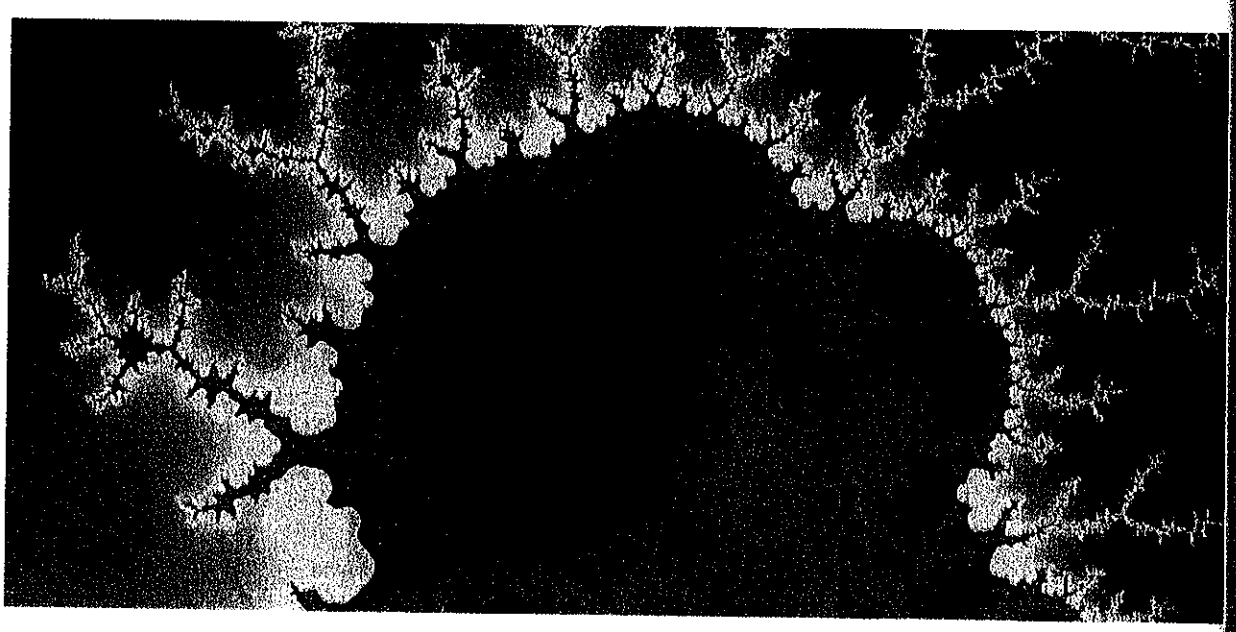
# Parte 1

# Introducción de UML y UP



# 1

# ¿Qué es UML?

## 1.1. Presentación del capítulo

Este capítulo proporciona una breve visión de conjunto de la historia y la estructura de alto nivel de UML. Mencionamos muchos temas que se ampliarán en capítulos posteriores. Los principiantes deberían empezar aprendiendo la historia y los principios de UML. Si tiene experiencia con UML o considera que sabe suficiente historia de UML, puede pasar directamente al apartado correspondiente, a la estructura de UML. Existen tres enfoques principales en esta explicación que se pueden leer en cualquier orden. En este capítulo sabrá más sobre los bloques de construcción de UML, los mecanismos comunes de UML y la arquitectura de UML.

## 1.2. ¿Qué es UML?

El Lenguaje Unificado de Modelado (*Unified Modeling Language*, UML) es un lenguaje de modelado visual para sistemas. Aunque UML está más asociado con modelar sistemas de software orientados a objetos, tiene una aplicación mucho más amplia que esto debido a sus mecanismos incorporados de extensibilidad.

UML se diseñó para incorporar las mejores prácticas en las técnicas de modelado y la ingeniería de software. Como tal, está explícitamente diseñado para implementarse por las herramientas de modelado UML.

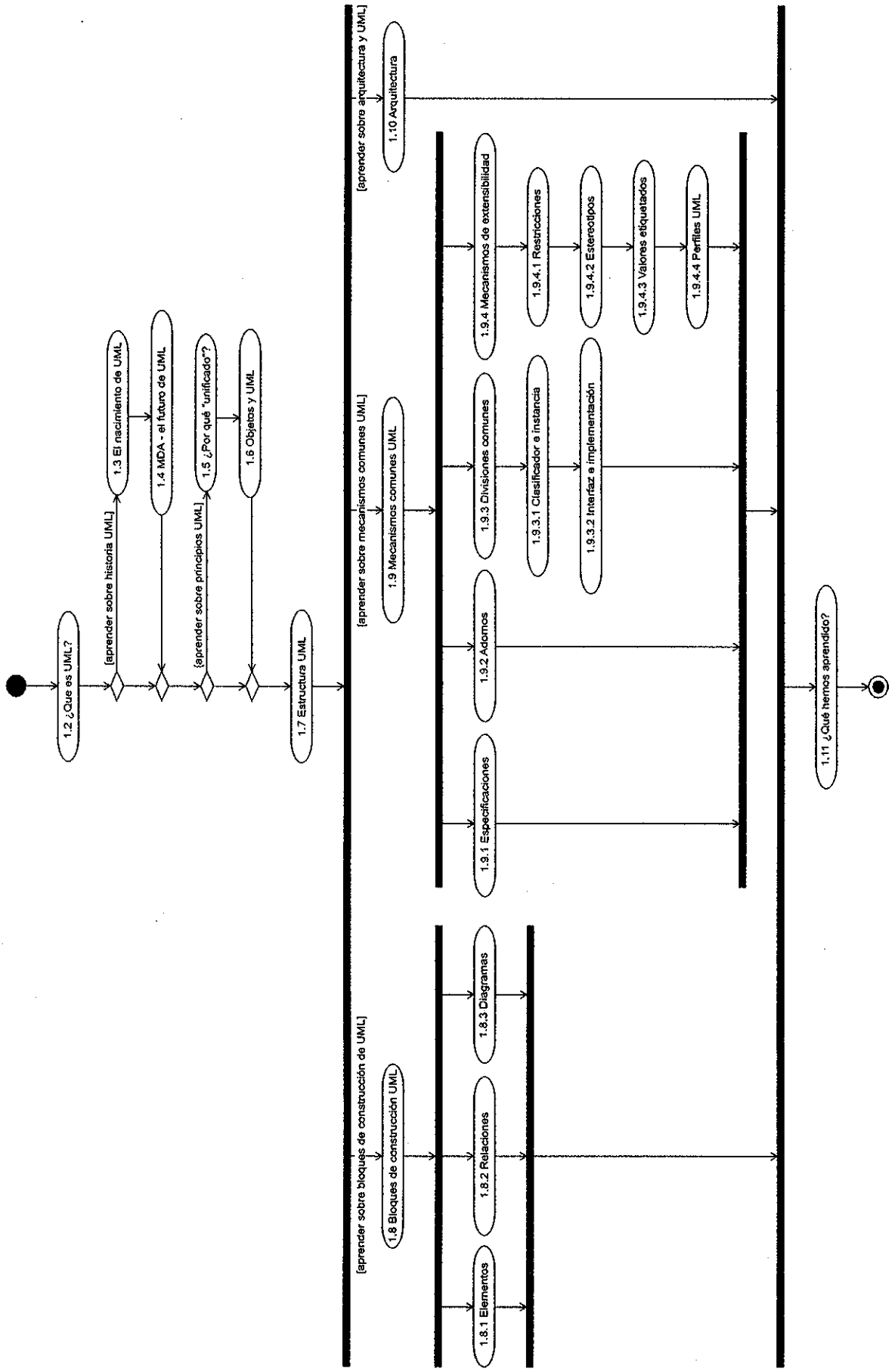

1.2 ¿Que es UML?
[aprender sobre historia UML]
1.3 El nacimiento de UML
1.4 MDA - el futuro de UML
[aprender sobre principios UML]
1.5 ¿Por qué "unificado"?
1.6 Objetos y UML
1.7 Estructura UML
[aprender sobre mecanismos comunes UML]
1.9 Mecanismos comunes UML
1.9.1 Especificaciones
1.9.2 Adornos
1.9.3 Divisiones comunes
1.9.3.1 Clasificador e instancia
1.9.3.2 Interfaz e implementación
1.9.4 Mecanismos de extensibilidad
1.9.4.1 Restricciones
1.9.4.2 Estereotipos
1.9.4.3 Valores etiquetados
1.9.4.4 Perfiles UML
[aprender sobre arquitectura y UML]
1.10 Arquitectura
[aprender sobre bloques de construcción de UML]
1.8 Bloques de construcción UML
1.8.1 Elementos
1.8.2 Relaciones
1.8.3 Diagramas
1.11 ¿Qué hemos aprendido?

