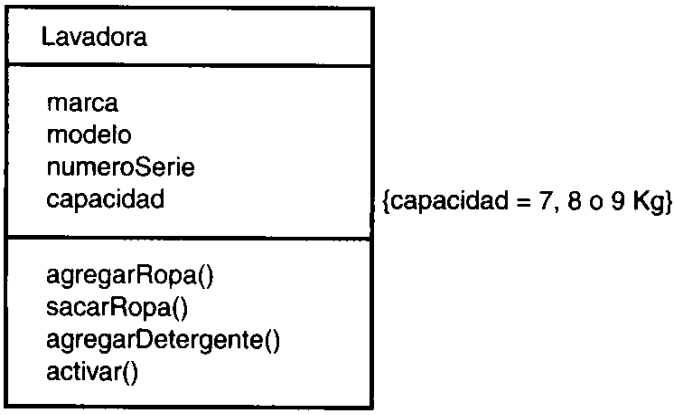

Ilustración 11: Restricción

UML cuenta con un lenguaje formal para especificar restricciones, llamado OCL (Lenguaje de Restricción de Objetos). OCL tiene su propio conjunto de reglas, términos y operadores.

Las **notas adjuntas** se añaden para ofrecer aún más información sobre la clase. Con frecuencia se agregan a un atributo o método. Pueden contener imágenes o texto. En la figura especifica una orden que fija la nomenclatura de números de serie de las lavadoras.

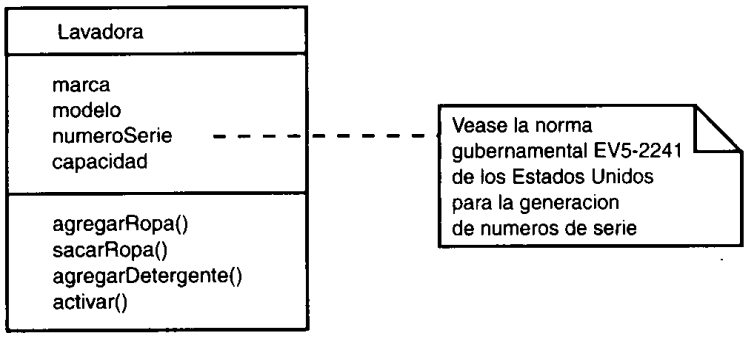

Ilustración 12: Nota adjunta al atributo numeroSerie

Ejercicio 3

Supón que eres un Analista que debe generar un modelo del juego del baloncesto. Para ello entrevistas a un entrenador para comprender el juego. La conversación que tienes es:

***Analista**: "Entrenador, de qué se trata el juego?*

***Entrenador**: Consiste en arrojar el balón a través de un aro, llamado canasta, y conseguir mayor puntuación que el oponente. Cada equipo consta de cinco jugadores: dos defensas, dos delanteros y un central. Cada equipo debe encestar en la canasta contraria.*

***Analista**: ¿Cómo se lleva el balón a la otra canasta?*

***Entrenador**: "botando y pasando la pelota a otros jugadores. Pero el equipo tendrá que encestar antes de que termine el tiempo de posesión para tirar."*

***Analista**: "¿El tiempo de posesión para tirar?"*

***Entrenador**: "Así es, son 24 segundos. Es el tiempo máximo para tirar el balón cuando un equipo toma posesión de él."*

***Analista**: "¿Cómo funcionan las puntuaciones?"*

***Entrenador**: "Cada canasta vale dos puntos, a menos que el tiro haya sido hecho detrás de la línea de los tres puntos. En tal caso, serán tres puntos. Un tiro libre contará como un punto. A propósito, un tiro libre es la penalización que paga un equipo por cometer una falta personal. Si un jugador comete una falta personal sobre un oponente, se detiene el juego y el oponente puede realizar diversos tiros al cesto desde la línea de tiros libres."*

***Analista**: "Hábleme más acerca de lo que hace cada jugador."*

***Entrenador**: "Base y escolta son quienes, en general, quienes realizan la mayor parte de los regates y pases. Por lo general tienen menor estatura que los aleros y ala-pívot, y éstos, a su vez, son menos altos que el pívot. Se supone que todos los jugadores pueden pasar, tirar y rebotear. Los ala-pívot realizan la mayoría de los rebotes y los disparos de mediano alcance, mientras que el pívot se mantiene cerca del cesto y dispara desde un alcance corto."*

***Analista**: "¿Qué hay de las dimensiones de la cancha? Y ya que estamos en eso ¿cuánto dura el juego?"*

***Entrenador**: "La cancha mide 28 metros de longitud y 15 de ancho; la canasta se encuentra a 3.05 m del piso. Las duraciones: en la FIBA, según su reglamento el partido está compuesto por cuatro períodos de 10 minutos cada uno. En la NBA la duración de cada período es de 12 minutos, y en NCAA se juegan dos períodos de 20 minutos cada uno. Un cronómetro del juego lleva un control del tiempo restante para el final."*

Se pide elaborar una lista de clases, propiedades y métodos que modelicen el juego descrito.

4.2 Representando Instancias de objetos

Veremos cómo representar **la instancia del objeto,** también llamada simplemente **objeto** a secas. La parte superior contendrá el nombre de la instancia y de la clase separados por dos puntos y, en el apartado inferior se representan los atributos, pero esta vez con los valores concretos que toman. No es obligado indicar los métodos.

Las instancias se nominan como los atributos. Tras el nombre de instancia, se pone el nombre de la clase, interponiendo dos puntos. El nombre de la instancia y la clase a la que pertenecen, se subrayan.

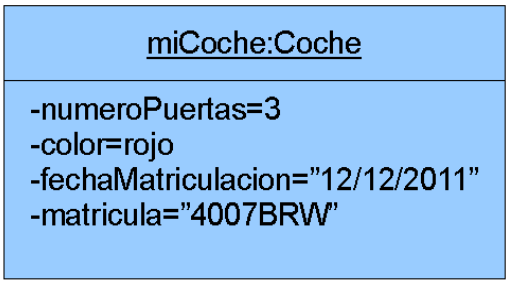

Ilustración 13: Instancia de la clase Coche

Un atributo puede tomar un valor, que se indica tras el símbolo "=".

Para representar **la relación entre clase e instancia** se utiliza la construcción siguiente:

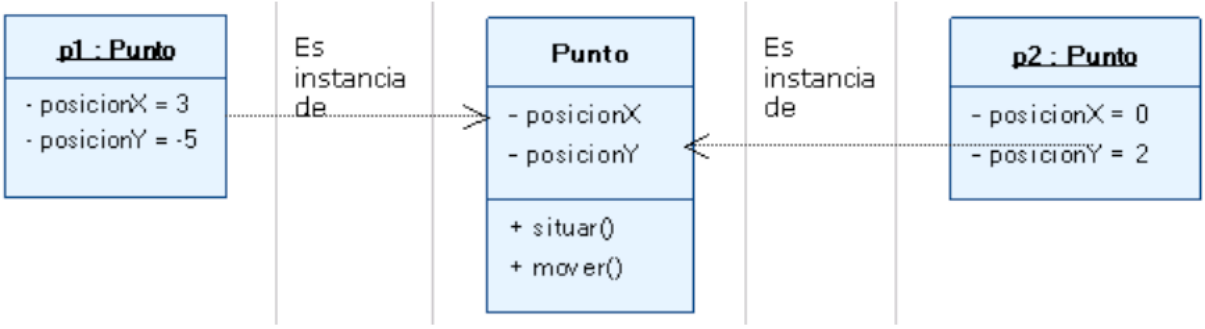

Ilustración 14: instancias de una clase

4.3 Uso de Relaciones

Hasta ahora hemos aprendido a representar las clases y (vagamente) sus relaciones jerárquicas o de herencia. Sin embargo entre distintas clases pueden existir otro tipo de relaciones. En el ejercicio anterior, y dado que conocemos el juego del baloncesto, sabemos que, de algún modo, **el balón tiene algún tipo de relación con los jugadores**, o que **los jugadores se agrupan entre sí para formar un equipo**.

Veremos ahora en detalle las relaciones de herencia y otro tipo de relaciones entre clases y cómo representarlas.

4.3.1 Herencia y generalización

Una clase (secundaria, subclase o clase hija) puede heredar los atributos y métodos de otra, llamada superclase o clase principal. UML también llama a ésto **generalización**. La clase principal es una generalización de la subclase, la clase **`Poligono`** es una generalización de la subclase **`Triangulo`**.

Además, en la generalización, dondequiera que aparece un **`Poligono`** puede ponerse un **`Triangulo`**. Ésto no ocurre a la inversa.

La jerarquía puede tener varios niveles. Una subclase puede ser clase principal de otra. **`Mamifero`** es subclase de **`Animal`** y, a la vez, superclase de **`Elefante`**.

UML representa la herencia con una línea que conecte a la clase principal con la secundaria. Se colocará un triángulo sin rellenar que apunte a la clase principal.

Esta conexión se puede leer como "es un tipo de". **`Mamifero`** es un tipo de **`Animal`**.

Si hay varias conexiones de subclases con una principal, pueden agruparse (no hace falta pintar todos los triángulos).

Para los atributos y métodos de la clase principal, aquellos heredados por las subclases, no es necesario volverlos a incluir en cada clase hija.

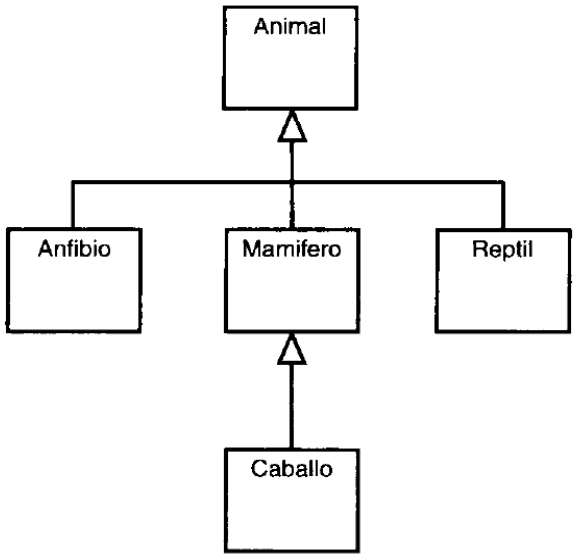

Ilustración 15: Generalización

En las clases secundarias, solo se suelen añadir atributos y métodos propios de la subclase. En **`Mamifero`**, se añaden pelo y leche, atributos que no existen en **`Animal`**.

Si una clase no es subclase de ninguna otra, se denomina **clase base o raíz**. **`Animal`** es la clase raíz en la representación de la Ilustración 15

Ejercicio 4

Para la lista de clases del Ejercicio 3, construye las jerarquías que observes en un diagrama de clases.

Puede darse el caso de una clase de la que nunca se vaya a dar una instancia, pero que contribuyan a diseñar o clasificar el modelo. Esas clases se denominan **clases abstractas**. En la jerarquía de animales, no habrá instancias de ninguna clase primaria, todas ellas serán abstractas. UML sugiere que las clases abstractas se representen en ***`cursiva`***.

