Diagramas de Clase en UML 1.1

Francisco José García Peñalvo
*Licenciado en Informática. Profesor del Área de Lenguajes y Sistemas Informáticos de la **Universidad de Burgos**.*
fgarcia@.ubu.es

Carlos Pardo Aguilar
*Licenciado en Informática. Profesor del Área de Lenguajes y Sistemas Informáticos de la **Universidad de Burgos***
cpardo@.ubu.es

El diagrama de clases es una técnica central y ampliamente difundida en los distintos métodos orientados a objeto. Cada método incluye sus propias variantes a esta técnica, pero en el presente artículo nos vamos a centrar en la visión que se encuentra implementada dentro del lenguaje estándar de modelado UML 1.1

Un lenguaje de modelado debe ser capaz de ofrecer los mecanismos necesarios para capturar y modelar la abstracción de un sistema desde diferentes puntos de vista. Estos puntos de vista deben dar lugar a diferentes diagramas que recojan tanto la definición estática del sistema, como la componente de comportamiento dinámico del mismo.

Para el modelado de la parte estática de un sistema, UML 1.1 [1], [2] cuenta con los diagramas de estructura, que fueron introducidos en [3]. En concreto, los diagramas de estructuras representan las abstracciones identificadas en forma de clases y objetos, mostrando su estructura interna, así como sus interrelaciones. Existen dos tipos de diagramas de estructura: *los diagramas de clase* y *los diagramas de objetos*.

Los diagramas de clase describen los tipos de objetos de un sistema, así como los distintos tipos de relaciones que pueden existir entre ellos. Los diagramas de clase se convierten así en la técnica más potente para el modelado conceptual de un sistema software, la cual suele recoger los conceptos clave del modelo de objetos subyacente al método orientado a objetos que la incorpora, en este caso UML 1.1.

Por su parte, los diagramas estáticos de objetos representan una instantánea del estado del sistema en un momento dado, esto es, cada diagrama de objetos es una instancia del diagrama de clase, que representa uno de los infinitos escenarios a los que puede dar origen un diagrama de clase.

Una vez introducidos los diagramas de estructura, nos vamos a centrar en los aspectos esenciales de un diagrama de clase, por ser éste el diagrama más importante y representativo del modelado estático de sistemas software.

1. Utilidad de un diagrama de clase

El propósito de un diagrama de clase es describir las clases que conforman el modelo de un determinado sistema. Dado el carácter de refinamiento iterativo que caracteriza un desarrollo orientado a objetos, el diagrama de clase va a ser creado y refinado durante las fases de análisis y diseño, estando presente como guía en la implementación del sistema.

Se puede decir que existen tres perspectivas diferentes desde las cuales se pueden utilizar los diagramas de clase:

- ***Conceptual:*** El diagrama de clase representa los conceptos en el dominio del problema que se está estudiando. Este modelo debe crearse con la mayor independencia posible de la implementación final del sistema.
- ***Especificación:*** El diagrama de clase refleja las interfaces de las clases, pero no su implementación. Aquí las clases aparecen más cercanas a los tipos de datos, ya que un tipo representa una interfaz que puede tener muchas implementaciones diferentes.
- ***Implementación:*** Esta vista representa las clases tal cual aparecen en el entorno de implementación.

Los diagramas de clase de UML pueden utilizarse en cualquiera de las tres perspectivas presentadas, como se muestra en la **Figura B**.

2. Elementos esenciales de los diagramas de clase

El objetivo del presente apartado es dar un repaso a los principales elementos de modelado que se encuentran presentes en los diagramas de clase de UML 1.1, para una mayor ampliación de los conceptos que aquí se van a tratar, así como para el estudio de los aspectos más avanzados y complejos de estos diagramas, se recomienda al lector dirigirse a la **UML Notation Guide** que puede encontrarse en [1].

Como se ha indicado anteriormente, en un diagrama de clase aparecen clases relacionadas entre sí, de esta forma, las clases y las principales relaciones semánticas entre ellas pueden ser considerados como los elementos esenciales de estos diagramas.