**Figura 1.1.**

Los sistemas de software modernos normalmente requieren el soporte de herramientas. Los diagramas UML son legibles por las personas y los ordenadores pueden mostrarlos fácilmente. Es importante apreciar que UML no nos proporciona ningún tipo de metodología de modelado. Naturalmente, algunos aspectos de la metodología se ven implicados por los elementos que componen un modelo UML, pero UML simplemente proporciona una sintaxis visual que podemos utilizar para construir modelos.

El proceso unificado (UP) es una metodología; nos dice los recursos humanos, las actividades y artefactos que necesitamos utilizar, desarrollar o crear para modelar un sistema de software.

UML no está unido a ninguna metodología específica o ciclo de vida y se puede utilizar con todas las metodologías existentes. UP utiliza UML como su sintaxis de modelado visual subyacente y por lo tanto puede pensar de UP como el método preferido para UML, pero UML puede y proporciona el soporte de modelado visual para otros métodos. Para un ejemplo específico de una metodología madura que también utiliza UML como su sintaxis visual, véase el método OPEN (*Object-oriented Process, Environment, and Notation*) en `www.open.org.au`.

El objetivo de UML y UP siempre ha sido dar soporte y encapsular las mejores prácticas en la ingeniería de software basándose en la experiencia de la última década. Para hacer esto, UML y UP unifican intentos anteriores de lenguajes de modelado visual y procesos de ingeniería de software en la mejor de las soluciones.

## 1.3. El nacimiento de UML

Anterior a 1994, el mundo de los métodos orientados a objetos era bastante confuso. Existían varios lenguajes y metodologías de modelado visual competidores entre sí, todas ellos con sus fortalezas y debilidades y con sus seguidores y detractores. En términos de lenguajes de modelado visual (véase la figura 1.2), los líderes evidentes eran Booch (el método Booch) y Rumbaugh (*Object Modeling Technique*, OMT, o Técnica de Modelado de Objetos) que entre ellos tenían la mitad del mercado. En el lado de las metodologías, Jacobson tenía el caso más fuerte y aunque muchos autores proclamaban tener un "método", en realidad lo que tenían era una sintaxis de modelado visual y una colección de aforismos y directrices más o menos útiles.

Hubo un primer intento de unificación en 1994 con el método de fusión de Coleman. Sin embargo, este intento no implicó a los autores originales de los métodos constituyentes (Booch, Jacobson y Rumbaugh) y llegó también bastante tarde al mercado con un libro que explicaba el enfoque. La fusión se vio superada rápidamente por el curso de los acontecimientos cuando en 1994 Booch y Rumbaugh se unieron al Rational Corporation para trabajar en UML. Esto nos preocupó a muchos en aquel momento y proporcionó a Rational cerca de la mitad del mercado de los métodos. Sin embargo, estos miedos han demostrado ser totalmente infundados y UML se ha convertido desde entonces en un estándar de la industria abierto.

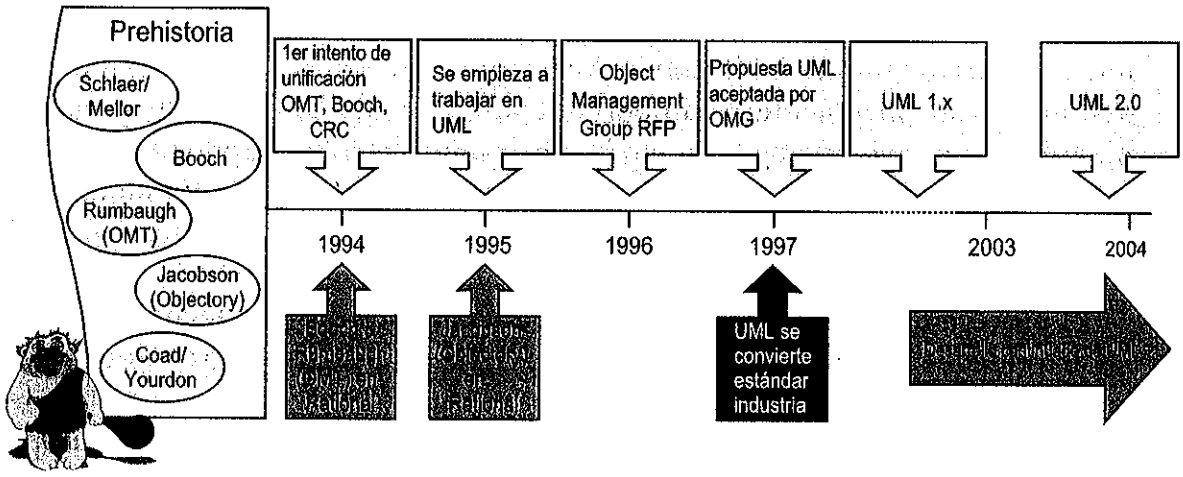



**Figura 1.2.**

En 1996, el Object Management Group (OMG) lanzó una RFP (*request-for-proposal*) para un lenguaje de modelado visual orientado a objetos, y se presentó UML. En 1997, OMG aceptó el UML y así nació el primer lenguaje de modelado visual orientado a objetos estándar de la industria abierto. Desde entonces, todos los métodos competidores han desaparecido y UML sigue siendo el lenguaje de modelado orientado a objetos estándar de la industria.

En el 2000, UML 1.4 presentó una importante ampliación de UML por medio de la incorporación de la semántica de acción. Ésta describe el comportamiento de un conjunto de acciones primitivas que se pueden implementar por lenguajes de acción específicos. La semántica de acción más un lenguaje de acción permitía la especificación detallada de los elementos de comportamiento de modelos UML (como operaciones de clase) directamente en el modelo UML. Éste fue un desarrollo muy significativo ya que hizo que la especificación UML fuera computacionalmente completa y permitió que los modelos UML fueran ejecutables. Para un ejemplo de una implementación UML que tiene un lenguaje de acción compatible con la semántica de acción, consulte xUML de Kennedy Carter (`www.kc.com`).

Puesto que actualizamos este libro en su segunda edición, estamos en el 2005 y la especificación UML 2.0 se ha finalizado. UML es ahora un lenguaje de modelado muy maduro. Hace ya casi siete años desde su primera aparición y ha demostrado su valor en miles de proyectos de desarrollo de software en todo el mundo.

UML 2 presenta numerosa sintaxis visual nueva. Parte de ésta reemplaza y aclara la sintaxis 1.x existente y parte de ella es completamente nueva y representa nueva semántica añadida al lenguaje. UML siempre ha proporcionado muchas opciones acerca de cómo se puede mostrar un elemento de modelo particular y no todo esto se soportará en todas las herramientas de modelado. Trataremos de utilizar las variantes sintácticas más comunes de forma coherente durante todo este libro y resaltar otras variantes allí donde creamos que pueden servir para otra finalidad en situaciones comunes de modelado. Algunas opciones sintácticas son bastante especializadas, y por tanto solamente las mencionamos de pasada si llega el caso. Aunque UML 2 realiza muchos cambios sintácticos a UML comparado con UML 1.x, la buena noticia es que los principios fundamentales permanecen más o

menos igual. Los modeladores que estén acostumbrados a utilizar versiones anteriores de UML deberían experimentar una transición fácil hacia UML 2. De hecho, los cambios más importantes incorporados en UML 2 se han realizado en el metamodelo de UML y no los encontrarán directamente la mayoría de modeladores. El metamodelo de UML es un modelo del lenguaje UML que se expresa en un subconjunto de UML. Define la sintaxis y semántica de todos los elementos de modelado UML que encontrará en este libro. Estos cambios al metamodelo de UML han estado relacionados con la mejora de la precisión y la coherencia de la especificación UML.

En uno de sus libros, Grady Booch dijo "si tiene una idea, entonces es mía". En cierto sentido, esto resume la filosofía de UML; toma lo mejor de todo lo que ha aparecido antes, lo integra y construye sobre ello. Esto es una reutilización en su sentido más amplio, y UML incorpora muchas de las mejores ideas de los métodos "prehistóricos" a la vez que rechaza algunos de sus extremos más idiosincrásicos.

## 1.4. MDA, el futuro de UML

El futuro de UML se ha definido por una reciente iniciativa de OMG denominada *Model Driven Arquitecture* (Arquitectura dirigida por el modelo o MDA). Aunque éste no es un libro sobre MDA, proporcionaremos una breve visión de conjunto sobre MDA en este apartado. Puede encontrar más información en el sitio Web de MDA de OMG (`www.omg.org/mda`) y en *MDA Explained* [Kleppe 1] y *Model Driven Architecture* [Frankel 1].

MDA define una visión de cómo se puede desarrollar software basándose en modelos. La esencia de esta visión es que los modelos dirigen la producción de la arquitectura de software ejecutable. Hasta cierto punto, esto sucede hoy pero MDA asigna un grado de automatización de este proceso que está apenas conseguida.