4.3.2 Asociaciones

Las clases pueden asociarse de forma conceptual. Esta conexión se denomina **asociación**.

En el ejercicio del baloncesto, los jugadores participan en un equipo. No existe relación de herencia, pero el hecho de participar, implica una asociación de otro tipo.

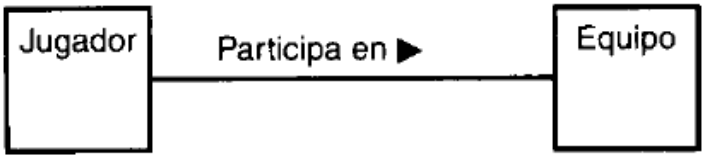

Ilustración 16: Asociación

Se representa como en la Ilustración 16, con una línea que une ambas clases. Sobre la línea se indica el nombre de la asociación. Es útil indicar el sentido de esa asociación mediante un triángulo relleno.

En una asociación, cada clase asociada juega un **papel o rol** en la misma. En la figura anterior, Jugador es el "contratado" mientras que el equipo es el "contratador".

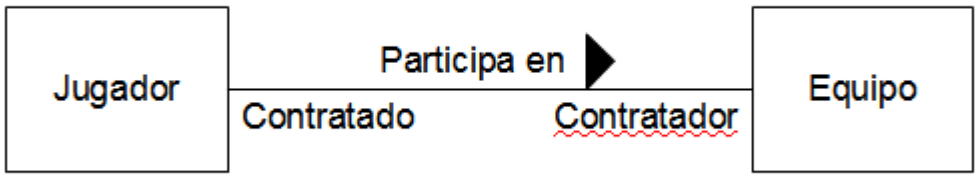

Ilustración 17: Roles

En función del verbo elegido, la relación puede tener dirección inversa. Si en lugar de "participa" decimos "Contrata", la relación tiene dirección opuesta. Podrían representarse ambas simultáneamente si se desea.

Las asociaciones pueden implicar más de dos clases:

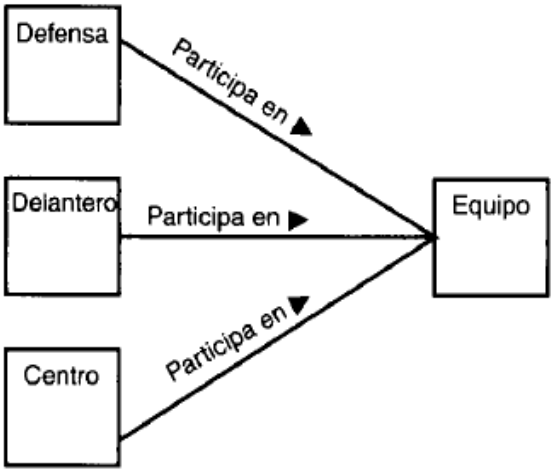

Ilustración 18: Asociación múltiple

→ Restricciones en las asociaciones

En ocasiones, una asociación entre dos clases debe seguir cierta regla. Ésta se indica junto a la línea de la asociación. En la Ilustración 19 un cajero atiende a clientes, pero debe hacerlo de forma ordenada.

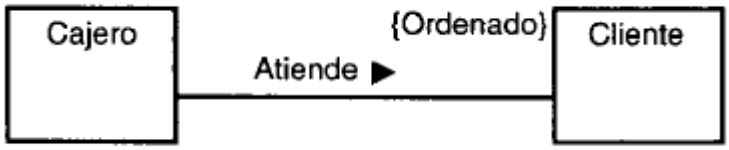

Ilustración 19: Restricción en la asociación

Otro tipo de restricción es la relación O (se especifica {OR}). Se añade en una línea discontinua que une dos líneas de asociación. En la Ilustración 20 se representa que un estudiante de "educación media superior" que elegirá entre un curso académico o un comercial.

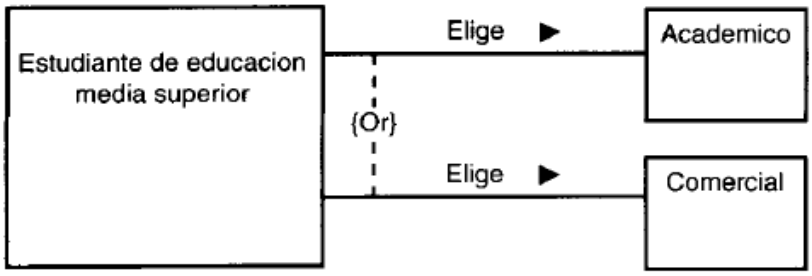

Ilustración 20: Restricción O

→ Clases de asociación

Una asociación, al igual que una clase, puede tener atributos y métodos. Si ésto ocurre, se define lo que se denomina **una clase de asociación.**

La clase se representa como cualquier clase, pero se unirá a la asociación con una línea discontinua.

A su vez, una clase de asociación, puede tener asociaciones con otras clases.

En la Ilustración 21 se muestra una clase de asociación para la asociación "`Participa en`", llamada **`Contrato`**. A su vez, **`Contrato`** tiene una asociación "`Negociado por`" hacia la clase "**`DirectorGeneral`**".

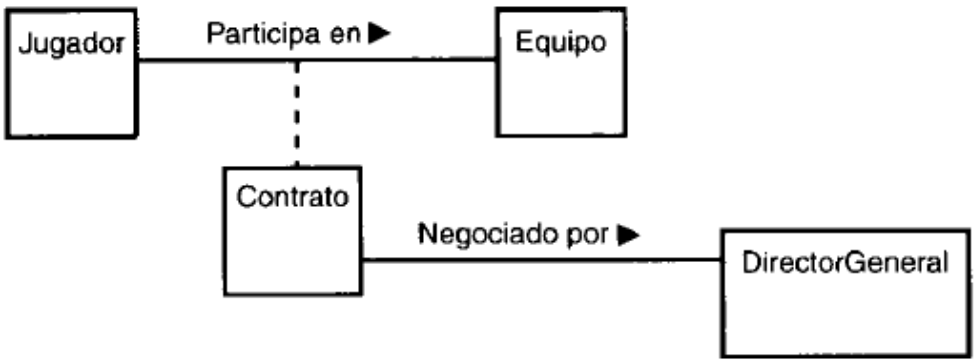

Ilustración 21: Clase de asociación

→ Vínculos

Una asociación también puede tener instancias, al igual que sucedía con los objetos. Imaginemos un jugador concreto, con nombres y apellidos, que juega para un equipo concreto. **En este caso la instacia de la relación se denomina Vínculo**. Se representa como la asociación, pero subrayando el nombre del vínculo.

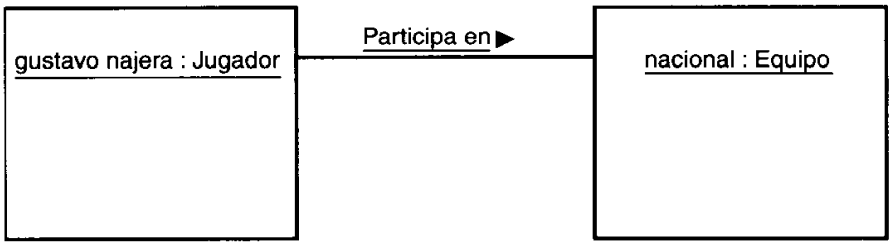

Ilustración 22: Vínculo

→ Multiplicidad

Se especifica la **multiplicidad** en una asociación para indicar la cantidad de objetos de una de las clases que se relacionan con cada objeto de la otra clase asociada.

En el caso del baloncesto, cada jugador puede pertenecer a un sólo equipo y cada equipo posee cinco jugadores. Se indican las cantidades sobre los extremos de la línea de la asociación.

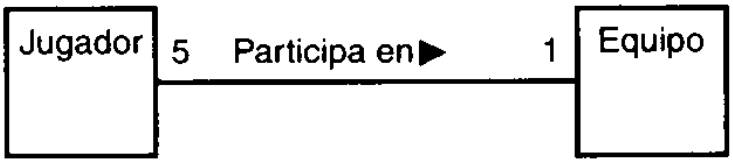

Ilustración 23: Multiplicidad

Sin embargo las cantidades pueden variar, ser un intervalo, ser indefinidas, ser optativas o puede que deba elegirse entre varias.

