3. DIAGRAMAS DE CLASES

3.1. INTRODUCCIÓN

La notación de UML está compuesta por dos subdivisiones importantes. Hay una notación para modelar los elementos estáticos tales como clases, atributos y relaciones. También hay otra notación para modelar los elementos dinámicos tales como objetos, mensajes y máquinas de estado finitas.

Los elementos estáticos se representan mediante diagramas de estructura estática, más conocidos por su nombre corto, diagramas de clases. Muestran el conjunto de clases que forman parte de la estructura estática de un sistema, junto con las relaciones existentes entre estas clases. Pero cuando se modela la estructura estática de un sistema, podemos usar los diagramas de clases de tres formas diferentes:

1. para modelar el vocabulario de un sistema

Modelar el vocabulario de un sistema implica tomar una decisión sobre qué abstracciones forman parte del sistema y qué otras caen fuera de sus límites. Aquí usamos los diagramas de clases para especificar dichas abstracciones y sus responsabilidades.

2. para modelar colaboraciones de forma sencilla

Una colaboración es un conjunto de clases, interfaces y otros elementos que trabajan juntos para proporcionar un comportamiento de cooperación mayor que la suma de todos los elementos. Por ejemplo, cuando queremos modelar las semánticas de una transacción en un sistema distribuido, no podemos fijarnos en una sola clase para entender lo que está sucediendo. Es mejor que estas semánticas sean llevadas a cabo por un conjunto de clases que trabajen juntas. Aquí usamos los diagramas de clases para visualizar y especificar este conjunto de clases y sus relaciones.

3. para modelar el esquema lógico de una base de datos

En muchos dominios podemos querer almacenar información continua (*persistent information*) en una base de datos relacional u orientada a objetos. Aquí usamos los diagramas de clase para modelar los esquemas de dichas bases de datos.

Nosotros vamos a estudiar los diagramas de clases que se utilizan para modelar el vocabulario de un sistema.

3.2. DIAGRAMAS DE CLASES

Los diagramas de clases se utilizan para modelar la visión estática de un sistema. Esta visión soporta los requisitos funcionales del sistema, en concreto, los servicios que el sistema debería proporcionar a sus usuarios finales. Normalmente contienen: clases, interfaces y relaciones entre ellas: de asociación, de dependencia y/o de generalización.

Los diagramas de clases también pueden contener a paquetes o subsistemas, que se usan para agrupar elementos del modelo en partes más grandes (por ejemplo, paquetes que a su vez contienen a varios diagramas de clases).

Al igual que otros diagramas, en los diagramas de clases pueden aparecer notas explicativas y restricciones.

3.2.1. Perspectivas

Según Fowler, hay tres perspectivas que podemos tener en cuenta a la hora de dibujar los diagramas de clases (o cualquier otro tipo de diagrama, pero es más evidente en los diagramas de clases):

- Conceptual: Estamos dibujando un diagrama que representa los conceptos del dominio del sistema. Estos conceptos se relacionarán de forma natural con las clases que los implementen, pero a menudo no hay una aplicación directa. Es decir, el modelo se dibuja sin tener en cuenta el software que lo implementa y generalmente es independiente del lenguaje de programación.

- Análisis: Desde el punto de vista del software, nos fijamos en las interfaces del software, no en su implementación. Es decir, miramos los tipos más que las clases.

  El desarrollo orientado a objetos pone mucho énfasis en la diferencia entre tipo y clase, pero luego no se aplica en la práctica. Es importante separar interfaz (tipo) de implementación (clase). Muchos lenguajes de programación no lo hacen y los métodos, influidos por ellos, tampoco. Esto está cambiando, pero no lo suficientemente rápido (Java y CORBA tendrán algo de influencia aquí). Los tipos representan una interfaz que puede tener muchas implementaciones diferentes debido, por ejemplo, a las características del entorno de instalación.

- Implementación: Estamos poniendo la implementación al descubierto, pues realmente tenemos clases. Es la perspectiva más utilizada, pero de todas formas la perspectiva del análisis se considera la mejor de las tres.

3.2.2. Clases

Las clases describen un conjunto de objetos con características y comportamiento idénticos, es decir, objetos que comparten los mismos atributos, operaciones y relaciones.

Las clases se representan gráficamente por medio de un rectángulo con tres divisiones internas. Los tres compartimentos alojan el nombre de la clase, sus atributos y sus operaciones, respectivamente. En muchos diagramas se omiten los dos compartimentos inferiores. Incluso cuando están presentes, no muestran todos los atributos y todas las operaciones. Por ejemplo, en la Figura 3.1 viene representada la clase Ventana de tres formas diferentes: sin detalles, detalles en el ámbito de análisis y detalles en el ámbito de implementación.

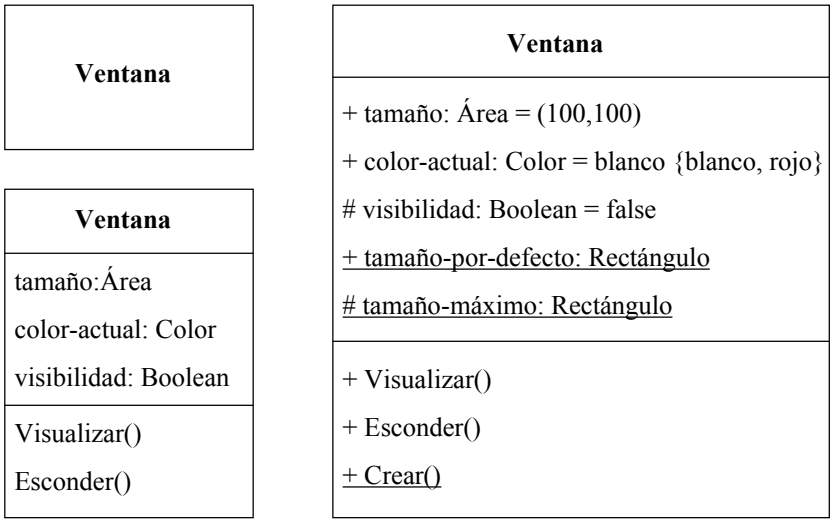

Figura 3.1

3.2.2.1. Compartimento del nombre

Cada clase debe tener un nombre que la distinga de las otras clases. Dicho nombre puede ser un nombre simple (nombre de la clase) o un nombre compuesto (nombre del paquete que contiene a la clase :: nombre de la clase).