En MDA, el software se produce por medio de una serie de transformaciones de modelo ayudadas por una herramienta de modelado MDA. Un modelo abstracto independiente del ordenador (CMI) se utiliza como una base para un modelo independiente de la plataforma (PIM). El PIM se transforma en un modelo específico de la plataforma (PSM) que se transforma en código.

La noción MDA del modelo es bastante general y el código se considera un tipo muy concreto de modelo. La figura 1.3 ilustra la cadena de transformación del modelo MDA.

El CIM es un modelo en su nivel más alto de abstracción que captura requisitos clave del sistema y el vocabulario del dominio del problema en una forma que es independiente de los ordenadores. Realmente es un modelo de esa parte del negocio que va a automatizar. La creación de este modelo es opcional, y si decide crearla, lo utiliza como una base para generar el PIM.

El PIM es un modelo que expresa la semántica del negocio del sistema de software independientemente de cualquier plataforma subyacente (como EJB, .NET, etc.). El PIM se encuentra al mismo nivel de abstracción que el modelo de análisis

del que hablaremos más adelante en este libro, pero es más completo. Esto es necesariamente así, ya que tiene que proporcionar una base suficientemente completa para transformarse en un PSM desde el que se pueda generar código. Un punto que merece la pena destacar es que el término "independiente de la plataforma" significa muy poco a menos que defina la plataforma o plataformas de la que desea ser independiente. Diferentes herramientas MDA soportan diferentes niveles de independencia de plataforma.

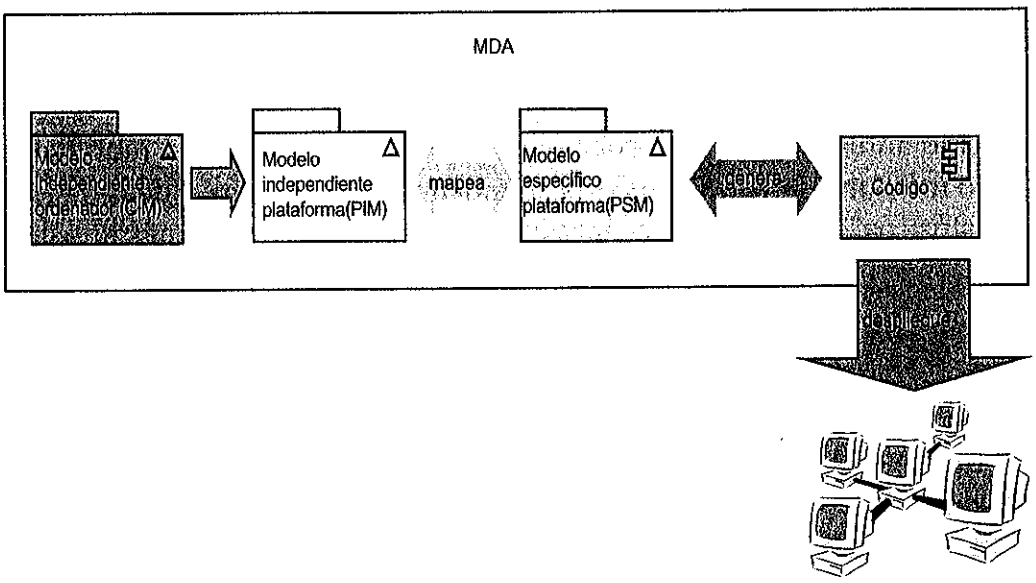



**Figura 1.3.**

El PIM está adornado con información específica de la plataforma para crear el PSM. El código fuente se genera desde el PSM contra la plataforma destino.

En principio, el cien por cien del código fuente y artefactos secundarios como documentación, archivos de creación, descriptores de despliegue se pueden generar desde un PSM suficientemente completo. Si esto sucede, el modelo UML se debe de hacer computacionalmente completo; es decir, la semántica de todas las operaciones se deben especificar en un lenguaje de acción.

Como mencionamos anteriormente, algunas herramientas MDA ya proporcionan un lenguaje de acción. Por ejemplo, la herramienta iUML de Kennedy Carter (`www.kc.com`) proporciona Action Specification Language (ASL) que se ajusta a la semántica de acción de UML 2. Este lenguaje de acción se encuentra en un nivel de abstracción más alto que lenguajes como Java y C++ y lo puede utilizar para crear modelos UML computacionalmente completos.

Otras herramientas MDA como ArcStyler (`www.io-software.com`) permiten la generación de entre el 70 y el 90 por ciento del código y otros artefactos, pero los cuerpos de operación todavía se tienen que completar en el lenguaje destino (por ejemplo Java).

En la visión MDA, el código fuente, como el código Java y C#, es el "código máquina" resultante de la compilación de los modelos UML. Este código se genera según se necesite directamente desde el PSM. Como tal, el código tiene intrínsecamente un valor menor en el desarrollo MDA que los modelos UML. MDA cambia

los modelos UML desde su rol actual como precursores para código fuente creado manualmente en mecanismos primarios de producción de código.

En el momento de escribir estas líneas, cada vez más vendedores de herramientas de modelado están añadiendo posibilidades MDA a sus productos. Debería comprobar el sitio Web MDA de OMG para los detalles más actualizados. Existen también algunas iniciativas MDA de código abierto bastante prometedoras, por ejemplo, Eclipse Modeling Framework (`www.eclipse.org/emf`) y AndroMDA (`www.andromda.org`). En este apartado nos hemos limitado a "la idea general" de MDA. Existe mucho más en la especificación de MDA que lo que hemos mencionado aquí y le animamos a que compruebe las referencias que hemos mencionado al principio de este apartado para más información.

## 1.5. ¿Por qué "unificado"?

La unificación de UML no solamente es histórica en su ámbito de aplicación; UML trata de estar unificada en diferentes dominios.

- **Ciclo de vida de desarrollo:** UML proporciona sintaxis visual para modelado directamente por medio del ciclo de vida del desarrollo de software desde los requisitos a la implementación.
- **Dominios de aplicación:** UML se ha utilizado para modelar de todo, desde sistemas incorporados en tiempo real a sistemas de soporte a la toma de decisión.
- **Lenguajes y plataformas de implementación:** UML es neutro tanto en lenguaje como en plataforma. Dispone de un excelente soporte para lenguajes orientados a objetos (Smalltalk, Java, C#, etc.), pero también es eficaz para lenguajes orientados a objetos híbridos como C++ y lenguajes basados en objetos como Visual Basic. Se ha utilizado incluso para modelar para lenguajes no orientados a objetos como C.
- **Procesos de desarrollo:** Aunque UP y sus variantes son probablemente los procesos de desarrollo preferidos para sistemas orientados a objetos, UML puede soportar muchos otros procesos de ingeniería de software.
- **Sus propios conceptos internos:** UML trata de de ser coherente y uniforme en su aplicación de un pequeño grupo de conceptos internos. No siempre tiene éxito, pero sigue siendo una gran mejora sobre intentos anteriores.

## 1.6. Objetos y UML

La premisa básica de UML es que podemos modelar software y otros sistemas como colecciones de objetos que interactúan. Esto encaja muy bien con sistemas de

software y lenguajes orientados a objetos, pero también funciona muy bien para procesos de negocios y otras aplicaciones.

Existen dos aspectos en un modelo UML:

- **Estructura estática:** Esto describe qué tipo de objetos son importantes para modelar el sistema y cómo se relacionan.
- **Comportamiento dinámico:** Esto describe los ciclos de vida de estos objetos y cómo interactúan entre sí para entregar la funcionalidad de sistema requerida.

Estos dos aspectos del modelo UML van íntimamente unidos, y uno no está del todo completo sin el otro.

Examinaremos los objetos (y las clases) en detalle en el capítulo 7. Hasta entonces, simplemente piense en un objeto como un núcleo unido de datos y comportamiento. En otras palabras, los objetos contienen información y pueden realizar funciones.

## 1.7. Estructura de UML

Puede empezar a entender por qué UML funciona como un lenguaje visual al observar su estructura. Esto se ilustra en la figura 1.4 (como verá más adelante, se trata de un diagrama UML válido). Esta estructura consta de:

- **Bloques de construcción:** Éstos son los elementos básicos de modelado UML, relaciones y diagramas.
- **Mecanismos comunes:** Formas UML comunes de conseguir objetivos específicos.
- **Arquitectura:** La visión UML de la arquitectura del sistema.

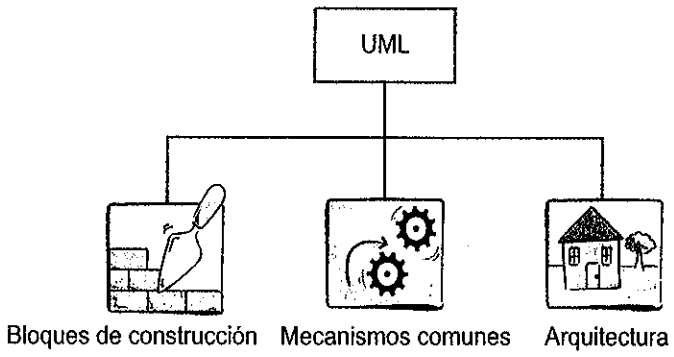



**Figura 1.4.**

Entender la estructura de UML nos proporciona un principio organizativo de utilidad para el resto de información presentada en este libro. También destaca que

UML es un sistema diseñado. De hecho, UML se ha modelado y diseñado utilizando UML. Este diseño es el metamodelo UML.