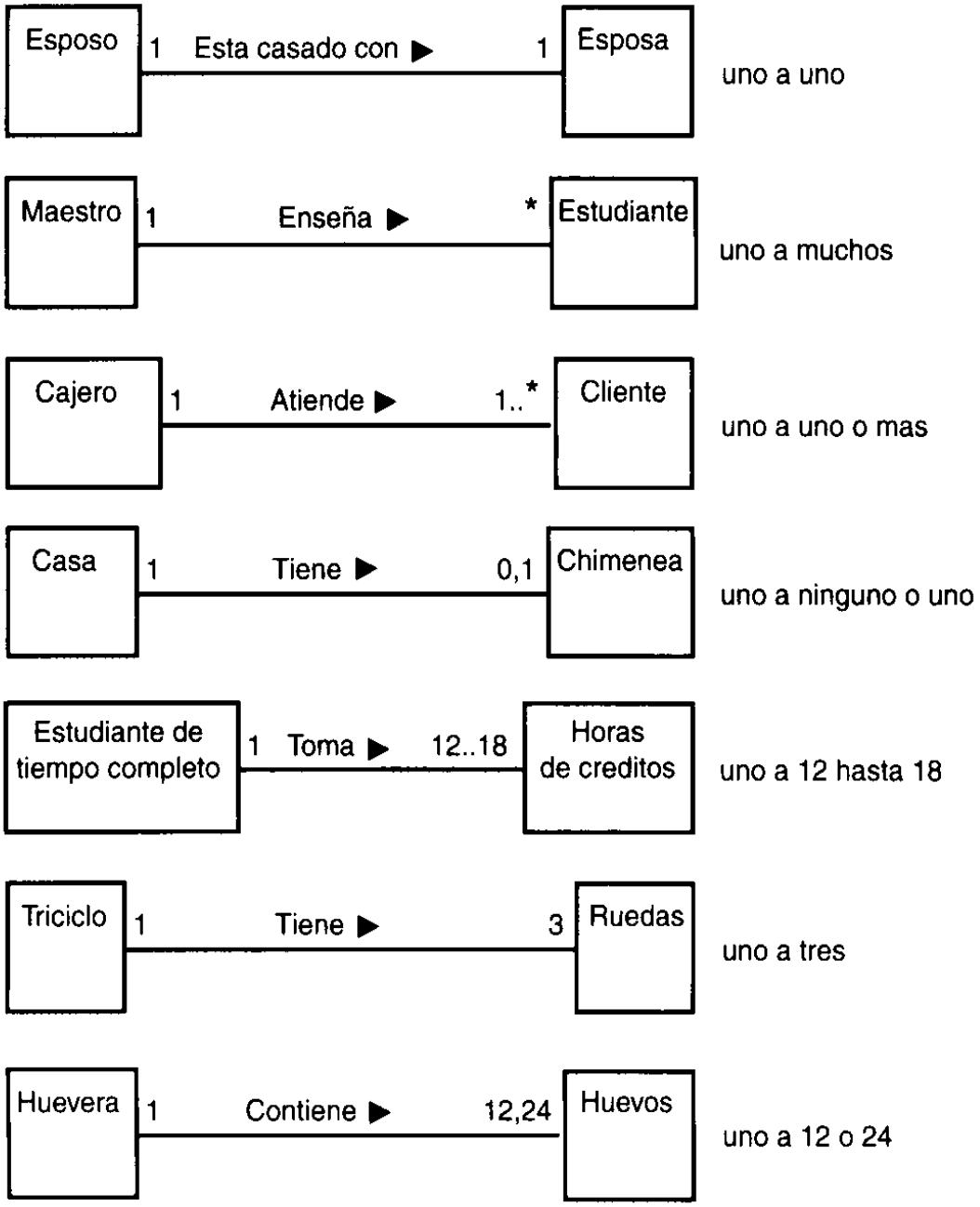

Ilustración 24: Tipos de multiplicidad

En el caso de la Casa y la Chimenea, se dice que la Chimemea es opcional. "*" indica "más" o "muchos". Si sólo se indica "*" el mínimo se asume que es 0. Un intervalo se representa separando con ".." cada límite: 1..18. Una opcionalidad se representa separando por comas: Una huevera contiene 12 ó 24 huevos: 12,24.

→ Asociaciones calificadas

Se da cuando la multiplicidad de la asociación es uno a muchos. En algunos casos, la clase asociada del lado "muchos" debe retornar como resultado de una búsqueda cuál es el elemento asignado. Por ejemplo, al hacer una reserva el recepcionista de un hotel asignará un número a esa reserva. Para referirse de nuevo a esta reserva (para cancelarla por ejemplo) habrá que indicar ese número.

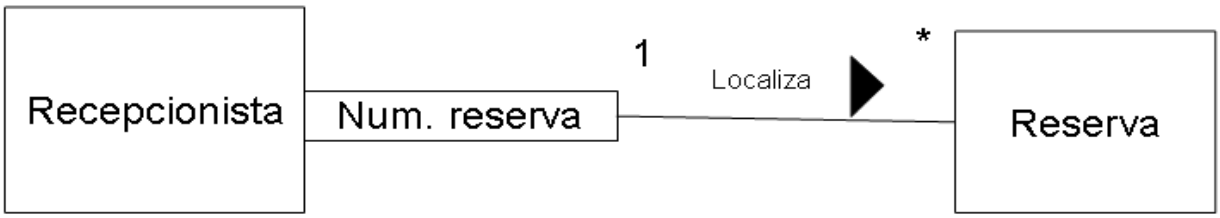

Ilustración 25: Asociación calificada

Ésto se representa cómo se indica en la Ilustración 25, adosando un rectángulo a la clase **`Recepcionista`**, que contendrá el calificador "`Num. reserva`".

→ Asociaciones reflexivas

Es una asociación de una clase consigo misma. Sucede cuando instancias de una misma clase toman papeles o roles diferentes. Ocurre con la asociación ser padre. Dos instacias de **`Persona`** se asocian mediante esta asociación. Cada una toma roles diferentes en la asociación.

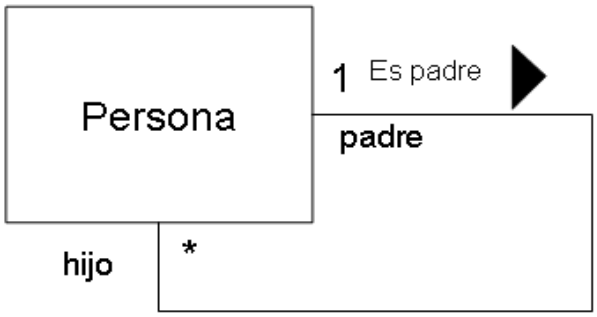

Ilustración 26: Asociación reflexiva

→ Dependencias

Es otro tipo de relación que se da cuando una clase utiliza a otra. Supón que tenemos un sistema donde aparece un menú en el que el usuario selecciona un determinado formulario para rellenar de entre varios. La clase Sistema utiliza a la clase Formulario.

Se representa como la relación de herencia, salvo que la línea es discontinua. La dirección es desde la clase que utiliza hacia la clase utilizada. Un método de la clase que utiliza enviará un mensaje a la clase utilizada.

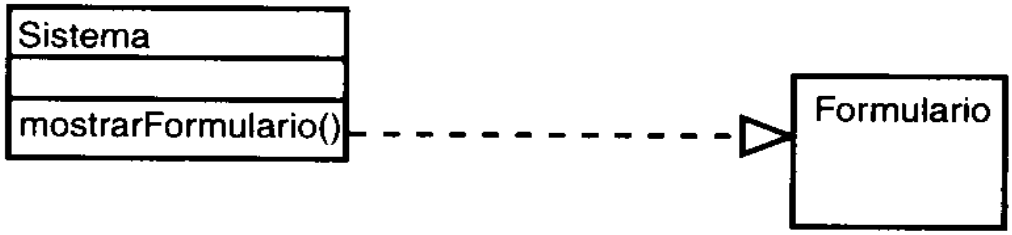

Ilustración 27: Dependencia

Ejercicio 5

Para la lista de clases del Ejercicio 3, construye las nuevas relaciones que observes a partir de la teoría explicada. ¿Hay alguna clase abstracta?¿Y clases de asociación?¿Y relaciones reflexivas?

→ Agregaciones

Una agregación sucede cuando una clase consta de otras clases. Los componentes y las clases que la constituyen, son una asociación que forma un todo. Un ejemplo sería un ordenador de sobremesa, que está formado por múltiples componentes, que, a su vez, se pueden descomponer.

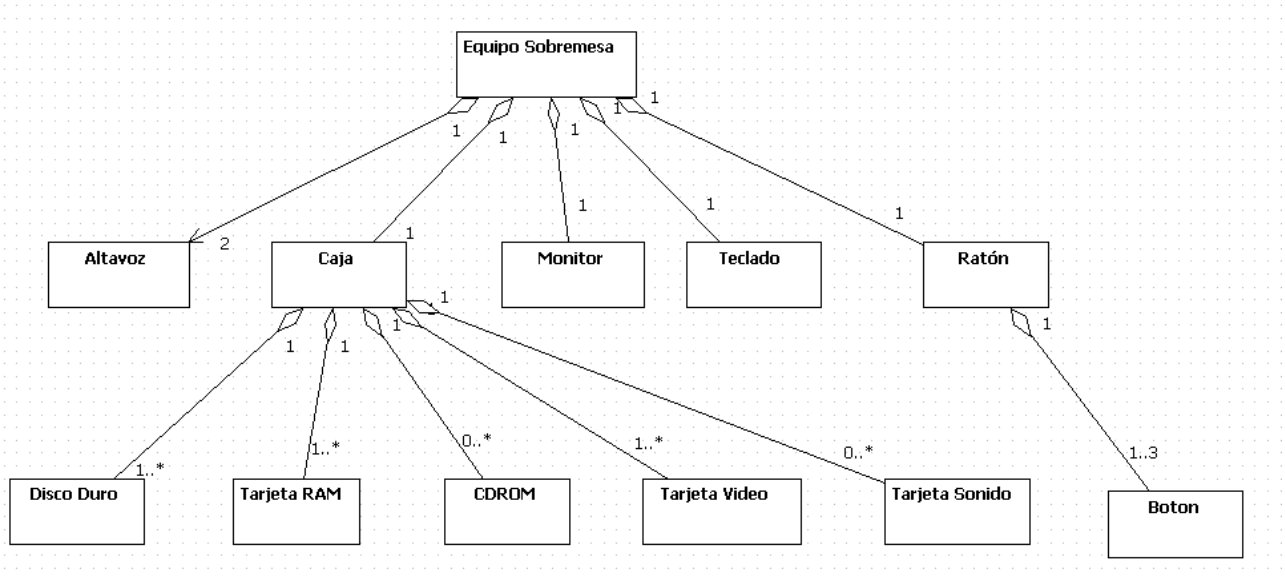

Ilustración 28: Agregación: Equipo de Sobremesa

Está permitido que uno de los componentes de cualquier agregación forme también parte de otra agregación.

Por ejemplo, el mando de una televisión, puede considerarse componente de la misma, pero también puede ser, a su vez, componente del DVD de sobremesa (asumiendo que se trata de un mando multifunción).

Las agregaciones pueden presentar restricciones. Por ejemplo cuando en el caso de ciertos componentes se puede optar por unos u otros. Para modelar ésto se usa la restricción {O} uniendo las relaciones de agregación de esos componentes a elegir.

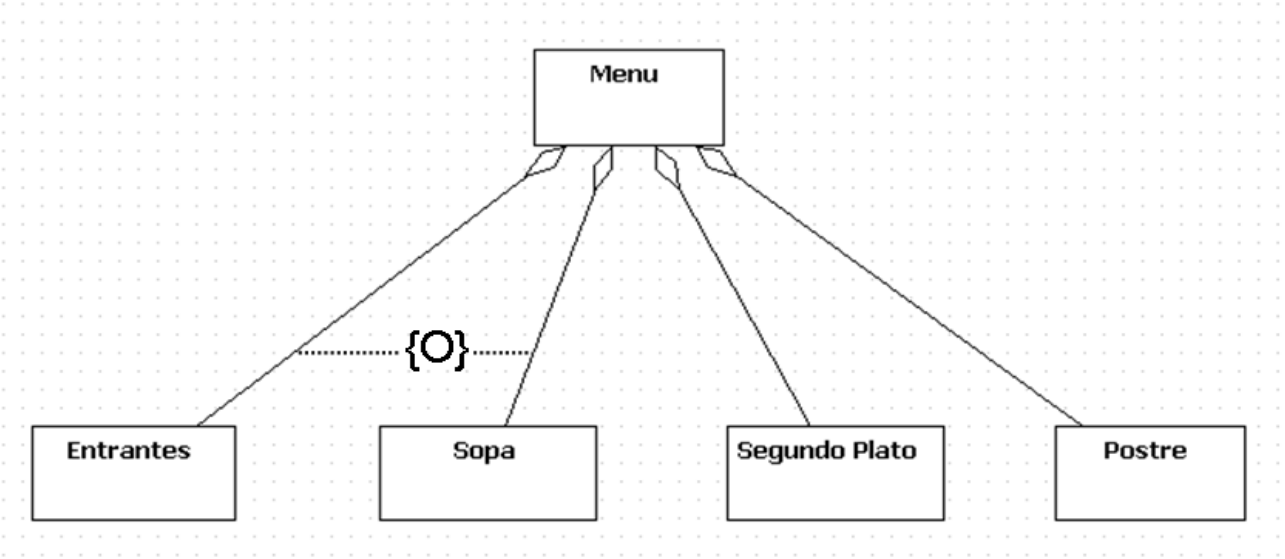

Ilustración 29: Restricción en las agregaciones

→ Composiciones

Es un tipo de agregación. La existencia de los componentes de la composición solo tiene sentido en tanto exista el todo. Además **cada componente sólo puede pertenecer a un todo determinado y no a varios**, como en el caso de mando a distancia. Un ejemplo son las partes de una mesa. La pata de una mesa solo puede ser componente de esa mesa y no de nada más. Si la mesa desaparece, desaparecen sus componentes. Si el ordenador deja de funcionar, los componentes pueden sobrevivir y ser usados en otros ordenadores. Se representa de modo similar, salvo que el rombo es relleno.