Clases

Para UML una clase es "*una descripción de un conjunto de objetos que comparten los mismos atributos, operaciones, métodos, relaciones, y semántica*". De esta forma, un diagrama de clase de UML puede describir todos los componentes de una clase de una forma sencilla. Así, el elemento fundamental de los diagramas de clase es el icono que representa una clase.

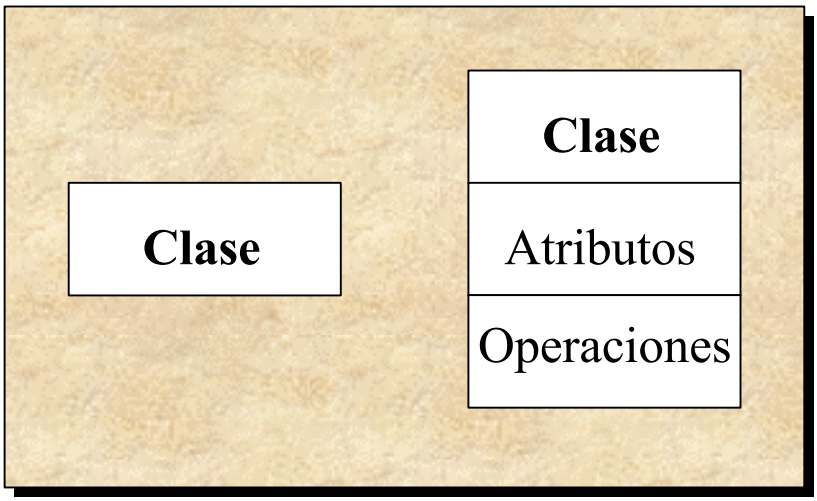

Figura A: Iconos para representar clases

El icono de una clase es un rectángulo dividido en tres secciones, como se puede apreciar en la **Figura A**. La sección superior contiene el nombre de la clase, la sección

intermedia contiene la lista de atributos, y la sección inferior contiene la lista de operaciones de la clase. Tanto la sección de atributos como la sección de operaciones pueden omitirse. Cuando estas secciones aparecen, normalmente no muestran todos los atributos ni todas las operaciones. El objetivo es mostrar sólo aquellos atributos y operaciones que son representativos para un determinado diagrama.

Dependiendo del detalle del diagrama de clase, la notación para un atributo puede indicar su nombre, su tipo, un valor de inicio y su visibilidad, siendo su sintaxis:

visibilidad nombre: tipo = valor

Donde:

- Visibilidad expresa si el atributo es visible para el resto de objetos del diagrama, pudiéndose dar los siguientes casos:
  - \+ *Visibilidad pública: Visible por todos los objetos*
  - \# *Visibilidad protegida: Visible sólo por el objeto y sus descendientes*
  - \- *Visibilidad privada: Visible sólo por el objeto*
- Nombre es el identificador del atributo
- Tipo indica el dominio del atributo
- Valor es un elemento opcional que indica un valor de inicio para el atributo

Al igual que sucede con los atributos, las operaciones de una clase pueden especificarse con diferente nivel de detalle según la siguiente sintaxis:

visibilidad nombre (lista de parámetros) : tipo de retorno

La propiedad de ocultar las secciones de atributos y operaciones de una clase en los diagramas de clase de UML, así como la posibilidad de especificar con un mayor o menor grado de detalle los atributos y las operaciones de una clase, permite utilizar un diagrama de clase de UML desde una perspectiva conceptual, de especificación o de implementación, como se puede apreciar en la **Figura B**.