En el ejemplo de la Figura 3.1 Ventana es un nombre simple de clase; pero si estuviese contenida en el paquete Gráficos, entonces el nombre compuesto sería Gráficos::Ventana. Otra forma de representar dicho nombre compuesto es escribir dentro de este compartimento primero el nombre del paquete (en este caso, «Gráficos») y debajo el nombre de la clase contenida en él.

3.2.2.2. Compartimento de la lista de atributos

Los atributos describen las características propias de los objetos de una clase. La sintaxis completa de un atributo es:

> [visibilidad] nombre [multiplicidad] [: tipo] [= valor-inicial]
> [{lista-propiedades}]
>
> donde visibilidad puede ser:
>
> - + (pública): que hace el atributo visible a todos los clientes de la clase;
> - - (privada): que hace el atributo visible sólo para la clase;
> - # (protegida): que hace el atributo visible a las subclases de la clase.
>
> Si un atributo no tiene asignada ninguna visibilidad, quiere decir que la visibilidad no está definida (no hay visibilidad por defecto).

donde nombre es una cadena de texto que representa el nombre del atributo;

donde tipo es un tipo de atributo típico: string, boolean, integer, real, double, point, area y enumeration. Se llaman tipos primitivos. También pueden ser específicos de un cierto lenguaje de programación, aunque se puede usar cualquier tipo, incluso otras clases.

donde multiplicidad es un indicador de la multiplicidad del atributo, que va encerrado entre corchetes. La ausencia de multiplicidad significa que tiene exactamente valor 1, mientras que una multiplicidad de 0..1 proporciona la posibilidad de valores nulos (la ausencia de un valor).

donde valor-inicial es una expresión con el valor inicial que va a contener el atributo cuando se crea de nuevo el objeto;

donde lista-propiedades es una lista que contiene los valores permitidos en un atributo. Se utiliza para especificar tipos enumerados tales como color, estado, etc.

Ciertos atributos pueden ser visibles globalmente, en toda la amplitud léxica de la clase. Para ello se definen como atributos cuyo ámbito es la clase (*class-scope attributes*), llamados también variables de clase (*class variables*), y se representan como los objetos por un nombre subrayado. La notación se justifica por el hecho de que una variable de clase aparece como un objeto compartido por las instancias de una clase.

3.2.2.2.1. Ejemplos de atributos

En el ejemplo de la Figura 3.1 la clase Ventana tiene los siguientes atributos:

+ tamaño: Área = (100,100)
+ color-actual: Color = blanco {blanco, rojo}
# visibilidad: Boolean = false
+ tamaño-por-defecto: Rectángulo
+ tamaño-máximo: Rectángulo

Los atributos tamaño-por-defecto y tamaño-máximo tienen por tipo la clase Rectángulo, mientras que el resto corresponde a tipos primitivos: Área, Color y Boolean. Asimismo, estos dos atributos, que aparecen subrayados, son atributos cuyo ámbito es la clase.

3.2.2.3. Compartimento de la lista de operaciones

Las operaciones describen el comportamiento de los objetos de una clase. La sintaxis completa de una operación es:

[visibilidad] nombre [(lista-parámetros)] [: tipo-devuelto]
[{lista-propiedades}]

donde visibilidad puede ser:

- + (pública)
- - (privada)
- # (protegida)

Si una operación no tiene asignada ninguna visibilidad, quiere decir que la visibilidad no está definida (no hay visibilidad por defecto).

donde nombre es una cadena de texto que identifica la operación;

donde lista-parámetros es una lista de parámetros formales separados por una coma, cada uno especificado según la siguiente sintaxis:

nombre [: tipo] [= valor-por-defecto]

donde tipo-devuelto es una especificación del tipo que representa el valor devuelto por la operación. Depende del lenguaje de programación. Si la operación no devuelve ningún valor (por ejemplo, void en C++), se omite.

donde lista-propiedades es una lista que contiene los valores permitidos en una operación.

Una clase puede tener operaciones cuyo ámbito es la clase (*class-scope operations*). Aunque no exista un objeto de la clase, se puede llamar a una operación de este tipo, pero su acceso está restringido sólo a los atributos cuyo ámbito es la clase (*class-scope attributes*). Las operaciones cuyo ámbito es la clase se definen para llevar a cabo operaciones genéricas tales como crear objetos y encontrar objetos donde un objeto específico no está implicado. Se representan por un nombre junto con sus argumentos, todos ellos subrayados.

3.2.2.3.1. Ejemplos de operaciones

En el ejemplo de la Figura 3.1 la clase Ventana tiene las siguientes operaciones:

+ Visualizar()
+ Esconder()
+ Crear()

En estas operaciones no se ha especificado ninguno de los argumentos mencionados antes, excepto el de la visibilidad. La operación Crear es una operación cuyo ámbito es la clase, pues aparece subrayada.

3.2.3. Relaciones en un diagrama de clases

Los diagramas de clases están compuestos por clases y por relaciones entre ellas. Las relaciones que se pueden usar son:

- Relación de asociación

Una asociación es una conexión entre clases, una conexión semántica (enlace) entre los objetos de dichas clases. Un tipo especial de asociación es la relación de agregación.

- Relación de dependencia

Una dependencia es una relación entre elementos, uno independiente y otro dependiente. Un cambio en el elemento independiente afectará al elemento dependiente.

- Relación de generalización

Una generalización es una relación entre un elemento más general y otro más específico. El elemento más específico puede contener sólo información adicional. Una instancia (un objeto es una instancia de una clase) del elemento más específico se puede usar si el elemento más general lo permite.

3.2.3.1. Asociación

Una asociación es una relación estructural que especifica que los objetos de una clase están conectados con los objetos de otra. Cuando una asociación es una conexión semántica entre los objetos de dos clases, se llama asociación binaria. Aunque no es lo común, se pueden tener asociaciones que conecten más de dos clases; éstas se llaman asociaciones n-arias. También es posible que, dado un objeto de una clase, se pueda enlazar con otros objetos de la misma clase.