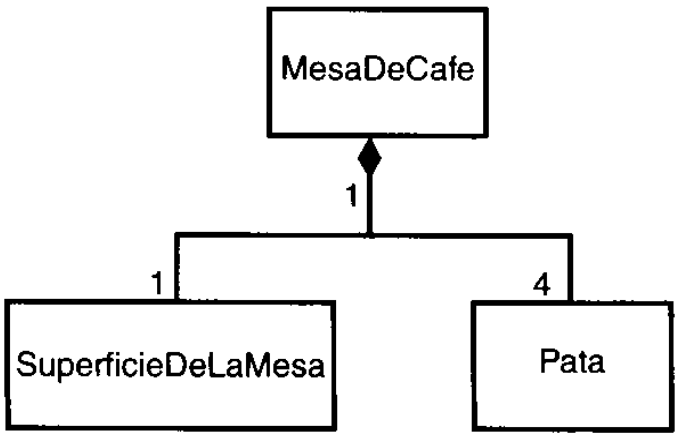

Ilustración 30: Composición: una mesa de café.

→ Contextos

Suponga que se está creando un modelo para representar una camisa. Ésta se compone de partes, pero, a su vez, la camisa puede formar parte de un traje. Para agrupar las partes de la camisa como un todo y usarlo en otro diagrama de un nivel superior en un solo bloque, se usan los contextos.

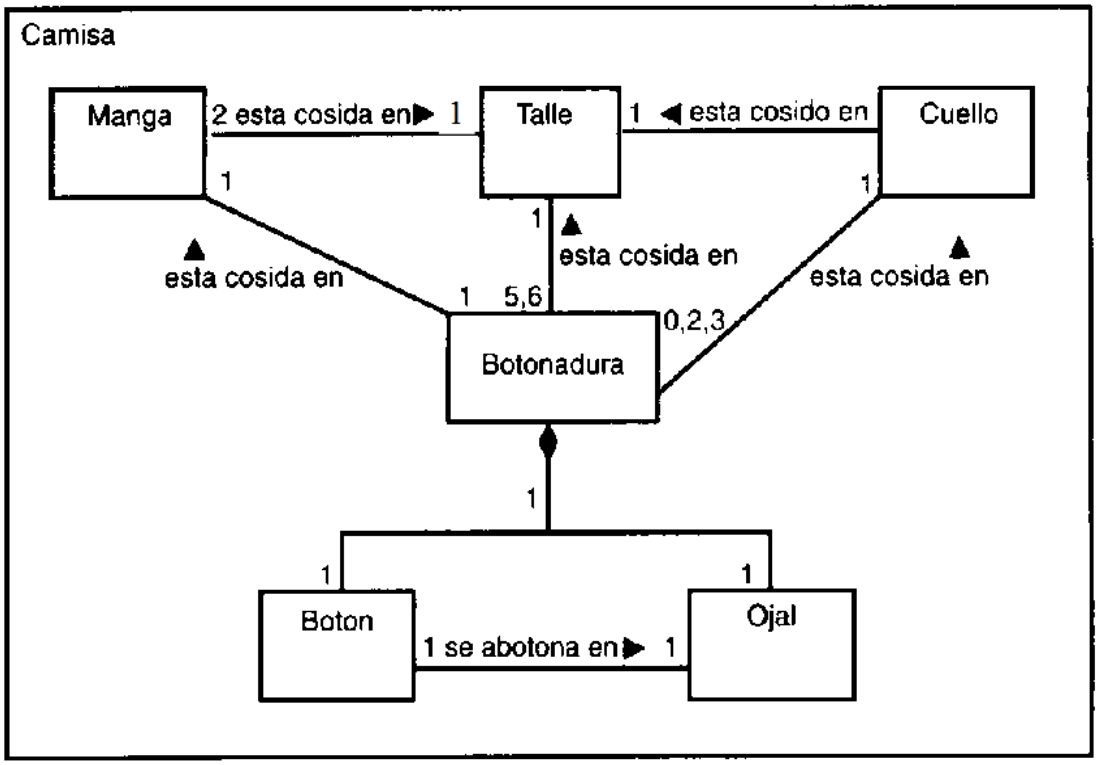

Ilustración 31: Contexto

En el siguiente diagrama se usa el contexto anterior de la camisa en un guardarropa y como parte de un traje.

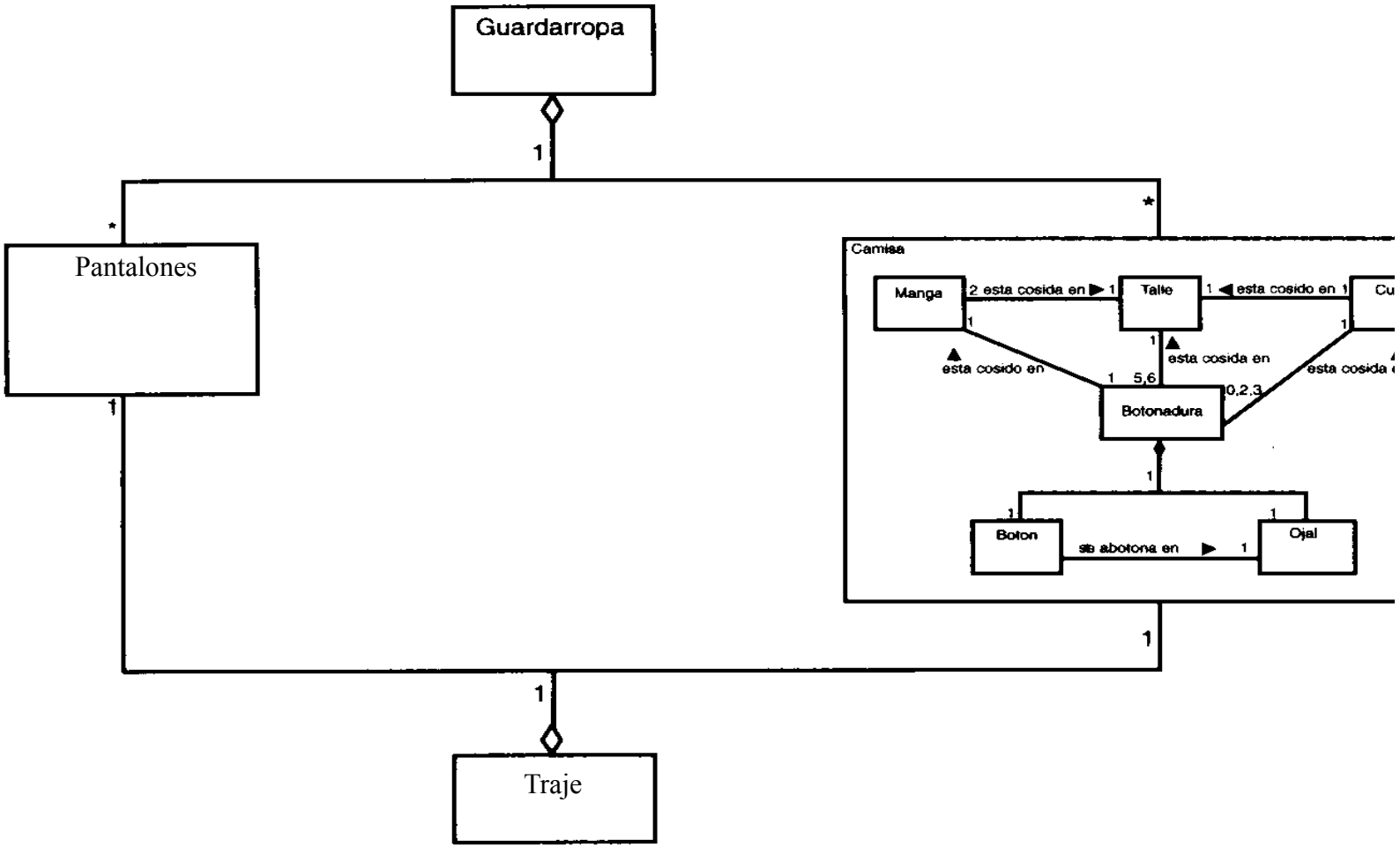

Ilustración 32: Uso del contexto "camisa"

→ Interfaces y realizaciones

En ocasiones, dos clases tienen las mismas operaciones, aunque no tengan relación de herencia (o generalización). Y además esas operaciones tienen el mismo nombre y argumentos[3].

En estos casos se puede desarrollar un esquema de operaciones para ciertas clases, de modo que ese esquema pueda ser reutilizado en otras clases. A ese esquema en UML se le denomina "**interfaz**".

Una **interfaz** es un conjunto de operaciones que especifica cierto aspecto de la funcionalidad de una clase y que permite que sea usado por otras clases.

Por ejemplo, un teclado de ordenador tiene funciones comunes con una máquina de escribir, aunque ambos objetos no puedan agruparse en una generalización. En ambos casos, las teclas de los teclados se pulsan y tienen similar disposición. Además ambas tienen una tecla de mayúsculas (sin bloqueo o "*shift*"), otra para bloquear las mayúsculas y un tabulador. Pero el teclado de ordenador tiene más funciones (teclas de función, AvPag, RePag, cursores, …). Como un interfaz puede constituir un subconjunto de las operaciones necesarias, no hay problema. Estas últimas operaciones se harían directamente en la clase **`Teclado`** y las primeras usando el interfaz definido para la máquina de escribir.

Una interfaz se modela con un rectángulo igual que una clase. La diferencia estriba en que un interfaz no tiene atributos. Esto hace difícil diferenciar una clase con los atributos ocultos de una interfaz. Una solución es indicar que se trata de una interfaz usando un estereotipo. Otra opción es nombrar todos los interfaces con el prefijo "I".

La relación entre una clase y un interfaz se denomina "**realización**". Se modela como una línea discontinua terminada con un triángulo en blanco hacia el lado del interfaz.