## 1.8. Bloques de construcción de UML

Según *The Unified Modeling Language User Guide* [Booch 2], UML se compone de tres bloques de construcción (véase la figura 1.5).

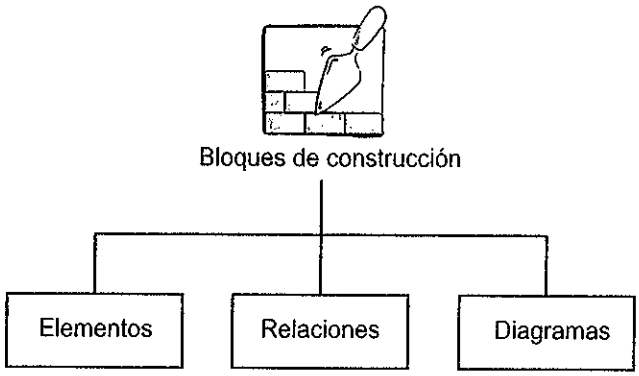



**Figura 1.5.**

- Elementos, que son los propios elementos de modelado.
- Relaciones, que unen a los elementos entre sí. Las relaciones especifican cómo dos o más elementos se relacionan semánticamente.
- Diagramas, que son vistas de los modelos UML. Muestran colecciones de elementos que "cuentan una historia" sobre el sistema de software y son nuestra forma de visualizar qué hará el sistema (diagramas a nivel de análisis) o cómo lo hará (diagramas a nivel de diseño).

Examinaremos los diferentes tipos de bloques de construcción con más detalle en los siguientes apartados.

### 1.8.1. Elementos

Los elementos UML se pueden dividir en:

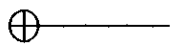
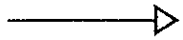
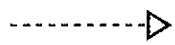
- **Elementos estructurales:** Los nombres de un modelo UML, como una clase, interfaz, colaboración, caso de uso, clase activa, componente, nodo.
- **Elementos de comportamiento:** Los verbos de un modelo UML, como interacciones, actividades, máquinas de estado.
- **Elementos de agrupación:** El paquete, que se utiliza para agrupar elementos de modelado semánticamente relacionados en unidades cohesivas.
- **Elementos de anotación:** La nota, que se puede anexar al modelo para capturar información ad hoc.

Examinaremos estos elementos y cómo se aplican en el modelado UML desde la parte 2 en adelante.

## 1.8.2. Relaciones

Las relaciones le permiten mostrar en un modelo cómo dos o más elementos se relacionan entre sí. Pensar en familias, y las relaciones entre todas las personas en una familia, le proporciona una idea bastante buena del rol que las relaciones desempeñan en modelos UML; le permiten capturar conexiones significativas (semántica) entre elementos. Por ejemplo, las relaciones UML que se aplican a los elementos estructurales y de agrupación en un modelo se resumen en la tabla 1.1.

**Tabla 1.1.**

| Tipo de relación | Sintaxis UML origen destino | Semántica breve |
|---|---|---|
| Dependencia |  | El elemento origen depende del elemento destino y se puede ver afectado por cambios en éste. |
| Asociación |  | La descripción de un conjunto de vínculos entre objetos. |
| Agregación |  | El elemento destino es una parte del elemento origen. |
| Composición |  | Una forma de agregación fuerte (más restringida). |
| Contención |  | El elemento origen contiene el elemento destino. |
| Generalización |  | El elemento origen es una especialización del elemento destino más general y se puede sustituir por éste. |
| Realización |  | El elemento origen garantiza llevar a cabo el contrato especificado por el elemento destino. |

Entender la semántica exacta de los tipos diferentes de relación es una parte muy importante del modelado UML, pero dejaremos la explicación detallada de esta semántica hasta más adelante en el libro.

## 1.8.3. Diagramas

En todas las herramientas de modelado UML, cuando crea un nuevo elemento o una nueva relación, se añade al modelo. El modelo es el repositorio de todos los

elementos y relaciones que ha creado para ayudar a describir el comportamiento requerido del sistema de software que está tratando de diseñar.

Los diagramas son ventanas o vistas de un modelo. El diagrama no es el modelo en sí. Ésta es una distinción muy importante, como elemento o relación se pueden eliminar de un diagrama, o incluso de todos los diagramas, pero puede seguir existiendo en el modelo. De hecho, permanecerá en el modelo hasta que se elimine explícitamente de él. Un error común de los modeladores principiantes de UML es eliminar elementos de los diagramas pero dejarlos en el modelo.

Existen trece tipos diferentes de diagramas UML y estos se listan en la figura 1.6. En la figura, cada cuadro representa un tipo de diagrama. Cuando el texto en el cuadro está en cursiva, representa una categoría abstracta de tipos de diagrama. Por lo tanto, por ejemplo, existen seis tipos diferentes de `DiagramaEstructura`. El texto normal indica un diagrama concreto que puede crear. Los cuadros sombreados indican tipos de diagrama concretos que son nuevos en UML 2.

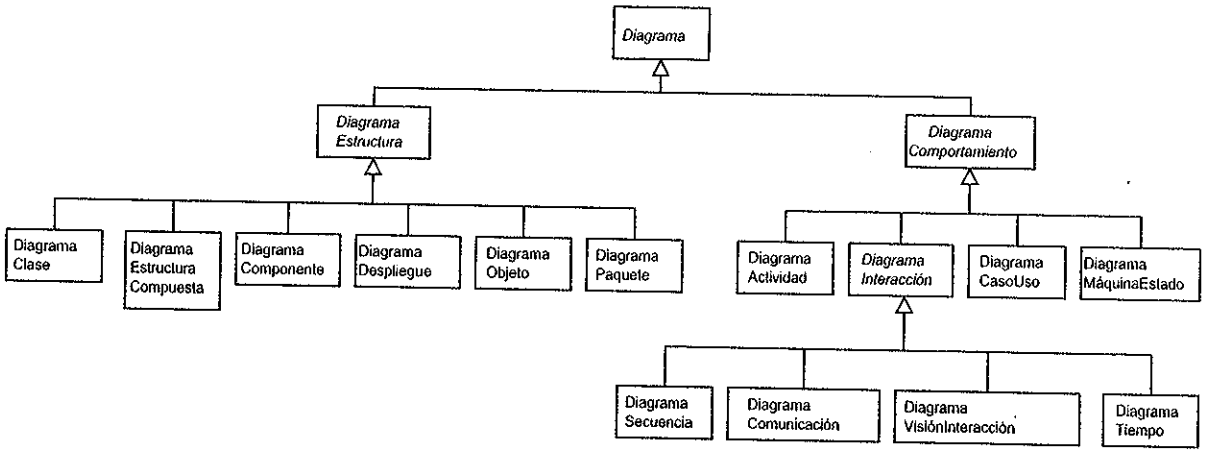
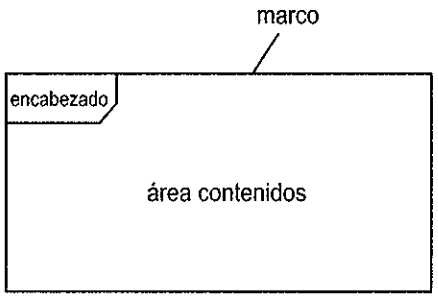



**Figura 1.6.**

Podemos dividir este conjunto de diagramas en los que modelan la estructura estática del sistema (el modelo estático) y los que modelan la estructura dinámica del sistema (el modelo dinámico). El modelo estático captura los elementos y las relaciones estructurales entre los elementos; el modelo dinámico captura cómo los elementos interactúan para generar el comportamiento requerido del sistema de software. Examinaremos el modelo estático y dinámico a partir de la parte 2 en adelante. No existe un orden específico en el que se creen los diagramas UML, aunque normalmente empieza con un diagrama de caso de uso para definir el ámbito de aplicación del sistema. De hecho, normalmente trabaja en varios diagramas en paralelo, refinando cada uno de ellos a medida que descubre cada vez más información y detalle sobre el sistema de software que está diseñando. De esta forma, los diagramas son tanto una vista del modelo y el mecanismo principal para incorporar información en el modelo.