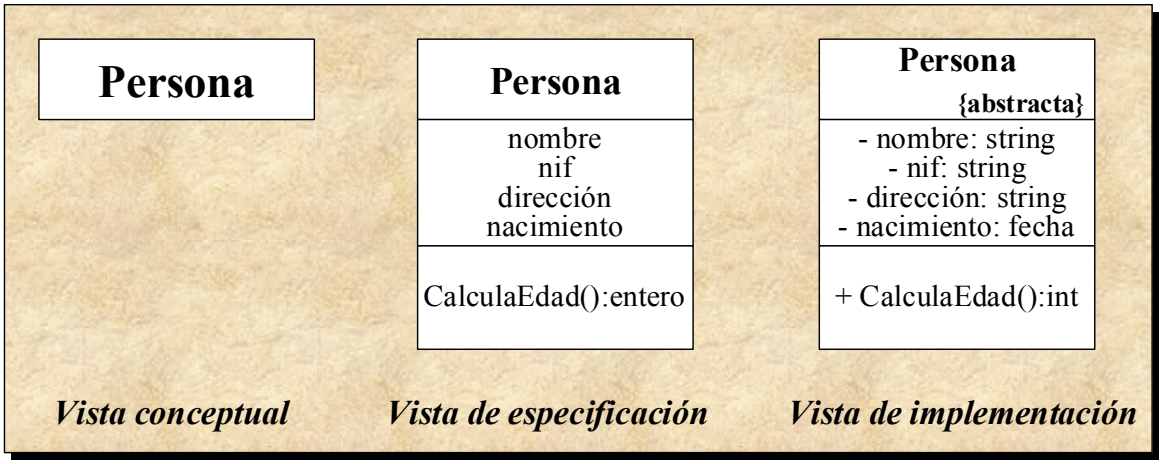

Figura B: Diferentes vistas de una clase

Asociaciones

Las asociaciones representan las relaciones más generales entre clases, es decir, las relaciones con menor contenido semántico. Para UML una asociación va a describir un conjunto de vínculos entre las instancias de las clases.

Las asociaciones pueden ser binarias (*conectan dos* clases) o n-arias (*conectan n clases*), aunque lo más normal en un modelo es utilizar sólo relaciones binarias (*en general, y sin entrar en detalles, se puede afirmar que una relación n-aria puede modelarse mediante un conjunto finito de relaciones binarias*).

La forma de representar las asociaciones binarias en UML es mediante una línea que conecta las dos clases. En general, las asociaciones son bidireccionales, esto es, no tienen un sentido asociado.

Cada asociación tiene dos roles; cada rol marca una dirección en la asociación. Así, en la **Figura C**, la asociación entre **Cliente** y **Pedido** contiene dos roles: uno de **Pedido** a **Cliente;** y otro de **Cliente** a **Pedido**. A cada rol se le puede asociar una etiqueta con su nombre. Si un rol no tiene asociado un nombre se le da el nombre de la clase destino de la asociación, así el rol de **Cliente** a **Pedido,** recibe el nombre de **Pedido**.

Cada rol tiene asociada una multiplicidad que especifica el número de instancias de una clase que pueden estar relacionadas con una única instancia de la clase asociada. La multiplicidad se expresa en UML mediante una cadena asociada a un rol que representa un subconjunto abierto de enteros no negativos. Sintácticamente esto se traduce en una secuencia de intervalos de números enteros separada por comas, donde cada intervalo representa un rango, quizás infinito, de enteros en el formato:

cota inferior .. cota superior

Donde cota inferior y cota superior son valores de literales enteros. Adicionalmente la cota superior se puede representar a través de un asterisco, '*', indicando la inexistencia de una cota superior. Un único '*' indica el rango 0..∞. Un único número indica una multiplicidad exacta, así por ejemplo **2** sería equivalente al rango 2..2. En la práctica, las multiplicidades más utilizadas son **1** (*exactamente una*), ***** (*muchas: 0 a infinito*) y **0..1** (*cero o una*).

Un aspecto importante, que en ocasiones puede ser una fuente de confusión, es cómo interpretar las multiplicidades que aparecen en los extremos de una asociación. En UML la multiplicidad se define con respecto a una instancia de la clase al otro extremo de la asociación, esto es, el indicador de multiplicidad se interpreta asumiendo una sola instancia de la clase al otro extremo de la asociación y después leyendo el mínimo y el máximo de instancias para la clase más cercana al indicador. Según esto, las cardinalidades que aparecen en la **Figura C** se leerían: "*Un **Cliente** puede tener asociados 0 o más **Pedidos**, y un **Pedido** tiene que tener asociado siempre un solo **Cliente***".

Como se ha dicho anteriormente, las asociaciones son por defecto bidireccionales, de forma que cuando se quiera modelar una asociación unidireccional entre clases, debe indicarse de forma explícita el sentido de la asociación mediante una punta de flecha al final de la línea de asociación. Esto es lo que se denomina en UML navegabilidad, y en nuestro ejemplo de la **Figura C** significa que un **Pedido** tiene la responsabilidad de decir a que **Cliente** está asociado, pero un **Cliente** no tiene la responsabilidad de decir que pedidos ha realizado.