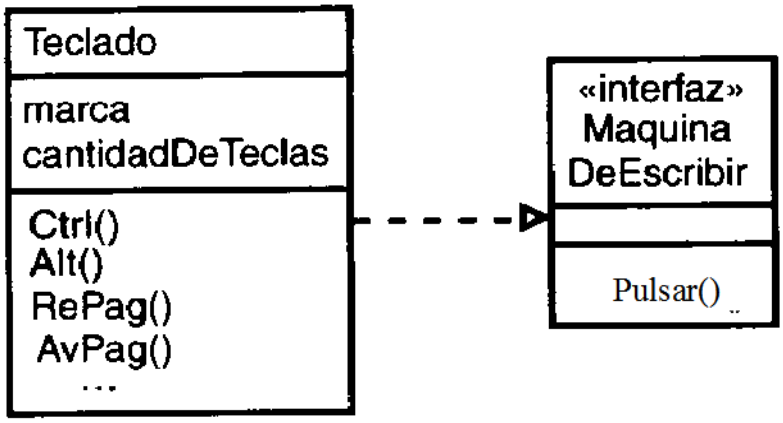

Ilustración 33: Interfaz "MaquinaDeEscribir"

En UML 2.0 se introduce una nueva nomenclatura llamada "ball and socket". El interfaz se representa aquí como un círculo. En caso de que alguna clase use la clase que implementa un interfaz, la relación de dependencia se representa como un enchufe ("socket") sobre la bola del interfaz usado.

Supongamos que un **`Ordenador`** usa el teclado a través del interfaz **`MaquinaDeEscribir`**. La representación sería la siguiente.

[3] Se denomina "firmna" al esquema de una operación o método

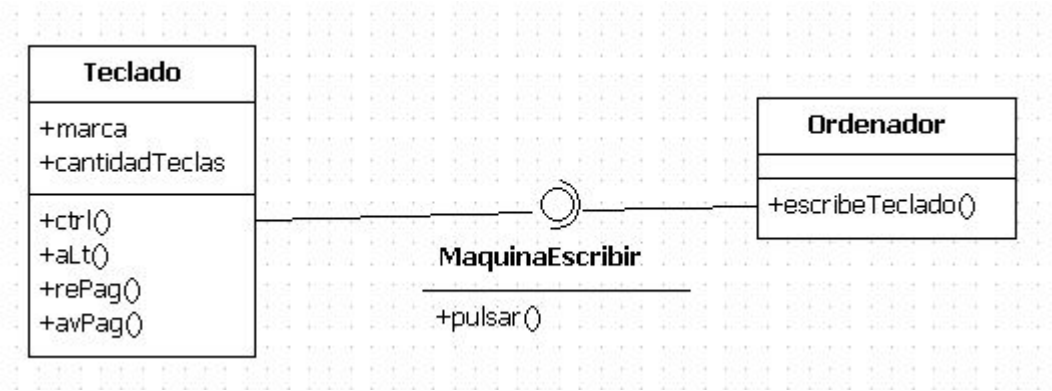

Ilustración 34: Notación bola y enchufe ("ball and socket")

Una clase puede realizar más de una interfaz y una interfaz puede ser realizada por más de una clase.

→ Visibilidad y ámbito

La **visibilidad** se aplica a atributos y operaciones. Establece en qué medida otras clases pueden hacer uso de atributos y operaciones de una clase dada (u operaciones de una interfaz).

Hay tres niveles:

- Nivel público (+). Otras clases tienen acceso a este atributo o método.
- Nivel privado (-). Ninguna otra clase tiene acceso a este atributo o método.
- Nivel protegido (#). Sólo las clases derivadas o que heredan de ésta clase pueden acceder a este atributo o método.

Para la realización de un interfaz, es necesario que las operaciones del interfaz sean públicas.

La visibilidad se indica anteponiendo los símbolos (+, - o #) al atributo o método.

El **ámbito** se aplica a atributos o métodos. Puede ser de instancia ("*instance"*) o de clasificador o archivador ("*classifier*"). En el primer caso, cada instancia del objeto tendrá un valor distinto y "personalizado" para ese atributo u operación. Es el más común. En el segundo (*classifier*), para todas las instancias de la clase, el atributo es el mismo y tiene un valor único. Todas las instancias comparten el atributo.

Ejercicio 6

Crea un diagrama de contexto de la composición de una revista. Ten en cuenta las partes:

- *Portada*
- *Editorial*
- *Artículos de Noticias*
- *Columnas de Opinión*
- *Anuncios*
- *Fotos*
- *Titulares*
- *Entradilla*
- *Texto*
- *Crea a continuación un diagrama de contexto con el suscriptor de la revista y el comprador directo (el que compra en el kiosko)*

Ejercicio 7

Obtener el modelo conceptual de la gestión de los estudiantes en un IES de FP.

En el Ies se imparten varios ciclos, cada uno tiene un nombre un currículo y una titulación al finalizarse y constan de dos cursos y varias asignaturas (o módulos) por curso.

Una persona viene caracterizada por su dni, nombre, dirección y estado civil, y ésta puede convertirse en estudiante al darse de alta como tal en el IES. Las personas también pueden ser profesores del mismo IES.

Como estudiante podrá matricularse de las asignaturas que se imparten, asignándosele un numero de estudiante. Las asignaturas tendrán un código de asignatura, un nombre, un profesor responsable y el curso en que se imparten (primero o segundo).

Una vez el estudiante se matricula, deberá hacer exámenes en Junio y Septiembre de las asignaturas en las que se encuentra matriculado hasta que finalice el curso y vuelva a matricularse de nuevo, o bien deje el IES por abandono o finalización del ciclo y con ello deje de ser estudiante.

Cada examen realizado en una asignatura contendrá: fecha, nota y enunciado del examen.

Durante el curso se realizarán prácticas de fin de curso en cada asignatura, de las que se entregará una memoria que deberá entregarse en soporte papel y digital, pudiendo en este segundo caso ser CD o Pen-drive. Las prácticas constan además, de enunciado, fecha de entrega y nota.

*Al final de cada curso se entregará un boletín con las notas de todas las asignaturas calculadas como nota examen*0,6+nota práctica*0,4 y un apartado donde figure si aprueba o no.*

El IES dispone de un a biblioteca donde el alumno puede "matricularse". El carnet contiene el Nombre, DNI, número de estudiante y número de carnet.

El IES está compuesto por alumnos, profesores, personal directivo (nombre, cargo), aulas (número de aula), laboratorios y biblioteca. Cada curso de un ciclo se imparte en un aula y puede tener asignado un laboratorio.

Un profesor se encargará de la biblioteca.

Ejercicio 8

Haz un diagrama de clases para cada caso, indicando correctamente las multiplicidades:

1. *En una empresa dada, una factura se envía a un cliente y puede haber muchas facturas enviadas a un mismo cliente.*
2. *Un elemento se usa en muchos proyectos y muchos proyectos usan el elemento. Los proyectos tienen al menos un elemento, pero un elemento concreto puede que no se use en ningún proyecto de la empresa.*
3. *Los estudiantes tienen asignaturas. Cada asignatura puede ser elegida por muchos estudiantes y cada estudiante puede tener muchas asignaturas.*
4. *Las personas solicitan préstamos. Cada préstamo debe concederse a una sola persona, pero cada persona puede tener muchas solicitudes.*
5. *Un operador puede trabajar en muchas máquinas y cada máquina tiene muchos operadores. Cada máquina pertenece a un departamento, pero un departamento puede tener muchas máquinas.*
6. *Los empleados usan herramientas. Un destornillador es una herramienta. Los destornilladores pueden ser manuales o eléctricos. Cada herramienta puede ser usada por el personal que esté capacitado para ello. Una herramienta nueva puede no haber sido usada aún.*
7. *En un edificio hay ascensores y montacargas. Ambos pueden subir o bajar a una planta. Ambos pueden abrir y cerran sus puertas. Una persona usa el ascensor. Un paquete usa el montacargas.*
8. *Representa para la clase Persona, las relaciones de parentesco: estar casado, ser padre y ser hermano y sus multiplicidades.*
9. *Para el caso anterior, considera tanto padres naturales como padrastros y madrastras y casos de matrimonios de más de dos personas (mormones, musulmanes, ...).*
10. *Para el caso anterior, considera las diferencias entre prohibir o permitir el matrimonio entre personas del mismo sexo.*

5 DIAGRAMAS DE CASOS DE USO

Los diagramas de clases, los vistos hasta ahora, sólo ofrecen una idea estática de la realidad, mediante la representación de ésta, pero no se incluye nada acerca de la acción, de cómo el sistema interactúa y evoluciona en el tiempo.

La **representación estática** sirve durante la fase de adquisición de requisitos, para recoger las necesidades del **cliente**. La **representación dinámica** es de mayor ayuda para la comunicación del analista con los **desarrolladores**, para ayudarles a crear los programas.

Sin embargo hay alguien que necesita ver reflejado su propio punto de vista. Éste es el **usuario** y es a quién van dirigidos los diagramas que veremos.

Aprenderemos a usar UML para representar casos de uso.

5.1 Definición de Caso de Uso

Siempre que compramos un electrodoméstico o similar nos hacemos una serie de preguntas sobre su funcionalidad: ¿Qué características necesito?¿Qué puede hacer?¿Cómo es el interfaz de uso?¿Qué accesorios tiene?...

Estas preguntas y otras constituyen el "análisis de uso" del aparato. Este tipo de análisis es muy importante para el desarrollo de un producto software. Es fundamental diseñar cómo el usuario va a utilizarlo para saber qué hacer y qué no. Además la aceptación o no del producto muchas veces dependerá de su usabilidad.

El caso de uso es una estructura que ayuda a los analistas a trabajar con los usuarios para determinar la forma en que se usará el sistema.

Se trata de hacer un esquema de lo que los usuarios pueden hacer con el producto.

El caso de uso está compuesto por una serie de situaciones relativas al uso del producto. Cada situación recibe el nombre de **escenario.** En cada escenario se describe una **secuencia** de eventos. Cada secuencia se inicia por los que se denomina **el actor**[4] o los actores y que pueden ser:

1. Una persona.
2. Otro sistema.
3. El hardware.
4. El paso del tiempo.

El resultado de la secuencia debe ser utilizable por el mismo actor que la inició o por otro actor.

El caso de uso estimula a los usuarios a hablar sobre el sistema que necesitan, es importante contar con ellos en las etapas iniciales de análisis y diseño. Esto aumenta las posibilidades de que el sistema realizado sea el correcto a ojos del receptor del mismo.

5.2 Un ejemplo de Caso de Uso.

El ejemplo que se va a plantear es el de una máquina de bebidas. La función principal es adquirir un refresco, por lo que éste será el **caso de uso** a estudiar.

5.2.1 Primer caso de uso

Nos podremos enfrentar a diferentes **escenarios**[5] dentro de este caso de uso.

[4] Cuando se trata con actores, conviene pensar en los papeles o roles que desempeñan, no en las personas físicas ni en los puestos que ocupan. Un comercial puede actuar como comercial en un momento dado y como agente de ventas en otro momento, por ejemplo. De ese modo se trataría de dos actores distintos aunque sea la misma persona.