UML 2 introduce una nueva sintaxis para diagramas que se ilustra en la figura 1.7. Todo diagrama puede tener un marco, un área de encabezado y un área de contenidos. El área de encabezado es un pentágono irregular que contiene el tipo de diagrama (opcional), nombre y parámetros (opcional).




sintaxis encabezado: <tipo><nombre><parametros>
Nota: <tipo> y <parametros> son opcionales

**Figura 1.7.**

El `<tipo>` especifica qué tipo de diagrama es y normalmente debería ser uno de los tipos concretos de diagrama listados en la figura 1.6. La especificación UML indica que `<tipo>` se puede abreviar pero no proporciona una lista de abreviaturas estándar. Rara vez necesita especificar `<tipo>` explícitamente porque está claro por la sintaxis visual.

El `<nombre>` debería describir la semántica del diagrama y los `<parámetros>` proporcionan información necesaria por los elementos de modelo en el diagrama. Verá ejemplos de utilizar `<parámetros>` más adelante en el libro.

Opcionalmente, un diagrama puede tener un marco implícito. Aquí es donde el marco está implícito por un área de diagrama en la herramienta de modelado. Puede ver un ejemplo en la figura 1.8.

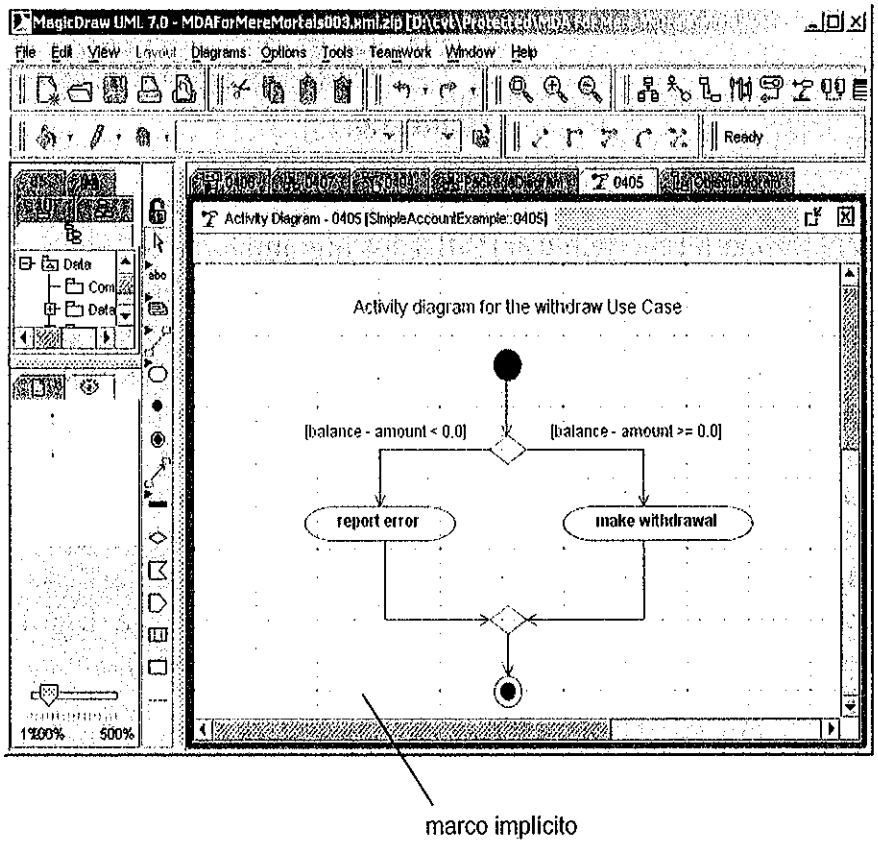



**Figura 1.8.**

## 1.9. Mecanismos comunes de UML

UML tiene cuatro mecanismos comunes que se aplican de forma coherente en todo el lenguaje. Éstos describen cuatro estrategias para acercarse al modelado de objetos, que se aplican de forma repetida en diferentes contextos en todo UML. Una vez más, vemos que UML tiene una estructura sencilla y elegante (véase la figura 1.9).

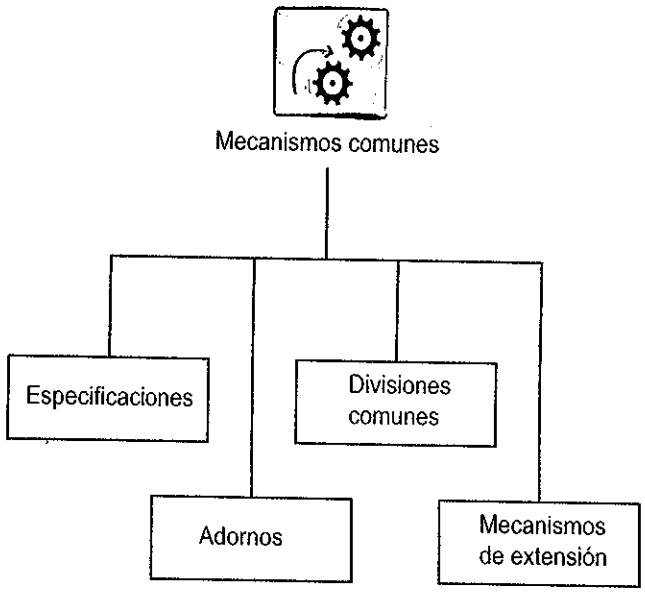



**Figura 1.9.**

### 1.9.1. Especificaciones

Los modelos UML tienen al menos dos dimensiones: una dimensión gráfica que le permite visualizar el modelo utilizando diagramas e iconos y una dimensión textual que consta de las especificaciones de los diferentes elementos de modelado. Las especificaciones son descripciones textuales de la semántica de un elemento.

Por ejemplo, podemos representar visualmente una clase, como la clase CuentaBancaria, como un cuadro con varios compartimentos (véase la figura 1.10), pero esta representación no nos dice realmente nada sobre la semántica de negocio de esa clase. La semántica detrás de los elementos de modelado se captura en su especificación, y sin estas especificaciones solamente puede averiguar qué representa un elemento de modelado. El conjunto de especificaciones es lo "más importante" del modelo y forma el plano posterior semántico que alberga el modelo y le da significado. Los diferentes diagramas son simplemente vistas o proyecciones visuales de ese plano posterior. Este plano posterior semántico se mantiene normalmente utilizando una herramienta de modelado UML que proporciona formas para entrar, ver y modificar especificaciones para cada elemento de modelado.

UML permite numerosa flexibilidad en las construcciones de modelos. En particular, los modelos pueden ser/estar:

- **Elididos:** Algunos elementos están presentes en el plano posterior, pero ocultos en cualquier diagrama para simplificar la vista.
- **Incompletos:** Algunos elementos del modelo pueden faltar por completo.
- **Incoherentes:** El modelo puede contener contradicciones.

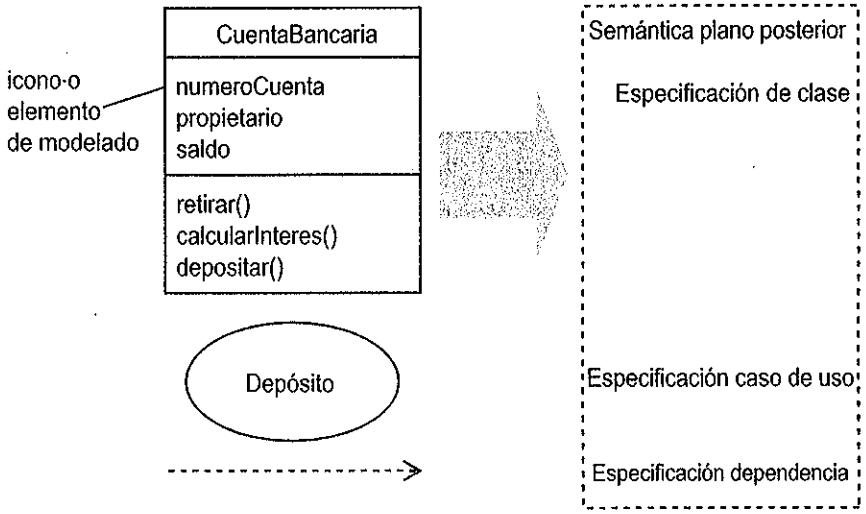



**Figura 1.10.**

El hecho de que las limitaciones de integridad y coherencia estén relajadas es importante, ya que verá que los modelos evolucionan con el tiempo y sufren muchos cambios. Sin embargo, el giro es siempre hacia modelos coherentes que están lo suficientemente completos para permitir la construcción de un sistema de software.

Es una práctica común en el modelado UML empezar con un modelo gráfico grande, que le permite visualizar el sistema, y luego añadir cada vez más semántica en el plano posterior a medida que evoluciona el modelo. Sin embargo, para que un modelo se considere completo o de utilidad, la semántica del modelo debe estar presente en el plano posterior. Si no es así, no tiene modelo, simplemente una colección de cuadros carentes de sentido, conectados por líneas. De hecho, un error bastante común entre los principiantes es lo que se denomina "muerto por los diagramas": el modelo tiene un exceso de diagramas pero no está especificado.