Normalmente una asociación es binaria y bidireccional (se puede navegar en ambas direcciones). Se dibuja como una línea sólida entre dos clases. También se puede representar una asociación binaria y unidireccional, añadiendo una flecha al final de la línea. Esta flecha indica que la asociación sólo se puede utilizar en la dirección de la flecha.

Cada asociación puede presentar algunos elementos adicionales que dan detalle a la relación, como son:

- Nombre

Una asociación puede tener un nombre, que se usa para describir la naturaleza de la relación. Así no hay ambigüedad sobre su significado. Para indicar la dirección en que se debe leer el nombre se emplea un triángulo.

- Rol

Cuando una clase participa en una asociación, juega un rol específico en dicha relación. Se puede designar de forma explícita mediante un nombre a los finales de la línea, el cual describe la semántica de la asociación en el sentido indicado.

- Multiplicidad

La multiplicidad describe la cardinalidad de la relación, es decir, cuántos objetos están conectados en una instancia de una asociación (enlace). Cuando se establece una multiplicidad al final de la línea de una asociación, indica que para cada objeto de la clase en el lado opuesto existen varios objetos en el otro extremo. El rango puede ser tres (3), cero-a-uno (0..1), cero-a-muchos (0..* ó *), uno-a-muchos (1..*), etc.

- Restricciones

Los elementos anteriores son suficientes para detallar una relación de asociación. Pero si queremos especificar un mayor significado, UML define cinco restricciones que se

pueden aplicar. Por ejemplo, si queremos que los objetos de una clase al final de una asociación (con una multiplicidad mayor que uno) estén en un orden explícito, debemos escribir {ordenado} al final de la misma y cerca de la clase.

En la Figura 3.2 se muestran los dos tipos de navegación en una asociación entre las clases Persona y Coche. En el primer caso un coche puede tener uno o más propietarios y una persona es propietaria de cero o más coches. Mientras que en el segundo caso la asociación navegable nos dice que una persona puede poseer varios coches o ninguno, pero no nos informa sobre cuántas personas son propietarias de un coche.

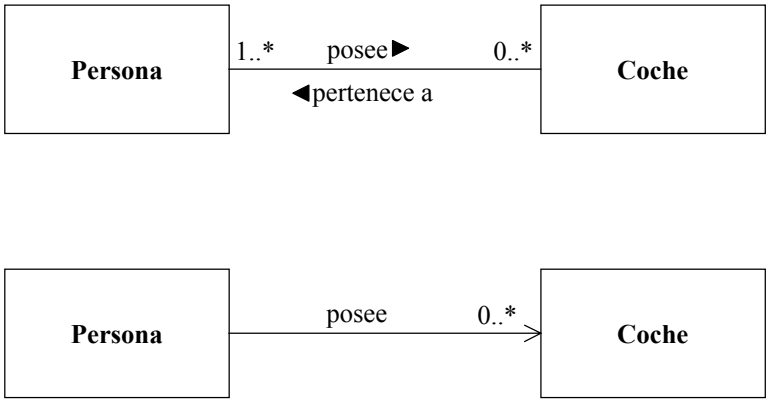

Figura 3.2

3.2.3.1.1. Tipos de asociación

Los tipos de asociaciones presentes en un diagrama de clases son:

1. Asociación normal

Una asociación normal entre dos clases representa una relación estructural entre sus objetos, lo que significa que ambas clases están conceptualmente al mismo nivel, ninguna es más importante que la otra. Se trata de una relación no muy fuerte.

2. Agregación

Una agregación sirve para modelar una relación "todo-parte", lo que significa que un objeto del todo tiene objetos de la parte. Podemos distinguir:

- Agregación normal

  Una agregación normal se denota dibujando una línea con un rombo sin rellenar al final de la misma del lado del todo (del lado de la clase que contiene a la otra clase). También puede tener un nombre (con dirección) y ser navegable, así como añadir roles en el lado de la parte (lado de la clase contenida).

Una agregación normal es un caso especial de la multiplicidad de una asociación, pues la cardinalidad del lado del todo sólo puede ser uno.

- Agregación compartida

Una agregación compartida es aquella en la que las partes pueden ser partes en varios todos. Se representa de la misma forma que una agregación normal. Para que una agregación se considere compartida es necesario que la multiplicidad en el lado del todo sea superior a uno.
Una agregación compartida es un caso especial de una agregación normal.

- Agregación de composición

Una agregación de composición se denota dibujando una línea con un rombo relleno al final de la misma del lado del todo (del lado de la clase que contiene a la otra clase). La multiplicidad en el lado del todo puede ser uno (1) ó cero-a-uno (0..1), pero la multiplicidad en el lado de la parte puede ser cualquier intervalo. Se trata de una relación de pertenencia muy fuerte.

3.2.3.1.2. Diferencias entre los tipos de asociación

La asociación expresa un acoplamiento débil; las clases asociadas siguen siendo relativamente independientes una de otra. Se trata de una relación no muy fuerte, es decir, no se exige dependencia existencial ni encapsulamiento.

La agregación es una forma particular de asociación que expresa un acoplamiento más fuerte entre clases; una de las clases cumple una función más importante que la otra en la relación. La agregación normal y compartida son relaciones no muy fuertes, con las mismas características que la asociación a la hora de implementarlas. Pero la agregación de composición es una asociación muy fuerte, que implica dependencia existencial (la clase dependiente desaparece al destruirse la que la contiene y, si es de cardinalidad uno, es creada al mismo tiempo) y pertenencia fuerte (la clase contenida es parte constitutiva y vital de la que la contiene). Una forma de implementar una agregación de composición es incluir físicamente las partes dentro de la clase del todo (encapsulamiento). En la forma de implementación es donde podemos hallar otra diferencia entre los tipos de agregaciones. Mientras que en la agregación de composición la clase del todo tiene el control del ciclo de vida de sus partes (al mismo tiempo que la clase es eliminada, se destruyen las partes), en las otras dos los tiempos de vida de la clase del todo y de sus partes no coinciden.