[5] La palabra escenario se refiere a una sola ruta a través de un caso de uso. A veces puede ser sinónimo de caso de uso.

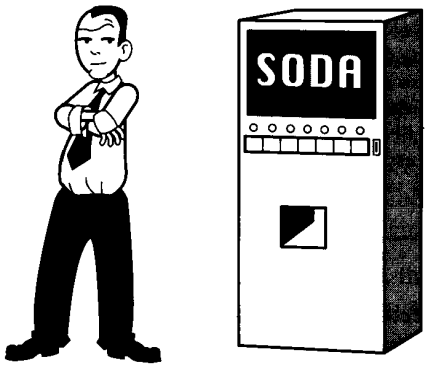

Ilustración 35: Caso de uso: adquirir refresco

Es escenario típico se inicia cuando el actor inserta dinero. Posteriormente escoge el refresco que desea. Si todo va bien, el refresco se entrega al cliente.

Aparte de la secuencia, se deben considerar los **condicionantes previos** y posteriores (**resultados**) del actor.

- ¿Qué le lleva a comprar?: probablemente la sed
- ¿Qué obtiene?: El refresco.

Por otro lado, el escenario planteado es el de "todo va bien". Pero hay que considerar todos las posibles situaciones que excepcionalmente puedan ocurrir:

- ¿Qué acciones se suceden si el cliente no introduce el importe exacto?
- ¿Qué sucede si pone de más?
- ¿Qué sucede si la máquina ha agotado el cambio?
- ¿Qué sucede si la máquina agota el producto?
- ¿Qué sucede si el producto se atasca al salir?

Ejercicio 9
Para el caso de uso anterior, plantea todos los posibles escenarios y sus soluciones.

5.2.2 Otros casos de uso

En la máquina hay otros usuarios (actores) aparte del cliente[6]:

- El proveedor
- El encargado de recoger la recaudación

Ambos se enfrentan a sus particulares casos de uso con sus escenarios:

[6] Puede darse el caso de que un mismo actor pueda realizar muchos casos de uso; A la inversa también, un caso de uso puede ser realizado Por varios actores diferentes.

→ Caso de uso: *"reabastecer" el género*

Escenarios:

Se inicia, posiblemente de modo periódico, transcurrido un tiempo fijo (una semana, por ejemplo). El proveedor abre la puerta de abastecimiento y procede a rellenar todos los compartimentos hasta el tope de capacidad. Luego cierra la puerta y asegura el cerrojo.

La **condición previa** que inicia el caso es que haya transcurrido una semana.

El **resultado** es que se incrementan las ventas potenciales.

→ Caso de uso: *"recaudar dinero"*

Este paso es similar al anterior, por lo que se deja como ejercicio al alumno.

Como los casos de uso tienen pasos comunes, **es posible agrupar esos pasos** en un caso de uso adicional al que se da un nombre para ser reutilizado. Por ejemplo *"abrir puerta"*. Como resultado aparecen casos de uso anidados, es decir, casos de uso que contienen casos de uso. A esta técnica se la denomina **inclusión[7] de un caso de uso**.

> ***Ejercicio 10***
> *Plantea los casos de uso y posibles escenarios para los dos actores anteriores en la máquina de refrescos, usando "casos de uso incluidos" o comunes.*

En los casos de uso, **no interesa la implementación**. No queremos saber cómo es por dentro la máquina, o si se cierra y abre con llave o con combinación. No interesa el mecanismo de refrigeración ni el método mecánico o electrónico de contar las monedas introducidas. Los casos de uso se preocupan del interfaz con el exterior, de la apariencia y mecanismos de uso para quien la utilice.

El resultado es una máquina diseñada por (al menos en parte) y para quienes la usan.

5.3 Extensión de casos de uso

En ocasiones sucede que se puede reutilizar un caso de uso, simplemente añadiendo nuevos pasos. Por ejemplo en la máquina anterior, en el caso de uso *"reabastecer"* puede suceder que el actor decida reorganizar las bandejas reduciendo la cantidad de productos que se venden peor y añadiendo bandejas para los que se venden mejor. Habría que retocar los rótulos del panel frontal de botones que usa el cliente. Este nuevo caso de uso es una extensión de *"reabastecer"* al que se puede denominar *"reabastecer de acuerdo a ventas"*. Simplemente se añaden pasos.[8]

5.4 Análisis de casos de uso

En el mundo real, analizar casos de uso tiene mucha más ciencia que el ejemplo visto.

Suele comenzar con entrevistas y reuniones con los clientes y con expertos en el tema. Se diseñan los diagramas de clases. Posteriormente se habla con los usuarios para diseñar los casos de uso. Para

[7] En inglés se usa "use" y no "include". Usamos la traducción "incluye" en lugar de "usa" por evitar la redundancia "uso de un caso de uso". No obstante, algunos autores traducen "usa" directamente.

[8] En el fondo Extensión e Inclusión cumplen una misma función al factorizar o reutilizar clases ya creadas. Utiliza extensiones cuando se describa una variación de conducta normal. Emplea inclusiones cuando nos encontramos con una parte de comportamiento similar en dos casos de uso y no queremos repetir la descripción de dicho comportamiento común.

ambas tareas pueden emplearse multitud de técnicas que dan para rellenar algún que otro libro más:

- Empleo de cuestionarios donde hay diversos tipos y modelos de cuestionario.
- Técnicas de entrevista.
- Técnicas de reunión.

5.5 Representación de los casos de uso en UML

El resultado del análisis de casos de uso son las representaciones de los mismos (mediante UML en nuestro caso) con objeto de sistematizar la perspectiva de los usuarios del producto software. Las representaciones obtenidas serán el punto de partida en la fase de mantenimiento, cuando se soliciten mejoras o actualizaciones del software finalizado tiempo atrás.

En UML, una elipse representará cada caso de uso. Los actores vienen representados por figuras esquemáticas humanas.

El actor (o actores) que inicia(n) el caso se sitúa(n) a la izquierda y el (los) que recibe(n) algo de valor como resultado, a la derecha.

Una línea conecta al actor con el caso de uso, representando la comunicación con el caso de uso. Se usa un rectángulo para indicar el límite del sistema. Lo que esté fuera no pertenece al sistema.

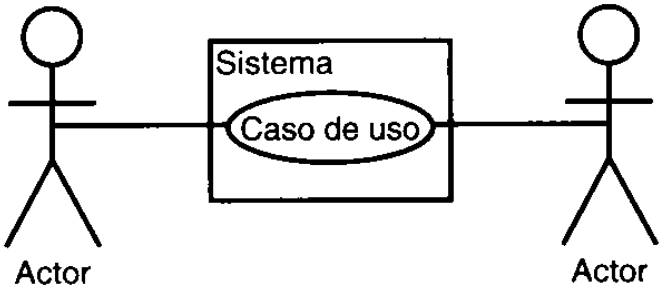

Ilustración 36: Caso de uso en UML

En el caso de la máquina de refrescos, el modelo queda así:

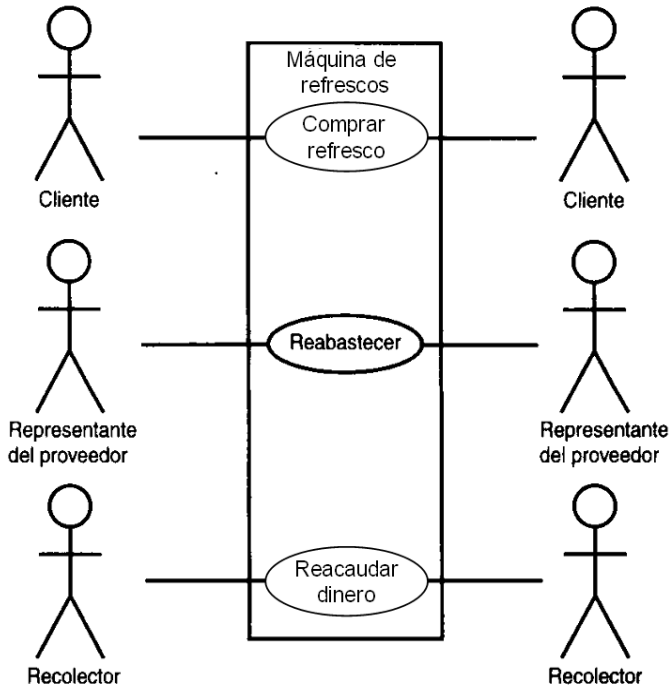

Ilustración 37: Máquina de refrescos

5.6 Secuencias de pasos en los escenarios

Hasta ahora los pasos de los escenarios de cada caso de uso no aparecen en el diagrama. En los diagramas cada diagrama se realizará en una página separada y cada escenario de caso en otra página más. En este segundo caso, se indicará:

1. El actor que inicia cada caso
2. Condiciones previas al caso
3. Pasos en el escenario
4. Condiciones posteriores (resultados) cuando se finaliza el escenario
5. El actor receptor de los resultados

Se pueden indicar otras condiciones necesarias (conjeturas) del escenario (por ejemplo, que sólo un cliente puede usar la máquina cada vez) y una breve descripción del escenario en una frase.

El caso de uso *"Comprar refresco"* presentaba escenarios alternativos: *"Producto agotado"*, *"Cambio agotado"*, etc, … Se pueden considerar escenarios aparte o bien excepciones al primer caso, según consideren analista, cliente y usuarios.

Los pasos de un escenario suelen indicarse con otro diagrama llamado "**diagrama de actividades**" que se verá más tarde.

5.6.1 Representar la Inclusión

Los casos *"Reabastecer"* y *"Recaudar dinero"* se inician mediante el caso incluido *"exponer interior"* y finalizan los dos con el *"cierre de la máquina"*. *"Exponer interior"* contempla la

apertura de cerradura y la de la puerta. *"Cierre de la máquina"* contiene los pasos inversos.

La inclusión se representa como la dependencia entre clases, apuntando la flecha hacia la clase dependiente. Sobre la línea se indica un estereotipo: la palabra **<<incluir>>**.