## 1.9.2. Adornos

Una buena característica de UML es que todo elemento de modelado tiene un símbolo sencillo al que añade un número de adornos para hacer visibles aspectos de la especificación del elemento. Utilizando este mecanismo puede adaptar la cantidad de información visible en un diagrama a sus necesidades específicas.

Puede empezar por construir un diagrama de alto nivel al utilizar los símbolos básicos con quizá uno o dos adornos. Luego puede mejorar el diagrama con el tiempo al añadir cada vez más adornos hasta que el diagrama está lo suficientemente detallado para sus fines.

Es importante recordar que cualquier diagrama UML es solamente una vista del modelo, y por lo tanto solamente debería mostrar aquellos adornos que resalten características importantes del modelo y eso aumenta la claridad global y legibilidad del diagrama. No hay necesidad de mostrarlo todo en un diagrama, es más importante que el diagrama sea claro, que ilustre exactamente los puntos que ha indicado y que sea fácil de leer.

La figura 1.11 muestra que el icono mínimo para una clase es un cuadro con el nombre de la clase en él. Sin embargo, puede presentar varias características del modelo que subyace como adornos para ampliar esta vista mínima. El texto en gris indica adornos opcionales posibles.

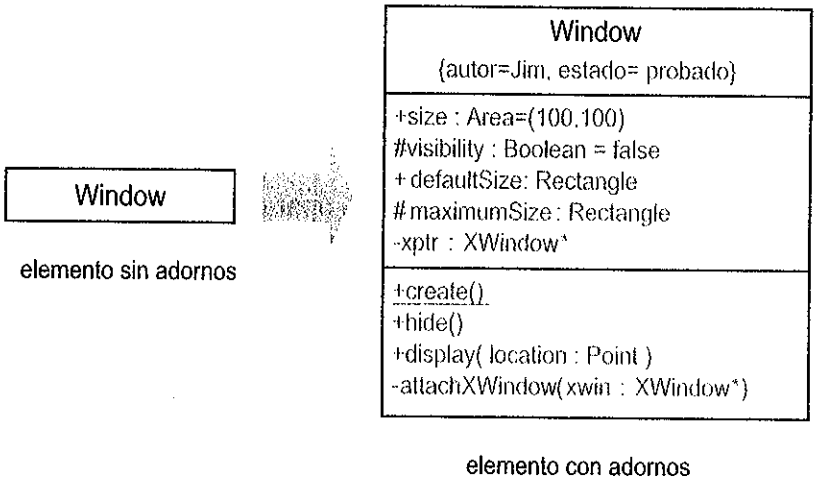



**Figura 1.11.**

## 1.9.3. Divisiones comunes

Las divisiones comunes describen formas particulares de pensar sobre el mundo. Existen dos divisiones comunes en UML: clasificador/instancia e interfaz/implementación.

### 1.9.3.1. Clasificador e instancia

UML considera que podemos tener la noción abstracta de un tipo de elemento (como una cuenta corriente) y luego instancias específicas, concretas de esa abstracción (como "mi cuenta" o "tu cuenta"). La noción abstracta de un tipo de elemento es un clasificador, y los elementos específicos, concretos, las instancias. Éste es un concepto muy importante que es fácil de entender. Los clasificadores e instancias nos rodean. Simplemente piense en este libro UML; podríamos decir que la idea abstracta del libro es "UML 2 y el proceso unificado" y existen muchas instancias de este libro, como la que está leyendo ahora mismo. Veremos que esta noción de clasificador/instancia es un concepto clave que impregna UML.

En UML una instancia normalmente tiene el mismo icono que el clasificador correspondiente, pero para las instancias, el nombre del icono está subrayado. Al principio, ésta puede ser una distinción visual difícil de entender.

UML 2 proporciona una rica taxonomía de treinta y tres clasificadores. Algunos de los clasificadores más comunes se listan en la tabla 1.2. Veremos todos estos (y otros) en detalle en los siguientes apartados.

**Tabla 1.2.**

| Clasificador | Semántica |
|---|---|
| Actor | Un rol desempeñado por un usuario fuera del sistema para el que el sistema proporciona cierto valor. |
| Clase | Una descripción de un conjunto de objetos que comparten las mismas características. |
| Componente | Una parte modular de un sistema que encapsula sus contenidos. |
| Interfaz | Una colección de operaciones que se utilizan para especificar un servicio ofrecido por una clase o componente. |
| Nodo | Un elemento físico, en tiempo de ejecución que representa un recurso computacional, por ejemplo, un PC. |
| Señal | Un mensaje asíncrono pasado entre objetos. |
| Caso de uso | Una descripción de una secuencia de acciones que realiza un sistema para proporcionar valor a un usuario. |

### 1.9.3.2. Interfaz e implementación

El principio aquí es separar qué hace algo (su interfaz) de cómo lo hace (su implementación). Por ejemplo, cuando conduce un coche interactúa con una interfaz muy sencilla y bien definida. Esta interfaz está implementada de diferentes formas por muchos coches físicos diferentes.

Una interfaz define un contrato (que tiene mucho en común con un contrato legal) que garantizan seguir implementaciones específicas. Esta separación de lo que algo promete hacer a partir de la implementación actual de esta promesa es un concepto UML importante. Trataremos esto en detalle en el capítulo 17.

Ejemplos concretos de interfaces e implementaciones están por todas partes. Por ejemplo, los botones de la parte delante de un reproductor de vídeo proporcionan (relativamente) una sencilla interfaz de lo que en realidad es un mecanismo muy complejo. La interfaz nos protege de esta complejidad al ocultárnosla.

## 1.9.4. Mecanismos de extensibilidad

Los diseñadores de UML se dieron cuenta que no era posible diseñar un lenguaje de modelado completamente universal que satisficiera las necesidades presentes y

futuras de todo el mundo, por lo que UML incorpora tres mecanismos sencillos de extensibilidad que se resumen en la tabla 1.3.

**Tabla 1.3.**

| Mecanismos de extensibilidad de UML | |
|---|---|
| Restricciones | Éstas amplían la semántica de un elemento al permitirnos añadir nuevas reglas. |
| Estereotipos | Éstos nos permiten definir un nuevo elemento de modelado UML basándose en uno existente. Nosotros mismos definimos la semántica del estereotipo. Los estereotipos añaden nuevos elementos al metamodelo UML. |
| Valores etiquetados | Proporcionan una forma de ampliar la especificación de un elemento al permitirnos añadir nueva información en él. |

Examinaremos estos mecanismos de extensibilidad en más detalle en los siguientes tres apartados.

### 1.9.4.1. Restricciones

Una restricción es simplemente una cadena de texto entre llaves ({}) que especifica cierta condición o regla sobre el elemento de modelado que se debe mantener como verdadero. Es decir, restringe cierta característica del elemento en cierto sentido. Encontrará ejemplos de esto en todo el libro.

UML define un lenguaje de restricciones denominado OCL (*Object Constraint Language* o Lenguaje de Restricción de Objetos) como una extensión estándar. Proporcionamos una introducción a OCL en el capítulo 25.

### 1.9.4.2. Estereotipos

*The UML Reference Manual* [Rumbaugh 1] indica "un estereotipo representa una variación de un elemento de modelo existente con la misma forma (como atributos y relaciones) pero con un propósito modificado".

Los estereotipos le permiten introducir nuevos elementos de modelado basándose en elementos existentes. Puede hacer esto al anexar el nombre del estereotipo entre <<. . .>> al nuevo elemento. Todo elemento de modelo puede tener cero a muchos estereotipos.

Cada estereotipo puede definir un conjunto de valores etiquetados y restricciones que se aplican al elemento estereotipado. También puede asociar un icono, color o textura con el estereotipo. Normalmente, el uso de color y textura se debería evitar en modelos UML ya que algunos lectores pueden tener problemas para interpretar los diagramas, y los diagramas a menudo se imprimen en blanco y negro. Sin embargo, es una práctica común asociar un nuevo icono con un estereotipo. Esto le permite ampliar la notación gráfica de UML de forma controlada.

Puesto que los estereotipos introducen nuevos elementos de modelado con diferente propósito, tiene que definir la semántica de estos nuevos elementos en algún lugar. ¿Cómo hace esto? Bien, si la herramienta de modelado no proporciona soporte incorporado para documentar estereotipos, la mayoría de los modeladores simplemente ponen una nota en el modelo, o insertan una referencia a un documento externo en el que se definen los estereotipos. En estos momentos, el soporte de herramientas de modelado para estereotipos es bastante desigual, la mayoría de las herramientas soportan estereotipos en cierto nivel, pero no todas las herramientas proporcionan facilidades para capturar semántica de estereotipos.