3.2.3.1.3. Ejemplos de asociaciones normales

En la Figura 3.3 podemos ver las clases y las relaciones de asociación normal, junto con sus elementos adicionales, correspondientes a una compañía de seguros. Observando el ejemplo, podemos darnos cuenta que:

- Una compañía de seguros tiene contratos de seguros, cada uno de los cuales se refieren a uno o más clientes.
- Un cliente tiene cero o más contratos de seguros, cada uno referido a una compañía de seguros.

- En consecuencia, un contrato de seguros se refiere a uno o varios clientes y a una compañía de seguros.
- El contrato de seguros está expresado en una (cero o una) póliza de seguros.
- La póliza de seguros se refiere a una compañía de seguros.

La multiplicidad especifica que el contrato de seguros está expresado en una (cero o una) póliza de seguros (contrato escrito) y que la póliza de seguros se refiere a una compañía de seguros. Si llamas por teléfono a la compañía y aseguras tu coche, hay un contrato de seguros, pero no una póliza. La póliza de seguros será enviada más tarde por correo.

Es importante modelar el mundo real. Si hubiésemos modelado la compañía de seguros basándonos sólo en la póliza de seguros, habrían surgido problemas. Por ejemplo, qué ocurriría si un cliente, que ha asegurado el coche por teléfono, tuviese un accidente con él un minuto más tarde (no hay póliza de seguros, pero sí un acuerdo oral con el agente).

Una clase puede jugar diferentes roles en diferentes asociaciones. En el diagrama de clases de la Figura 3.3, la clase Compañía de Seguros juega el papel de aseguradora y la clase Persona el papel de marido, mujer o asegurado.

Si nos fijamos en el ejemplo, todas las clases están conectadas mediante asociaciones binarias, excepto la clase Persona, donde un objeto de esta clase se puede enlazar con otros objetos de la misma clase. Un marido está casado con una mujer. Tanto el marido como la mujer son personas. Si una persona no está casada, entonces ni él ni ella juegan el papel de marido o mujer, lo que significa que la asociación (en este caso, casado con) no se aplica.

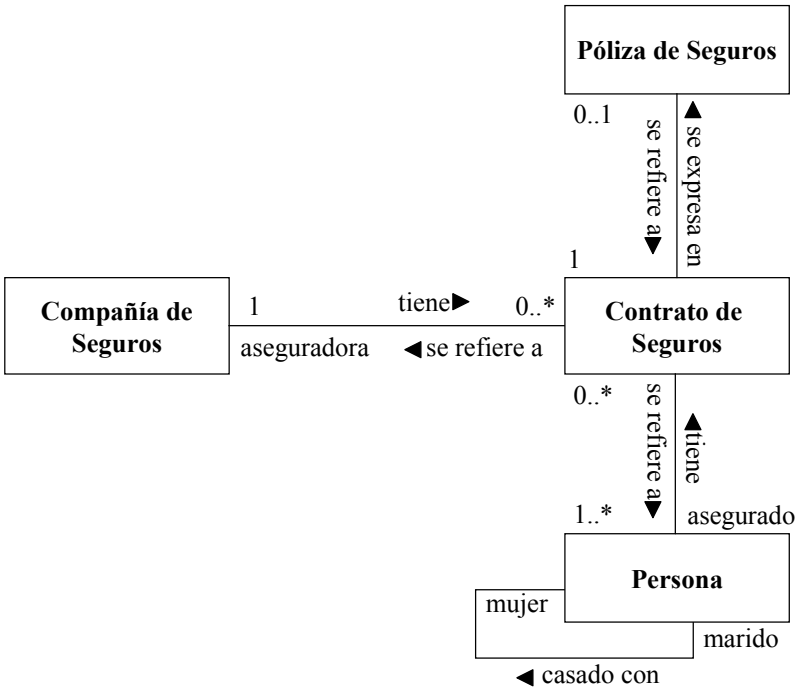

Figura 3.3

3.2.3.1.4. Ejemplos de agregaciones

En la Figura 3.4 podemos encontrar tres ejemplos con los tres tipos de agregaciones: normal, compartida y la de composición.

El primer ejemplo se trata de una agregación normal. La flota contiene muchos barcos de guerra. Algunos barcos pueden ser eliminados y todavía es una flota. Lo mismo ocurre si se añaden barcos, pues sigue siendo una flota. Esto es significativo para una agregación normal, pero no para una agregación de composición. Las partes (los barcos de guerra) componen el todo (flota). El rombo sin rellenar en el lado del todo, junto con la multiplicidad 1, nos indica que se trata de una agregación normal.

En el segundo ejemplo la agregación es compartida. Un remix (mezcla de sonidos) está compuesto de varias bandas sonoras; la misma banda sonora podría formar parte de varias mezclas musicales. Se diferencia de la agregación normal en que la multiplicidad en el lado del todo es superior a 1. Los objetos tanto de la clase Remix como de la clase Banda Sonora se encuentran ordenados, lo cual está indicado por la restricción {ordenado}.

El tercer ejemplo se trata de una agregación de composición. La ventana contiene muchos textos, cajas de listado de elementos, botones y menús. El tiempo de vida de las partes coincide con la clase del todo, en este caso, la clase Ventana. El rombo relleno en el lado del todo, junto con la multiplicidad 0..1 o 1, nos indica que se trata de una agregación de composición.