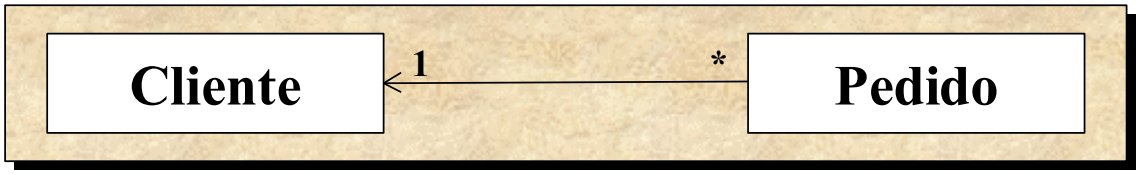

Figura C: Ejemplo de asociación

Agregación y composición

La agregación es una asociación con unas connotaciones semánticas más definidas: la agregación es la relación *parte-de*, que presenta a una entidad como un agregado de partes (*en orientación a objeto, un objeto como agregado de otros objetos*). Como ejemplo ilustrativo de lo que es una agregación podemos recurrir al ya clásico ejemplo de la bicicleta, donde una bicicleta se modela como un agregado de ruedas, sillín, manillar, cuadro y pedales (ver **Figura D**).

La agregación en UML presenta el siguiente matiz, la existencia de las partes agregadas es independiente de la existencia del objeto agregado, esto es, cuando se crea el objeto agregado se irán estableciendo las relaciones con cada una de las partes que lo constituyen a medida que se vayan necesitando. Los objetos que representan las partes del objeto agregado pueden ya existir o crearse para formar parte del objeto agregado, pero cuando se destruye el objeto agregado, los objetos que lo forman no tienen porque ser destruidos, por consiguiente, las partes pueden sobrevivir a la destrucción del objeto agregado.

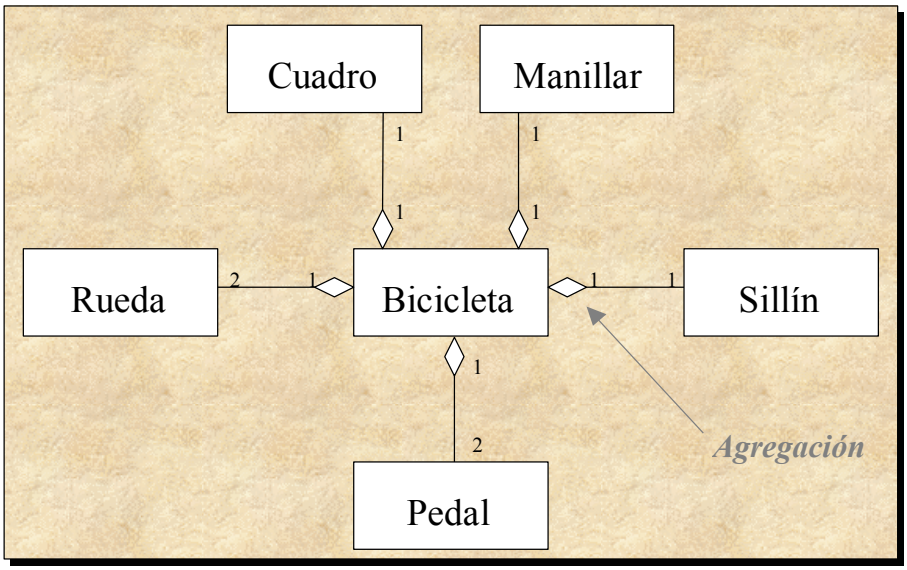

Figura D: Ejemplo de agregación

Volviendo al ejemplo de la bicicleta, cuando se crea una instancia de la clase bicicleta, ésta deberá asociarse con las instancias oportunas de las clases de los objetos que forman el agregado bicicleta. Pero, si la instancia de bicicleta es destruida, las instancias de sus objetos constituyentes pueden seguir existiendo, y convertirse en partes de otros objetos.