Puede modelar estereotipos al utilizar el elemento clase (capítulo 7) con el estereotipo UML especial predefinido `<<stereotype>>`. Esto crea un metamodelo de su sistema de estereotipos. Es un metamodelo porque es un modelo de un elemento de modelado y se encuentra en un nivel completamente diferente de abstracción del sistema UML normal o modelos de negocio. Puesto que éste es un metamodelo, nunca debe combinarlo con sus modelos normales; siempre debe mantenerlo como un modelo aparte. Crear un nuevo modelo para los estereotipos solamente merece la pena hacerlo cuando hay numerosos estereotipos. Esto es bastante raro, por lo que la mayoría de los modeladores tienden a documentar los estereotipos con una nota o un documento externo.

Existe numerosa flexibilidad en cómo se pueden mostrar los estereotipos. Sin embargo, la mayoría de los modeladores utilizan simplemente el nombre del estereotipo en << >> o el icono. Las otras variantes no se suelen utilizar tanto y la herramienta de modelado a menudo limita lo que puede hacer. Algunos ejemplos se muestran en la figura 1.12. (Nota: Las estrellas no forman parte de la sintaxis UML, simplemente resaltan las opciones de mayor utilidad.)

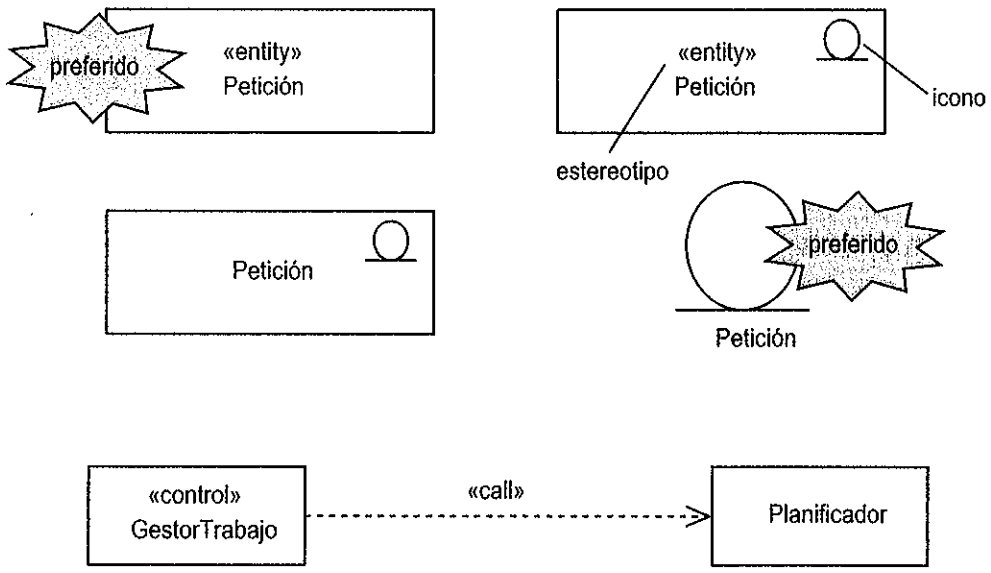



**Figura 1.12.**

Observe que puede estereotipar relaciones al igual que clases. Verá muchos usos de esto a lo largo del libro.

### 1.9.4.3. Valores etiquetados

En UML, una propiedad es cualquier valor anexado a un elemento de modelo. La mayoría de los elementos tienen grandes números de propiedades predefinidas. Algunas de éstas se muestran en diagramas y otras son simplemente parte del plano posterior semántico del modelo. UML le permite añadir sus propias propiedades para modelar elementos al utilizar valores etiquetados. Un valor etiquetado es una idea muy sencilla, es simplemente una palabra clave que puede tener un valor anexado. La sintaxis para los valores etiquetados se muestra aquí: `{etiqueta1=valor1, etiqueta2=valor2, ..., etiquetaN=valorN}`. Se trata de una lista delimitada por comas de pares etiqueta/valor separados por un signo de igual. La lista de etiquetas se sitúa entre llaves.

Algunas etiquetas son simplemente información adicional aplicada a un elemento de modelo como `{autor=Jim Arlow}`. Sin embargo, otras etiquetas indican propiedades de los nuevos elementos de modelado definidos por un estereotipo. No debería aplicar estas etiquetas directamente en elementos de modelo; en su lugar, debería asociarlas con el propio estereotipo. Luego, cuando el estereotipo se aplica a un elemento de modelo, también consigue las etiquetas asociadas con ese estereotipo.

### 1.9.4.4. Perfiles UML

Un perfil UML es una colección de estereotipos, valores etiquetados y restricciones. Utilice un perfil UML para personalizar UML para una finalidad específica.

Los perfiles UML le permiten personalizar UML para que pueda aplicarlo de forma eficaz en muchos dominios diferentes. Los perfiles le permiten utilizar estereotipos, etiquetas y restricciones de una forma coherente y bien definida. Por ejemplo, si quiere utilizar UML para modelar una aplicación .NET, entonces puede utilizar el perfil UML para .NET que se resume en la tabla 1.4.

**Tabla 1.4.**

| Estereotipo | Etiquetas | Restricciones | Extiende | Semántica |
|---|---|---|---|---|
| `<<NETComponent>>` | Ninguno | Ninguno | Componente | Representa un componente en el .NET framework. |
| `<<NETProperty>>` | Ninguno | Ninguno | Propiedad | Representa una propiedad de un componente. |
| `<<NETAssembly>>` | Ninguno | Ninguno | Paquete | Un paquete en tiempo de ejecución .NET para componentes. |
| `<<MSI>>` | Ninguno | Ninguno | Artefacto | Un archivo de componente autoinstalador. |
| `<<DLL>>` | Ninguno | Ninguno | Artefacto | Un ejecutable portable de tipo DLL. |
| `<<EXE>>` | Ninguno | Ninguno | Artefacto | Un ejecutable portable de tipo EXE. |

Este perfil es uno de los ejemplos de perfiles UML en la especificación UML 2.0 [UML2S] y define nuevos elementos de modelado UML que se adaptan para modelar aplicaciones .NET. Cada estereotipo en un perfil extiende uno de los elementos de metamodelo de UML (una Clase o Asociación) para crear un nuevo elemento personalizado. El estereotipo puede definir nuevas etiquetas y restricciones que no eran parte del elemento original.

## 1.10. Arquitectura

*The UML Reference Manual* [Rumbaugh 1] define la arquitectura de sistema como "la estructura organizacional de un sistema, incluida su descomposición en partes, su conectividad, interacción, mecanismos y los principios directores que informan al diseño de un sistema". IEEE define la arquitectura de sistema como "el concepto de más alto nivel de un sistema en su entorno".

La arquitectura trata sobre capturar los aspectos estratégicos de la estructura de alto nivel de un sistema. Existen muchas formas de examinar la arquitectura, pero una forma muy común es "Vistas 4+1" descrito por Phillippe Kruchten [Kruchten 2]. Los aspectos esenciales de la arquitectura del sistema se capturan en cuatro vistas diferentes del sistema: la vista lógica, la vista de proceso, la vista de implementación y la vista de despliegue. Todas éstas están integradas por una quinta vista, la vista de caso de uso. Cada una de estas vistas aborda diferentes aspectos de la arquitectura de software como se indica en la figura 1.13. Examinemos cada una de estas vistas:

- **Vista lógica:** Captura el vocabulario del ámbito del problema como un conjunto de clases y objetos. El énfasis está en mostrar cómo los objetos y las clases que componen un sistema implementan el comportamiento requerido del sistema.
- **Vista de proceso:** Modela los procesos en un sistema como clases activas. Es una variación orientada al proceso de la vista lógica y contiene los mismos artefactos.
- **Vista de implementación:** Modela los archivos y componentes que conforman la base de código físico del sistema. También trata sobre ilustrar dependencias entre componentes y sobre la gestión de la configuración de conjuntos de componentes para definir una versión del sistema.
- **Vista de despliegue:** Modela el despliegue físico de artefactos en un conjunto de nodos físicos computacionales como ordenadores y periféricos. Le permite modelar la distribución de artefactos en los nodos de un sistema distribuido.
- **Vista de caso de uso:** Captura los requisitos básicos para el sistema como un conjunto de casos de uso (véase el capítulo 4). Estos casos de uso proporcionan la base para la construcción de otras vistas.