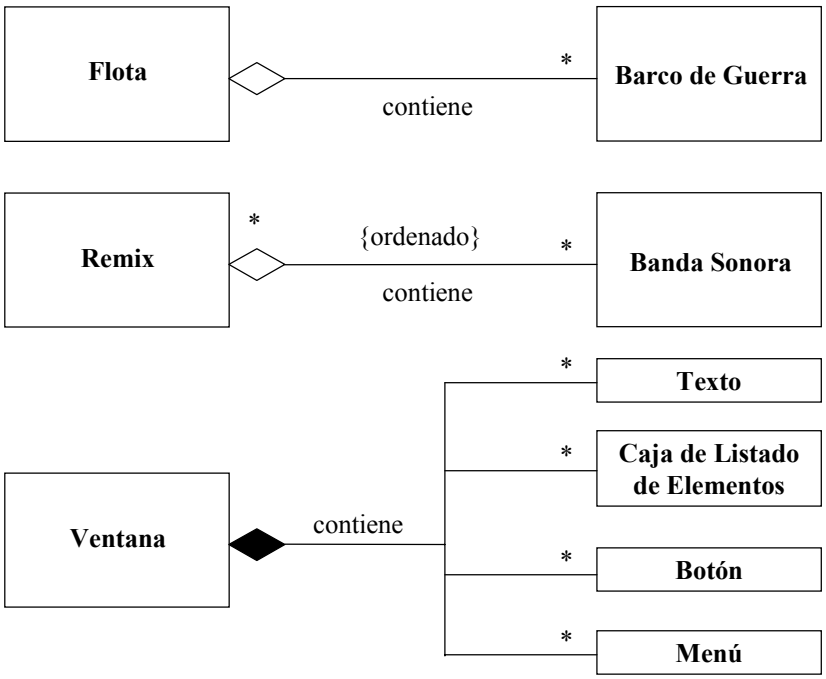

Figura 3.4

3.2.3.2. Dependencia

Una dependencia es una conexión semántica entre dos elementos de modelado, uno independiente y otro dependiente. Un cambio en el elemento independiente afectará al dependiente. Un elemento de modelado puede ser una clase, un paquete, un caso de uso, etc. Los casos más típicos de dependencia son: una clase tiene por parámetro en una operación un objeto de otra clase; una clase accede a un objeto global de otra clase; o una clase llama a una operación cuyo ámbito es la clase en otra clase.

La relación de dependencia se representa por una línea discontinua con una flecha en el lado del elemento independiente. Si se utiliza una etiqueta, entonces se trata de un estereotipo que identifica el tipo de dependencia.

En UML existen ocho estereotipos que se pueden aplicar a las relaciones de dependencia entre clases:

- bind

Se utiliza cuando se quiere modelar los detalles de las clases contenidas en plantillas. La relación entre una plantilla que contiene una clase y una instanciación de dicha clase se podría modelar como una dependencia *«bind»*. Bind incluye una lista de los argumentos actuales que se aplican a los argumentos formales de la plantilla.

- derive

Se utiliza cuando se quiere modelar la relación entre dos atributos o dos asociaciones, uno de los cuales es concreto y el otro es conceptual. Por ejemplo, la clase Persona podría tener el atributo fecha-de-cumpleaños (que es concreto), así como el atributo edad (que se puede derivar a partir del atributo anterior). Se podría mostrar esta relación mediante una dependencia *«derive»*, indicando que edad se deriva a partir de fecha-de-cumpleaños.

- friend

Se utiliza cuando se quiere modelar que una clase tenga acceso especial a la estructura interna de otra clase (incluso a aquellas partes con visibilidad privada). La dependencia *«friend»* es una reminiscencia de las clases amigas de C++.

- instanceOf
- instantiate

Estos dos estereotipos se utilizan para modelar de forma explícita las relaciones clase/objeto.
La dependencia *«instanceOf»* se utiliza cuando se quiere modelar la relación entre una clase y un objeto en el mismo diagrama, o entre una clase y su metaclase.
La dependencia *«instantiate»* se utiliza cuando se quiere especificar qué elemento crea objetos de otro.

- powertype

Se utiliza cuando se quiere modelar clases que encubran a otras clases, tal como sucede cuando se trata de modelar bases de datos.

- refine

Se utiliza cuando se quiere modelar clases que son esencialmente la misma, pero a diferentes niveles de abstracción. Por ejemplo, durante el análisis, tenemos una clase Cliente que, durante el diseño, vamos a refinar en una clase Cliente más detallada, es decir, más completa de cara a su implementación.

- use

Se utiliza cuando se quiere marcar de forma explícita una dependencia como una relación de uso, en contraste con los matices que proporcionan los otros estereotipos de dependencia.

3.2.3.2.1. Ejemplos de dependencias

La Figura 3.5 nos muestra un conjunto de cuatro clases junto con sus asociaciones y sus dependencias, correspondientes a una parte de un sistema que gestiona la asignación de profesores a los cursos en una universidad.

En el ejemplo hay una dependencia normal de la clase Planificación del Curso respecto a la clase Curso, pues Curso se utiliza como parámetro en las operaciones Añadir y Eliminar de Planificación del Curso.

Si proporcionamos la lista de parámetros en las operaciones, tal como se puede observar en el ejemplo, entonces no es necesario mostrar la dependencia, pues la utilización de la clase Curso ya figura de forma explícita en la clase Planificación del Curso. Sin embargo, a veces queremos que esta dependencia se dibuje, especialmente si hemos omitido la lista de parámetros en las operaciones.