En UML la agregación se representa por una asociación en la que el rol del extremo unido a la clase agregada presenta el adorno de un diamante vacío.

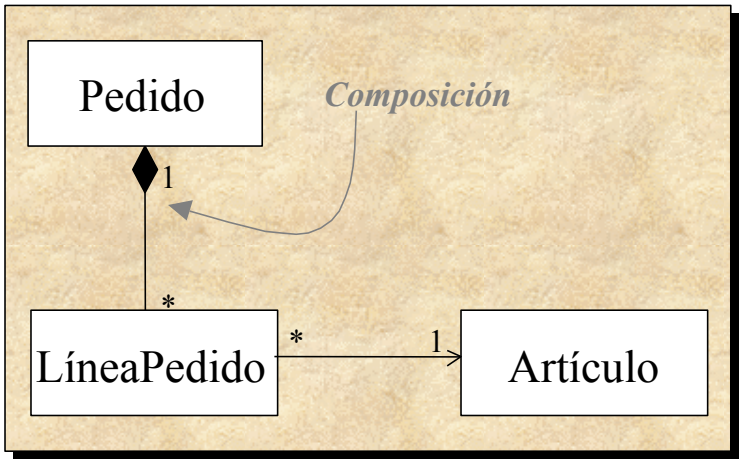

Figura E: Ejemplo de composición

UML presenta una variación mucho más restrictiva de agregación que recibe el nombre de composición. La composición implica que los componentes de un objeto sólo pueden pertenecer a un solo objeto agregado, de forma que cuando el objeto agregado es destruido todas sus partes son destruidas también.

La notación empleada para la composición es la misma que para la agregación con la diferencia que el adorno de composición es un diamante relleno.

En la **Figura E** se presenta un modelo de un pedido. Un pedido se ha modelado como una composición de líneas de pedido, cada una de las cuales está asociada con un producto.

Aunque los conceptos de agregación y composición se han presentando en un contexto de análisis y diseño orientado a objetos, para ilustrar sus diferencias vamos a ver como se implementarían ambas en C++.

Una agregación se implementa en C++ incluyendo en la declaración de la clase agregada punteros a las instancias de las clases que constituyen sus agregados. Esto es lo que denomina Booch agregación por referencia. Un esquema de la implementación de la clase **Bicicleta** puede apreciarse en la **Figura F.**

```
Class Bicicleta {
      Rueda *delantera, *trasera;
      Pedal *izq, *drch ;
      Cuadro *c;
      Manillar *m;
      Sillín *s;
      …
};
```

Figura F: Implementación de una agregación en C++

Por su parte una composición se implementa en C++ incluyendo las declaraciones de los objetos componentes en la declaración de la clase agregada. Esto es lo que denomina Booch agregación por valor. En la **Figura G** se presenta el esqueleto de la clase **Pedido** en C++. Como un **Pedido** esta forma por un número ilimitado de **LíneasPedido**, se tiene un vector de instancias de **LíneaPedido** que será dimensionado en los constructores de la clase **Pedido**, y destruido en el destructor de dicha clase.

```
Class  Pedido {
       LíneaPedido  *líneas;
       …
public:
       Pedido(int nl) {
              líneas = new LíneaPedido [nl];
              …
       }
       …
       ~Pedido() {
              delete [] líneas;
       }
};
```

Figura G: Implementación de una composición en C++

Herencia

La herencia es la típica relación de generalización/especialización entre clases. En UML la herencia se representa mediante una flecha, cuya punta es un triángulo vacío. La flecha que representa a la herencia va orientada desde la subclase a la superclase. Cuando de una superclase se derivan varias subclases existen dos notaciones diferentes, aunque totalmente equivalentes, para su representación. En la primera forma de representar esta situación se muestra una superclase a la que llegan tantas flechas como clases derivadas tiene. En la segunda representación se tiene una única punta de flecha que llega a la superclase, pero a la base del triángulo que hace de punta de flecha llegan tantos caminos como subclases haya.

La **Figura H** muestra un caso en el que los pedidos pueden ser realizados por dos tipos de clientes: una empresa o una persona individual. Según esto, la clase **Cliente** es una generalización de las clases **Empresa** y **PersonaIndividual**, o visto en la otra dirección, **Empresa** y **PersonaIndividual** son especializaciones de **Cliente**.