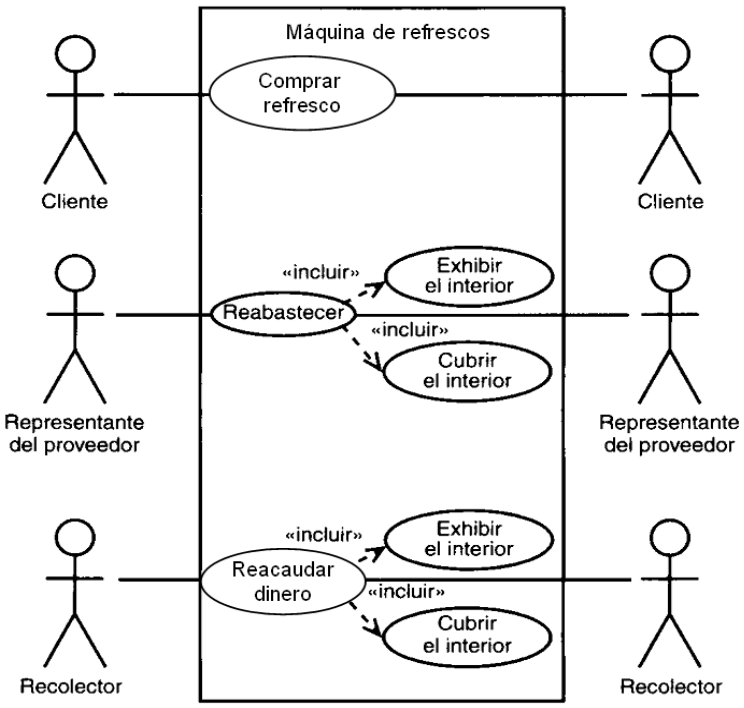

Ilustración 38: Inclusión de casos de uso

5.6.2 Representar la Extensión

Vimos anteriormente la extensión *"reabastecer de acuerdo a ventas"* como ampliación del original *"Reabastecer"*, que se conoce como **caso base**.

Para incluir una extensión en un caso base, debemos señalar el lugar donde es permitido hacerlo, mediante la especificación de lo que se conoce como **puntos de extensión**.

Para nuestro ejemplo, los casos de extensión podemos llamarlos *"anotar las ventas"* y *"abastecer de acuerdo a ventas"*. Los puntos de extensión hay que colocarlos tras abrir la máquina y antes de reponer el género. El punto de extensión en el ejemplo es *"llenar los compartimentos"*.

Como la inclusión, se representa la extensión mediante una línea discontinua con flecha dirigida desde el caso extendido hacia el caso base. Se muestra <<extender>> como estereotipo.

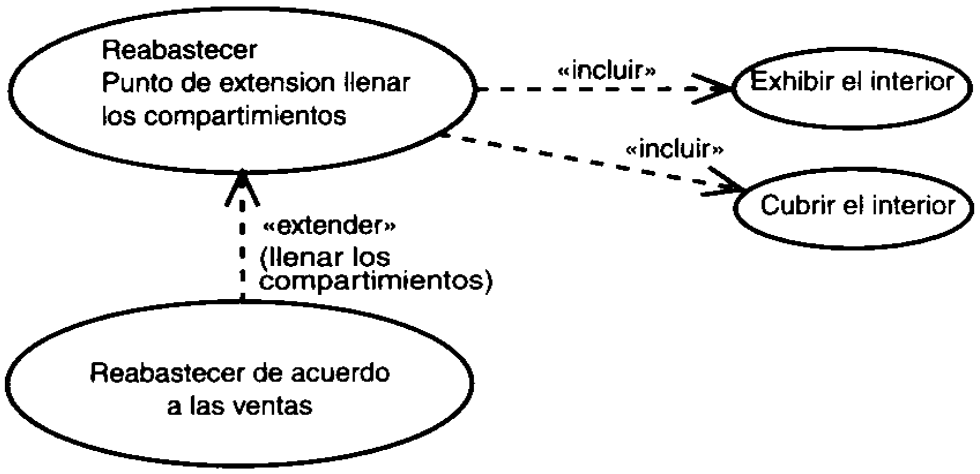

Ilustración 39: Extensión "Reabastecer de acuerdo a las ventas"

El nombre del punto de extensión se pone dentro de la elipse del caso base, debajo del nombre del caso base. En la figura se muestran las inclusiones anteriores también.

5.6.3 Generalización

Además de la inclusión y la extensión, los casos de uso ofrecen dos relaciones más: la **generalización** (cuando un caso de uso hereda de otro) y el **agrupamiento**, que es una manera de organizar los casos de uso. Veamos el primero.

En la generalización de los casos de uso, el caso de uso secundario hereda las acciones y significado del primario y además agrega sus propias acciones. Más tarde, puede aplicar el caso de uso secundario en cualquier lugar donde esté permitido aplicar el caso primario del que hereda.

Con un ejemplo puede verse que el caso de uso “comprar vaso de refresco” hereda acciones del caso “comprar refresco” visto en la máquina anterior. El caso secundario tendrá acciones nuevas como “colocar vaso” o “añadir hielo”. Se representa con el mismo símbolo que en el diagrama de clases.

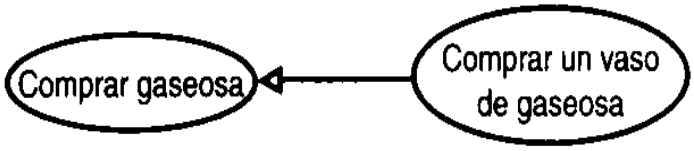

Ilustración 40: Generalización de casos de uso

La generalización también puede aplicarse a actores. Si introducimos un actor “Agente proveedor”, tanto el “Reabastecedor” de género como el “Recolector” de dinero serían derivados de ese primero.

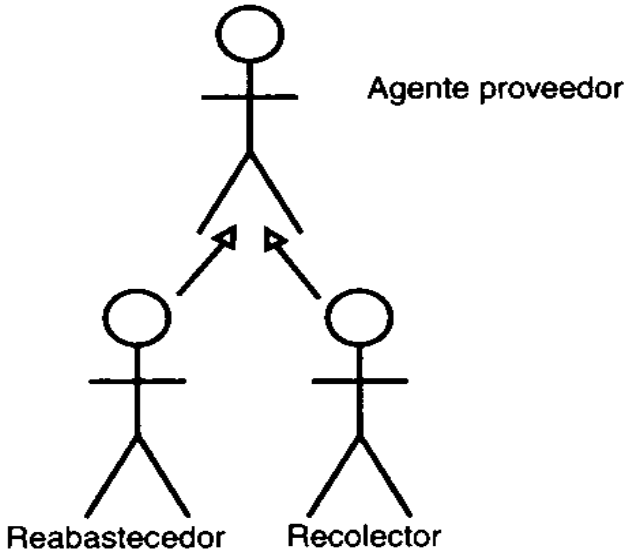

Ilustración 41: Herencia en actores

5.6.4 Agrupamiento

Cuando un sistema consta de subsistemas pueden aparecer muchos casos de uso que es preciso organizar. Habrá que organizar los casos de uso por categorías.

Los casos de uso se agrupan en paquetes de una misma categoría, representando cada paquete por una carpeta con etiqueta.

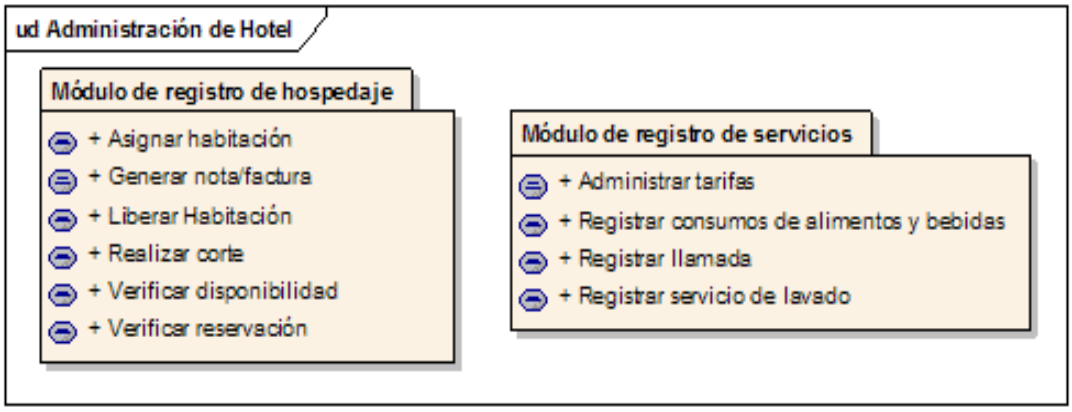

Ilustración 42: Casos de uso para administrar un hotel, agrupados en paquetes

Ejercicio 11
Elabora el diagrama de casos de uso de una aplicación informática para una biblioteca, considerando los posibles actores y situaciones: solicitar carnet, sacar libro, devolver libro, ...

6 DIAGRAMAS DE ESTADOS

Conforme pasa el tiempo suceden cambios en el estado de las cosas. El campo cambia con las estaciones, en una estación de tren entran y salen distintos tipos de trenes, los objetos y su estado cambian con el paso del tiempo.

Un sistema también cambia por interacción con otros sistemas o con lo usuarios con los que interactúa.

Se necesita un mecanismo para modelar ésto.

Los sistemas cambian como respuesta a los sucesos y al tiempo:

- Cuando se acciona un interruptor, la luz pasa de apagada a encendida y viceversa.
- Cuando se presiona el control remoto, la televisión cambia de canal.
- Pasado un tiempo, una lavadora pasa de aclarado a centrifugado.

El diagrama de estados refleja los cambios **de un solo objeto**, presentando los estados junto con las transiciones a otros estados y los sucesos que los originan. También muestra el estado inicial y el estado final de una secuencia completa de cambios de estado

Un sinónimo para diagrama de estados es “motor de estado”.

6.1 Estados

El estado se representa por un rectángulo de esquinas redondeadas.

La transición es una flecha, que une los estados.

El punto de inicio es un círculo relleno y el final uno similar a una diana.

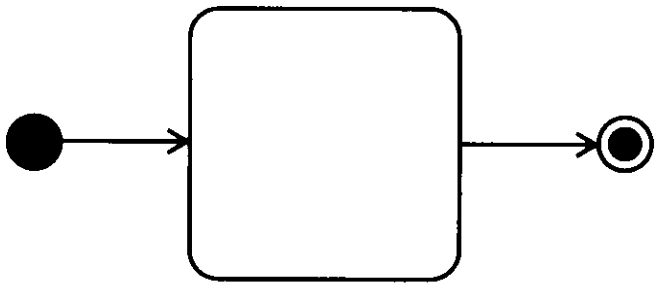

Ilustración 43: punto inicial, estado, punto final y transiciones

6.1.1 Partes de un estado

El símbolo del estado se puede dividir en tres apartados:

1. Nombre del estado
2. Variables cuyo valor condicionan el estado.
3. Actividades. Constan de sucesos y acciones: tres de las más utilizadas son:
   a) entrada: qué sucede cuando el sistema entra al estado
   b) salida: qué sucede cuando el sistema sale del estado.
   c) Hacer: qué sucede cuando el sistema está en el estado

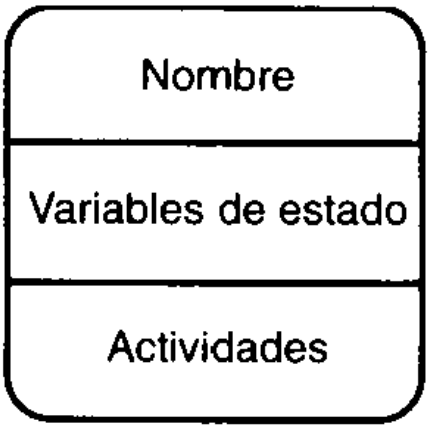

Ilustración 44: partes de un estado

Como ejemplo se muestran los dos estados en que puede estar una máquina de envío de FAX.