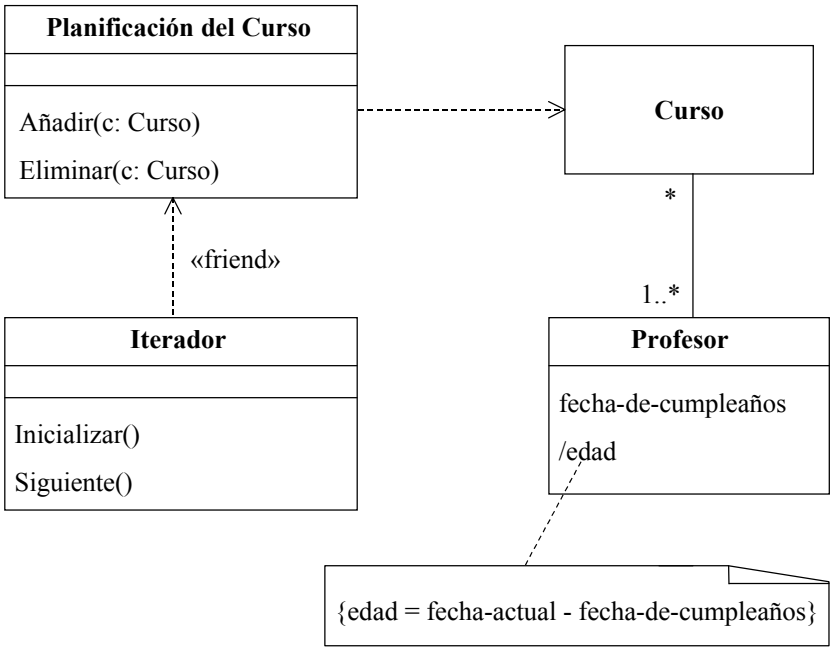

Figura 3.5

En el ejemplo aparecen otras dos dependencias marcadas con un estereotipo. Una de ellas es la dependencia *«derive»* entre los atributos edad y fecha-de-cumpleaños, la cual no se marca de forma explícita, sino como: /edad. La otra es la dependencia *«friend»* de la clase Iterador respecto a la clase Planificación del Curso. Esta dependencia muestra el uso de un iterador, especificado en la clase Iterador, sobre la programación del curso, especificada en la clase Planificación del Curso. Sin embargo, esta última clase no sabe nada acerca de la primera.

3.2.3.3. Generalización

Una generalización es una relación de clasificación entre un elemento más general y otro más específico. La generalización es utilizada por clases y casos de uso y por otros elementos de modelado como los paquetes.

La generalización normal es una relación de herencia entre clases, entre una general y otra específica. La clase específica, llamada subclase, hereda todo de la clase general, llamada superclase. Los atributos, las operaciones y todas las asociaciones son heredadas. Los atributos y las operaciones con visibilidad pública en la superclase serán también públicos en la subclase. Los atributos y las operaciones que tengan visibilidad privada también serán heredados, pero no serán accesibles dentro de la subclase. Para proteger los atributos y las operaciones de accesos desde fuera de la superclase y de la subclase, los podemos declarar con visibilidad protegida. No se puede acceder a ellos desde otras clases, pero están disponibles para la superclase y para cualquiera de sus subclases.

La relación de generalización se representa por una línea continua desde la clase más específica (subclase) hasta la clase más general (superclase), con un triángulo sin rellenar al final de la línea en la superclase.

Una clase abstracta es aquella que no tiene objetos. Sólo se utiliza para heredar a partir de ella, es decir, en una clase abstracta se describen los atributos y las operaciones comunes para otras clases. Se especifica de forma explícita poniendo {abstracta} dentro del compartimento del nombre de la clase y debajo de su nombre. Normalmente una clase abstracta tiene operaciones abstractas. Una operación abstracta es aquella que no tiene método de implementación en la superclase donde está definida. Se especifica de forma explícita escribiendo {abstracta} después del nombre de la operación en la superclase.

Lo contrario de una clase abstracta es una clase concreta, lo que significa que es posible crear objetos a partir de la clase y que tiene implementaciones para todas las operaciones.

A la hora de modelar aplicaciones sencillas es suficiente con la relación de generalización normal. Sin embargo, el modelado de ciertos dominios complejos (como puede ser el modelado de todas las aplicaciones de una empresa) requiere especificar un mayor significado, debido principalmente a la existencia de un gran número de subclases. Por ello UML define un estereotipo y cuatro restricciones que se pueden aplicar. El estereotipo es *«implementation»*. Las restricciones, que nos indican si se pueden añadir o no más hijos en la generalización, son: {completo} e {incompleto}. Las otras dos restricciones son: {disjoint} y {overlapping} y se aplican sólo en el contexto de herencia múltiple, es decir, cuando una clase tiene más de un padre.

3.2.3.3.1. Ejemplos de generalizaciones

La Figura 3.6 nos muestra un conjunto de cuatro clases junto con sus asociaciones y dependencias, referidas a tipos de vehículos.

En nuestro ejemplo la clase Vehículo muestra las cosas comunes entre coches y barcos, incluyendo la relación de asociación a la clase Persona. Se trata de una clase abstracta que no tiene objetos, mientras que las clases Coche y Barco son concretas. Estas subclases heredan el atributo color y la operación Conducir de la superclase Vehículo. Dicha operación está redefinida en las clases Coche y Barco, ya que no está implementada en la clase Vehículo al aparecer junto a la operación la palabra {abstracta} y además su implementación es diferente dependiendo del tipo de objeto: coche o barco.