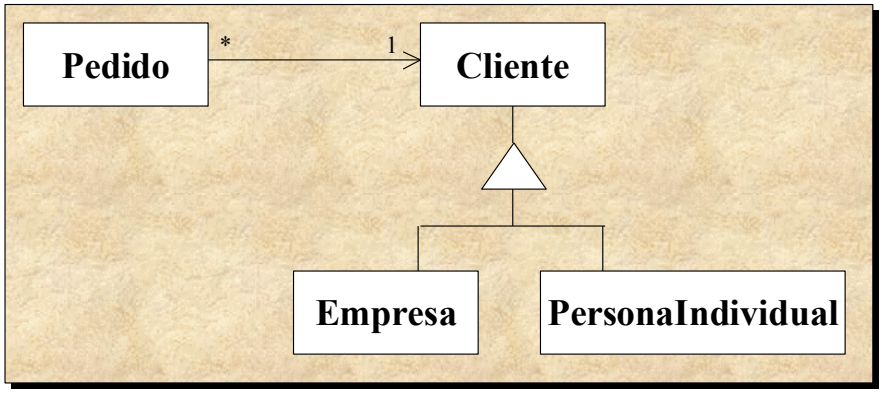

Figura H: Ejemplo de herencia

La herencia tiene diferentes interpretaciones según la perspectiva de modelado que se esté utilizando.

En el nivel conceptual, simplemente expresa que una determinada entidad es un subtipo de otra entidad, por ejemplo **Empresa** es un subtipo de **Cliente** siendo, por tanto, un tipo especial de **Cliente**. La idea central es expresar que todo lo que se dice que es cierto para una clase (*atributos, operaciones y relaciones*) es cierto para sus subclases.

Cuando se está realizando un modelo de especificación, la idea principal se centra en que la interfaz de una subclase debe incluir todos los elementos de la interfaz de su superclase. Otra forma de expresar el objetivo del nivel de especificación en cuanto a la herencia es el principio de capacidad de sustitución, o lo que es lo mismo, que en el lugar donde se espere una instancia de la superclase, pueda aparecer una instancia de cualquiera de sus subclases, y todo siga funcionando correctamente. La instancia de la subclase puede responder a ciertas órdenes de forma diferente a como lo haría una instancia de la superclase (*gracias al polimorfismo*), pero los clientes que esperan la instancia de la superclase no necesitan conocer ni preocuparse por estas diferencias.

Desde el punto de vista de la implementación, la herencia está asociada a la capacidad de los lenguajes de programación para representar este mecanismo de transmisión de estructura y comportamiento desde la superclase a la subclase, esto es, la subclase hereda todos los métodos y atributos de la superclase.

3. Conclusiones

En este artículo se ha presentado una de las técnicas de modelado más difundidas en los métodos de análisis y diseño orientado a objetos, vista desde el prisma de UML 1.1, haciendo un rápido recorrido por lo que son los elementos esenciales de los diagramas de clase, las clases y sus principales relaciones (*asociación, agregación y herencia*). UML 1.1 presenta otros diversos elementos (*estereotipos, relaciones de dependencia, relaciones de refinamiento*...) que pueden aparecer en este tipo de diagramas, pero su semántica e influencia en el modelado de sistemas software son lo suficientemente complejos como para estar fuera de un artículo introductorio como el que aquí se presenta.

Para concluir, y a modo de resumen, citar la facilidad de comprensión de estos diagramas en el modelado de sistemas software, así como su apoyo y presencia desde los modelos conceptuales iniciales a la implementación, como dos de sus principales características.

4. Bibliografía

[1] **Rational Software Corporation et al**. "*UML 1.1 Documentation Set*". http://www.rational.com/uml. 1 September 1997.

[2] **García Peñalvo, Francisco José y Pardo Aguilar, Carlos**. "*UML 1.1. Un lenguaje de modelado estándar para los métodos de ADOO*". RPP, Nº36, pp. 57-61. Enero, 1998.

[3] **García Peñalvo, Francisco José y Pardo Aguilar, Carlos**. "*Introducción al Análisis y Diseño Orientado a Objetos*". RPP, Nº37. Febrero, 1998.