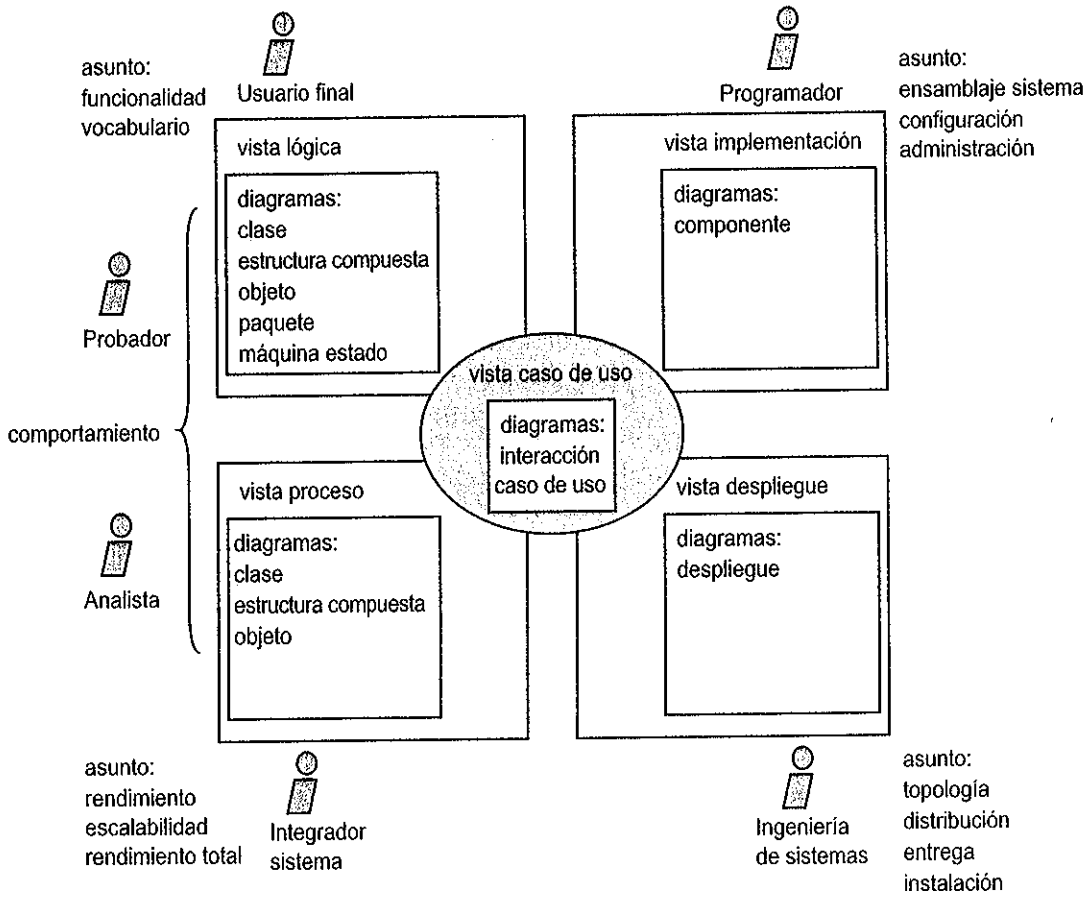



**Figura 1.13.** Adaptada de la figura 5.1 [Kruchten 1] con permiso de Addison-Wesley.

Como verá en el resto del libro, UML proporciona un soporte excelente para cada una de las vistas 4+1 y UP es un enfoque dirigido por requisitos que encaja muy bien en este modelo 4+1.

Una vez que ha creado sus vistas 4+1, ha explorado todos los aspectos clave de la arquitectura del sistema con modelos UML. Si permite el ciclo de vida iterativo de UP, esta arquitectura 4+1 no se crea en un paso, sino que evoluciona con el tiempo. El proceso del modelado UML dentro del marco de trabajo de UP es un proceso de mejora hacia una arquitectura 4+1 que captura suficiente información sobre el sistema para permitir que se cree.

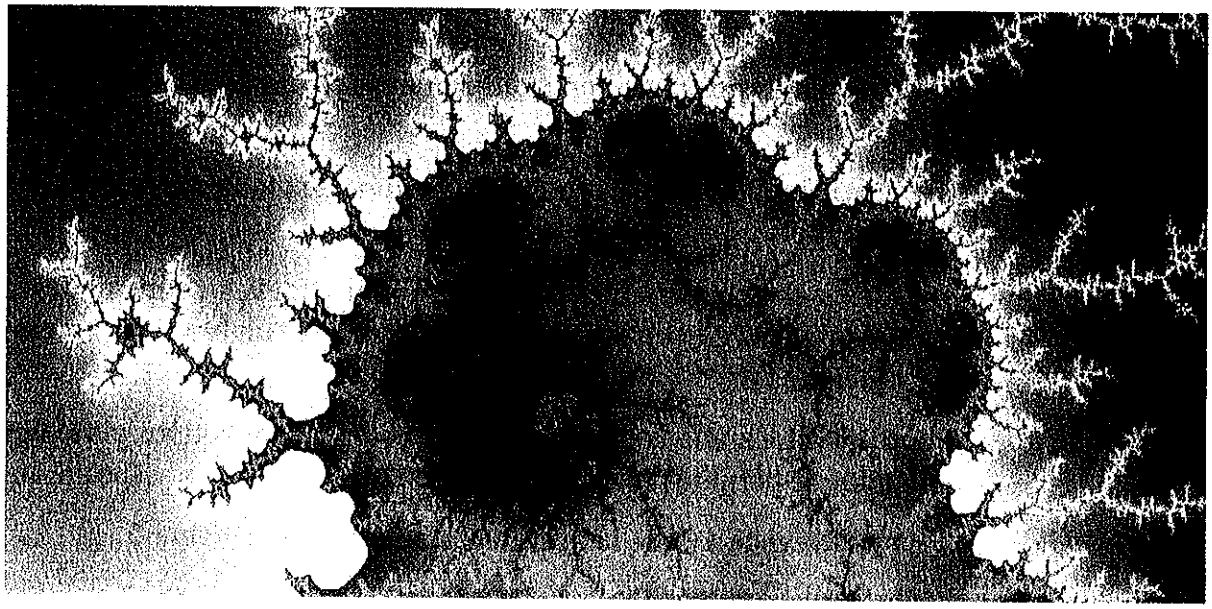
## 1.11. ¿Qué hemos aprendido?

Este capítulo ha proporcionado una introducción a la historia, estructura, conceptos y características clave de UML. Ha aprendido lo siguiente.

- El Lenguaje Unificado de Modelado (UML) es un lenguaje de modelado visual, estándar de la industria, abierto y extensible aprobado por el Object Management Group.
- UML no es una metodología.

- El Proceso Unificado (UP), o una variante, es el tipo de metodología que mejor complementa UML.
- El modelado de objetos considera el mundo como sistemas de objetos que interactúan. Los objetos contienen información y pueden realizar funciones. Los modelos UML tienen:
  - Estructura estática: Qué tipos de objetos son importantes y cómo se relacionan.
  - Comportamiento dinámico: Cómo los objetos colaboran conjuntamente para realizar las funciones del sistema.
- UML se compone de tres bloques construcción:
  - Elementos:
    - Elementos estructurales son los nombres de un modelo UML.
    - Elementos de comportamiento son los verbos de un modelo UML.
    - Existe un solo elemento de agrupación, el paquete; éste se utiliza para agrupar semánticamente elementos relacionados.
    - Existe solamente un elemento de anotación, la nota, igual que un post-it.
  - Las relaciones enlazan elementos.
  - Los diagramas muestran vistas interesantes del modelo.
- UML tiene cuatro mecanismos comunes:
  - Especificaciones: Descripciones textuales de las características y semántica de los elementos del modelo; lo más importante del modelo.
  - Adornos: Elementos de información presentados sobre un elemento de modelado en un diagrama para ilustrar un punto.
  - Divisiones comunes:
    - Clasificador e instancia:
      - Clasificador: La noción abstracta de un tipo de elemento; por ejemplo, cuenta bancaria.
      - Instancia: Una instancia específica de un tipo de elemento; por ejemplo, mi cuenta bancaria.
    - Interfaz e implementación:
      - Interfaz: Un contrato que especifica el comportamiento de un elemento.
      - Implementación: Los detalles específicos de cómo funcionan los elementos.

- Mecanismos de extensibilidad:
  - Las restricciones nos permiten añadir nuevas reglas para modelar elementos.
  - Los estereotipos introducen nuevos elementos de modelado basándose en los antiguos.
  - Los valores etiquetados nos permiten añadir nuevas propiedades a elementos de modelo, un valor etiquetado es una palabra clave con un valor asociado.
  - Un perfil UML es una colección de restricciones, estereotipos y valores etiquetados; le permite personalizar UML para una finalidad específica.
- UML se basa en un vista 4+1 de la arquitectura de sistema:
  - Vista lógica: Funcionalidad del sistema y vocabulario.
  - Vista de proceso: Rendimiento del sistema, escalabilidad y rendimiento total de procesamiento.
  - Vista de implementación: Ensamblaje y gestión de la configuración del sistema.
  - Vista de despliegue: Topología del sistema, distribución, entrega e instalación.
  - Estos están unidos por la vista de caso de uso, que describe requisitos de los grupos de decisión.



# 2

# ¿Qué es el proceso unificado?