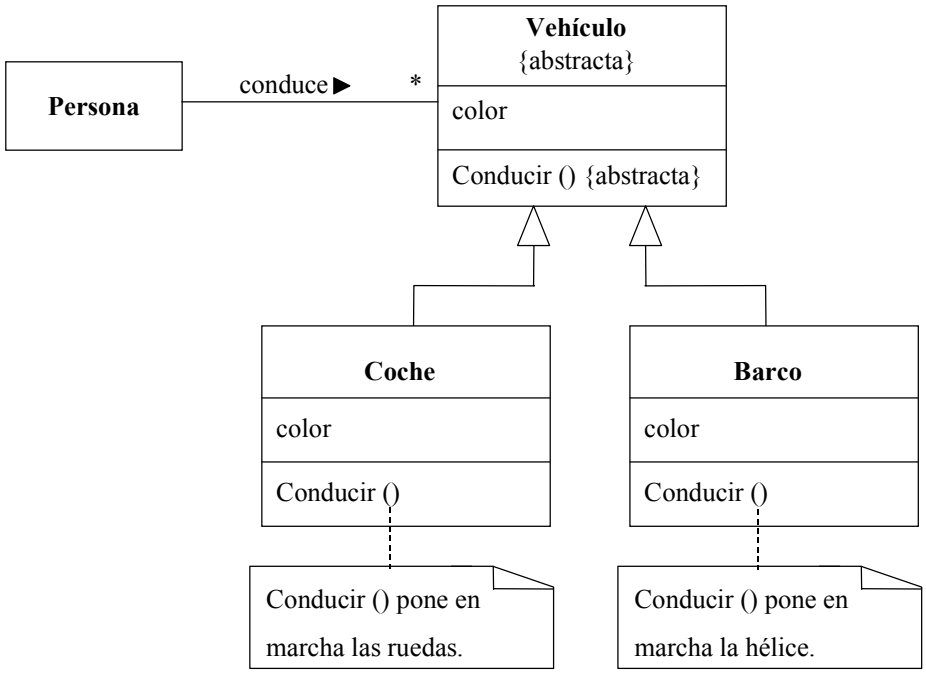

Figura 3.6

La Figura 3.6 nos muestra que la clase Persona tiene una asociación, llamada conduce, hacia la clase Vehículo. La clase Vehículo es abstracta; lo que significa que los objetos actuales que la persona conduce se obtienen a partir de las subclases concretas Coche y Barco. Cuando la persona ejecuta la operación Conducir, el resultado depende de si el objeto usado en dicha situación es un coche o un barco. Si es un objeto de la clase Coche, se pondrán en marcha las ruedas (utilizando la implementación especificada en la clase Coche). Si es un objeto de la clase Barco, se pondrá en marcha la hélice (utilizando la implementación especificada en la clase Barco). Esta técnica, donde un objeto de una subclase actúa según un objeto de una superclase y una o más operaciones en la superclase son redefinidas en la subclase, se llama polimorfismo.

3.2.4. Notas

Una nota es un comentario dentro de un diagrama cualquiera. Puede estar relacionado con uno o más elementos en el diagrama mediante líneas punteadas.

Una nota se representa gráficamente mediante un rectángulo con su borde superior derecho doblado. Sirve para representar aclaraciones al diagrama o restricciones sobre los elementos relacionados (cuando el texto se encuentre entre "{" y "}). Estas dos

representaciones se muestran en los ejemplos de la Figura 3.6 y la Figura 3.5, respectivamente.

3.2.5. Restricciones

Una restricción es una condición que limita el uso del elemento o las semánticas (significado) del elemento. Se declara como una cadena encerrada entre "{" y "}" y situada cerca del elemento asociado.

Las restricciones se pueden escribir como un texto en formato libre. Pero si queremos especificar las semánticas de forma más precisa, entonces debemos usar el lenguaje OCL (*Object Constraint Language*) de UML. De todas maneras, existe un número de restricciones ya definidas en UML que se pueden utilizar en cualquier diagrama.

3.2.6. Clases avanzadas (clasificadores)

Las clases constituyen el elemento de construcción más importante de cualquier sistema orientado a objetos. Sin embargo, las clases son uno de los tipos de un bloque de construcción más general en UML, los clasificadores.

Un clasificador es un mecanismo que describe las características de estructura y de comportamiento. UML proporciona un gran número de tipos de clasificadores, aparte de las clases, que nos pueden ayudar a modelar nuestro proyecto del mundo real.

- Interfaz: Un conjunto de operaciones que se utilizan para especificar un servicio de una clase o de un componente.
- Tipo de datos (*datatype*): Un tipo cuyos valores no tienen identidad, incluyendo tipos primitivos (tales como enteros y cadenas de caracteres), así como tipos enumerados (tales como booleanos).
- Señal (*signal*): La especificación de un estímulo asíncrono comunicado entre instancias.
- Componente: Una parte física de un sistema que proporciona la ejecución de un conjunto de interfaces.
- Nodo: Un elemento físico en tiempo de ejecución, que representa un recurso de ordenador, generalmente con memoria y capacidad de procesamiento.
- Caso de uso: Una descripción de un conjunto de acciones, incluyendo variantes, que un sistema lleva a cabo tras recibir un estímulo por parte de un actor en particular.
- Subsistema: Un grupo de elementos de los cuales algunos constituyen una especificación del comportamiento ofrecido por los otros elementos contenidos en el mismo.

Todos los clasificadores anteriores se representan gráficamente de distinta forma, tal como muestra la Figura 3.7.

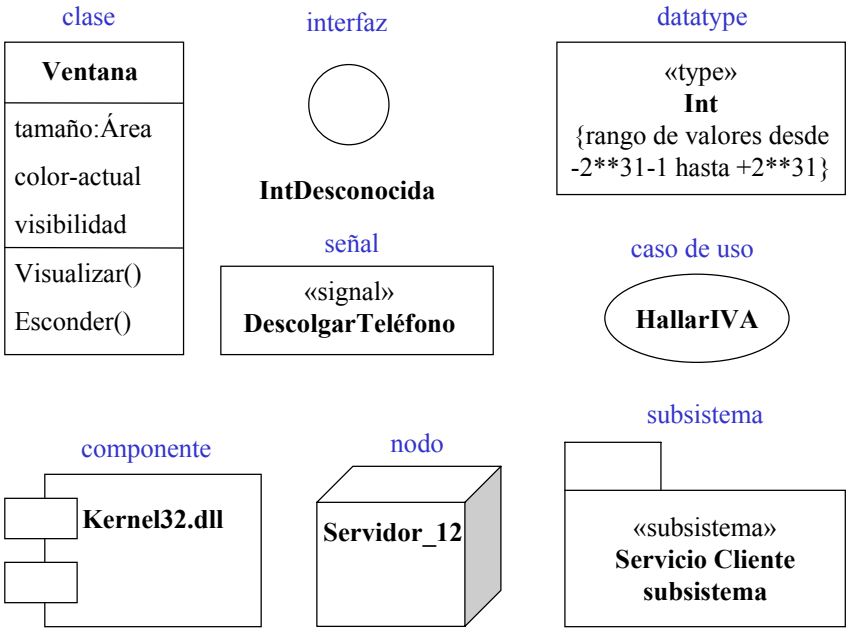

Figura 3.7

Un clasificador tiene características de estructura (en la forma de atributos), así como características de comportamiento (en la forma de operaciones). Las interfaces son la única excepción, pues no pueden tener atributos.

3.2.6.1. Interfaces

Una interfaz describe sólo operaciones abstractas, que especifican un comportamiento que un elemento (paquete, componente o clase) puede elegir soportar implementando la interfaz. A diferencia de las clases, las interfaces no especifican ninguna estructura (por ello no incluyen atributos) ni ninguna implementación (por ello no incluyen métodos, los cuales proporcionan la implementación de una operación). Estas operaciones abstractas pueden ser adornadas con propiedades de visibilidad y con restricciones. Al igual que las clases, las interfaces pueden participar en relaciones de asociación, dependencia y generalización.