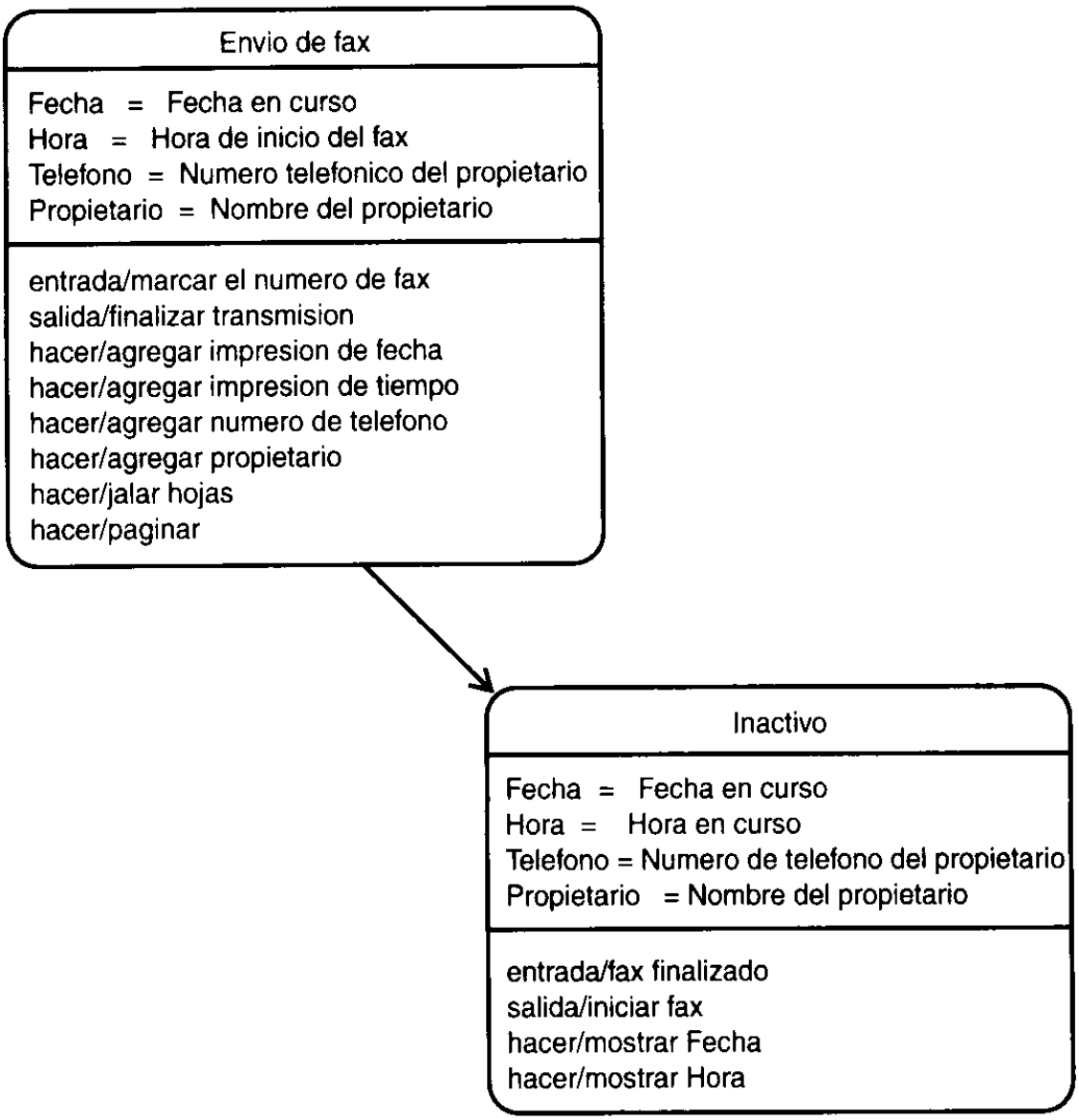

Ilustración 45: máquina de fax

6.2 Sucesos y acciones

En las líneas de transición es posible indicar ciertos detalles:

Puede indicar:

1. Un **suceso** que provoque una transición y
2. una **actividad** o acción que se realice con motivo de la transición de estados.

Ambos (suceso y actividad) se indican sobre la línea de transición, separados por una línea diagonal.

Es posible que la transición no tenga actividad (o acción) asociada. Otras veces la transición sucede únicamente por finalización de un proceso y no por un suceso (transición no desencadenada).

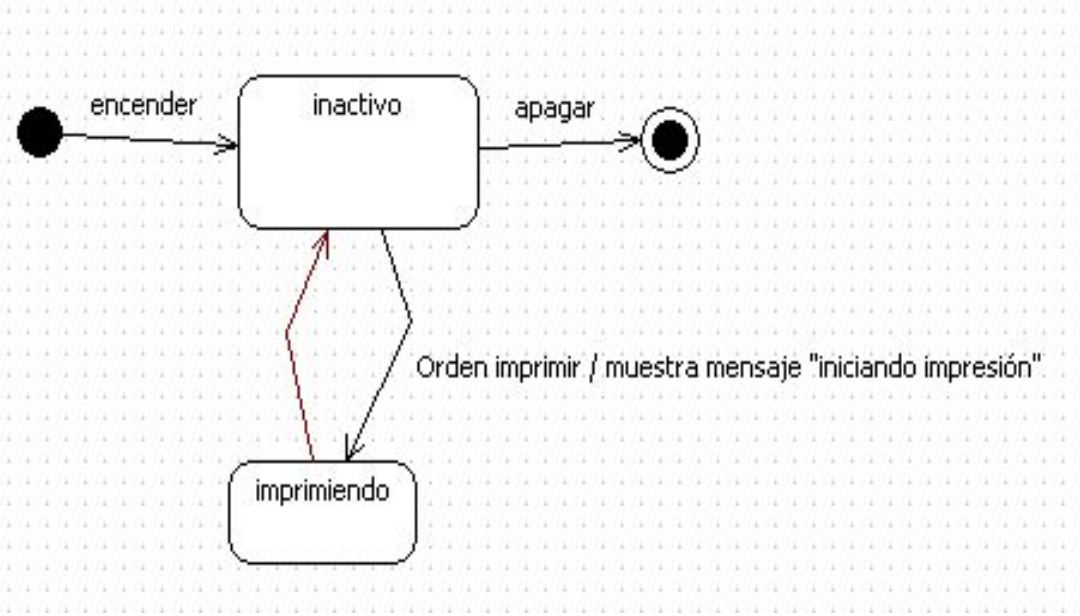

Ilustración 46: Impresora

En el diagrama se aprecian los tres casos anteriores de transición.

Ejercicio 12
Dibuja el diagrama de estados de un ascensor.
El ascensor no se moverá si se superan las 4 personas en él. Se considera que, mediante sensores, es capaz de contar cada persona que entra o sale. Tiene botones para indicar el piso al que se desea ir.
El ascensor muestra una alarma sonora y luminosa si hay más de cuatro personas, alarma que apaga en cuanto salen del ascensor las personas sobrantes. Se considera que las personas entran y salen de uno en uno.
Si se pulsa un botón de un piso, el ascensor pasa a un estado de movimiento hasta que se alcanza el piso solicitado.

6.3 Subestados

A veces un estado es algo más complejo de lo que se refleja inicialmente. En el ejemplo de la impresora, el estado imprimiendo puede ser más elaborado: comprueba papel, recoge papel, imprime, guarda bandeja; incluyendo la posibilidad de agotar el papel.

El estado imprimiendo atraviesa en ese caso varios subestados. Son estados que están dentro de otro y pueden ser de dos tipos: secuencial y concurrente.

6.3.1 Subestados secuenciales

Suceden uno a continuación del otro. Por ejemplo, para reflejar el proceso de un servidor web, podría descomponerse en los subestados secuenciales: esperando petición de cliente, resolviendo petición, enviando página a cliente.

Estos subestados pueden ser estados contenidos dentro de un estado llamado Operación, por

ejemplo.

Se representa conteniendo los subestados dentro del estado “contenedor”:

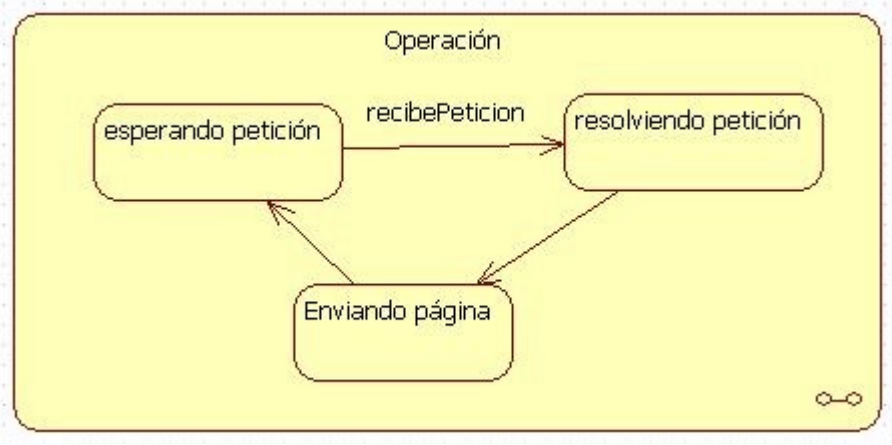

Ilustración 47: subestados secuenciales

6.3.2 Subestados concurrentes

Dentro del estado Operación del servidor web, sucede que también se hacen otras operaciones aparte de esperar peticiones de páginas web. Por ejemplo, puede suceder que exista un proceso que atienda peticiones de los administradores de las páginas web para subir o modificar nuevos arvchivos.

Todo sucede al mismo tiempo, es decir, concurrentemente. Se representa dividiendo el estado contenedor (llamado estado compuesto) en varios compartimentos separados por líneas discontinuas, donde se representan los conjuntos de subestados concurrentes.

6.4 Mensajes y señales

Si modelamos el comportamiento de determinados objetos, por ejmplo un ordenador, encontramos que los sucesos que desencadenan las transiciones son a menudo originados por otros objetos. Por ejemplo, si se encuentra activado el protector de pantalla, al pulsar una tecla del teclado o mover el ratón, el protector de pantalla desaparece.

El suceso es desencadenado por un mensaje de un objeto (Raton) a otro (GestorDeVentanas).

Si un mensaje desencadena una transición, se conoce como **“señal”**.

Las señales son también objetos y pueden componer jerarquías y tener atributos y métodos.

7 BIBLIOGRAFÍA Y ENLACES

- Aprendiendo UML en 24 horas, Joseph Schmuller , Prentice-Hall
- UML 2.0 In a Nutshell, Junio 2005, Dan Pilone, Neil Pitmande, O'Reilly
- www.omg.org (Object Management Group).
- http://www.omg.org/technology/readingroom/UML.htm