Las interfaces se representan gráficamente por medio de un círculo pequeño con un nombre situado debajo de él. Se conectan a sus elementos mediante una línea continua, que representa una relación de asociación con multiplicidad 1-1.

Una clase que usa la interfaz implementada en una clase específica se conecta al círculo de dicha interfaz mediante una relación de dependencia (línea discontinua). La clase dependiente sólo depende de las operaciones de la interfaz, no depende de ninguna de las operaciones de la clase que la implementa (en cuyo caso, la relación de dependencia debería apuntar al símbolo de dicha clase). La clase dependiente podría llamar a las operaciones declaradas en la interfaz, las cuales no se muestran directamente en el diagrama. Para verlas es necesario que la interfaz sea especificada como una clase con el estereotipo *«interfaz»* dentro del rectángulo y por encima de su nombre.

En el primer ejemplo de la Figura 3.8 la clase Persona implementa la interfaz Almacenable.

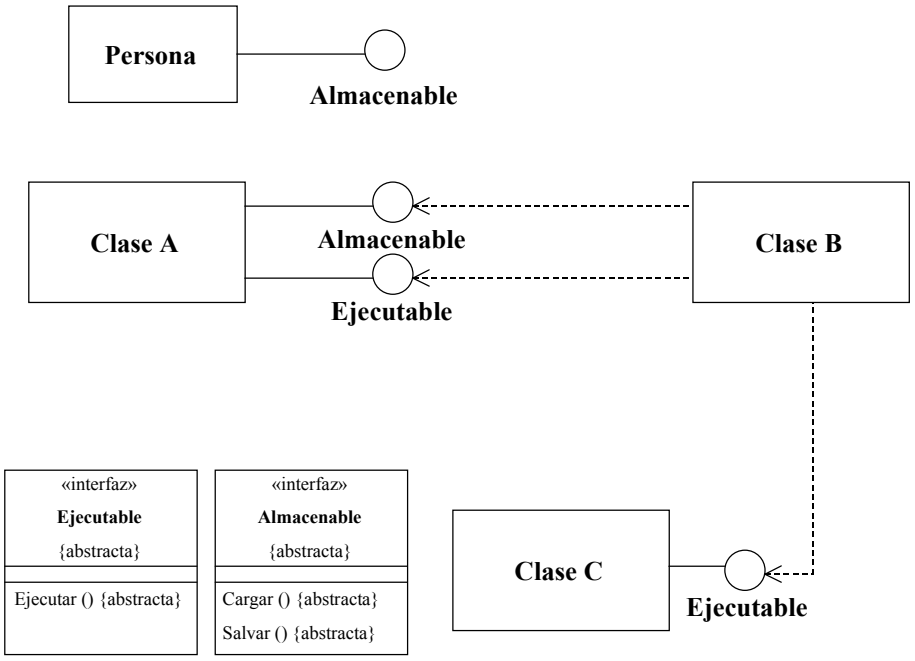

Figura 3.8

En el segundo ejemplo de la Figura 3.8 la clase A implementa las interfaces Ejecutable y Almacenable y la clase C la interfaz Ejecutable. La clase B usa las operaciones de las interfaces Ejecutable y Almacenable implementadas en A y las de Ejecutable implementada en C. Las interfaces son especificadas como clases con el estereotipo *«interfaz»* y contienen las operaciones abstractas que las clases que las implementan tienen que implementar.

3.2.6.2. Paquetes

Un paquete es un mecanismo para agrupar clases u otros elementos de otro tipo de diagramas en modelos más grandes, donde dichos elementos pueden ser linkados. Muchas metodologías de OO usan el término subsistema para describir un paquete.

Los paquetes o subsistemas se representan gráficamente por medio de un rectángulo grande junto con otro más pequeño situado en la esquina superior izquierda del rectángulo mayor. Si los contenidos de los paquetes no se muestran, entonces el nombre del paquete debe ir dentro del rectángulo grande; en caso contrario dentro del rectángulo pequeño.

Las relaciones permitidas entre paquetes son dependencia y generalización, tal como se muestra en la Figura 3.9.

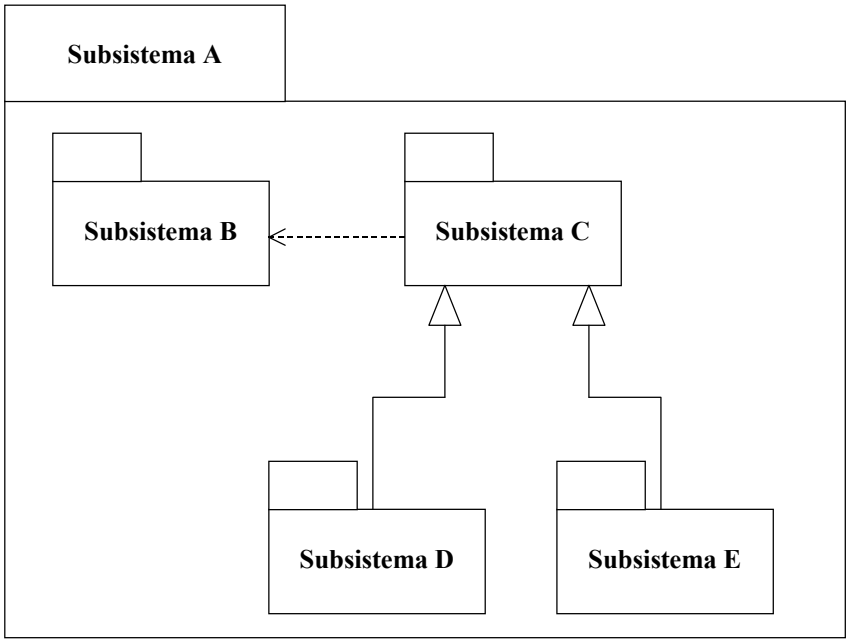
Subsistema A
Subsistema B
Subsistema C
Subsistema D
Subsistema E

Figura 3.9