Introducción al Análisis y Diseño Orientado a Objetos

La construcción de un sistema software, con independencia de su tamaño, de sus características funcionales y de la tecnología elegida, consta de una serie de fases que abarcan desde su concepción hasta su retirada, definiendo un espacio temporal que recibe el nombre de ciclo de vida del software. Existen diferentes modelos de ciclo de vida, cada uno con sus propias peculiaridades, adaptándose unos mejor que otros a los distintos paradigmas o estilos de programación. Pero, lo que sí se puede afirmar es que en ninguno de estos modelos de desarrollo se comienza un proyecto software por la fase de implementación.

Comenzar un proyecto software por la fase de implementación, esto decir, colocarse inmediatamente delante del ordenador y comenzar a generar código fuente, es desgraciadamente una forma de trabajo bastante extendida, que toma sus tintes más preocupantes cuando sale del entorno del programador ocasional o aficionado, para convertirse en la forma de trabajo de la inmensa mayoría de las empresas de construcción de software dentro y fuera de nuestras fronteras.

Por tanto, puede parecer utópico y dar la sensación de predicar en el desierto, el dedicar un artículo al análisis y el diseño en la orientación a objeto, cuando prácticamente nadie se molesta en seguir unos principios metodológicos básicos en sus desarrollos y, además, la orientación a objeto en nuestro país no acaba de convertirse en una alternativa completamente aceptada. No obstante, desde nuestra modesta posición vamos a intentar aportar nuestro granito de arena en favor de lo que sería una forma más correcta de realizar la construcción de una aplicación software desde el prisma de la orientación a objetos.

Fases de un desarrollo orientado a objetos

Nada más lejos de la intención de este artículo que realizar la descripción de ningún modelo de ciclo de vida del software, ya que para este fin el lector puede recurrir a cualquiera de los textos clásicos sobre ingeniería del software, por ejemplo [1] y [2], donde encontrará una información más amplia y precisa. Pero, dado que se ha comenzado el artículo criticando la construcción de las aplicaciones que empiezan directamente por la codificación, se va a dar un marco muy general de lo que sería un desarrollo software, marco que en su generalidad es perfectamente válido para cualquier tipo de desarrollo, independientemente que sea orientado a objeto o no.

El marco de desarrollo de una aplicación software estaría formado por las tres fases tradicionales: análisis, diseño e implementación.

El análisis es la fase cuyo objetivo es estudiar y comprender el dominio del problema, es decir, el análisis se centra en responder al interrogante ¿QUÉ HACER?

El diseño, por su parte, dirige sus esfuerzos en desarrollar la solución a los requisitos planteados en el análisis, esto es, el diseño se haya centrado en el espacio de la solución, intentando dar respuesta a la cuestión ¿CÓMO HACERLO?

Por último, la fase de implementación sería la encarga de la traducción del diseño de la aplicación al lenguaje de programación elegido, adaptando por tanto la solución a un entorno concreto.

Que este marco de desarrollo sea válido tanto para los desarrollos tradicionales (*desarrollos estructurados*) como para los desarrollos orientados a objeto, no significa que la realización de las actividades propias de cada fase se lleve a cabo de la misma manera. De hecho, en los desarrollos estructurados hay mucha distancia entre las fases de análisis de diseño, e incluso entre los diferentes modelos generados en una misma fase. Esta separación se conoce con el nombre de *gap semántico*, y es la barrera que la orientación a objeto intenta eliminar difuminando la frontera entre las diferentes fases.

Los métodos orientados a objeto acortan la distancia existente entre el espacio de conceptos (*lo que los expertos o usuarios conocen*) y el espacio de diseño (*lo que conocen los desarrolladores*) e implementación, ya que los objetos del mundo real tienen una correspondencia bastante clara con los objetos del sistema informático, evitando así los grandes abismos existentes entre el análisis y el diseño en el enfoque estructurado, esto es la falta de continuidad en la representación de los conceptos en una y otra fase, por ejemplo los DFDs y los diagramas de estructura, como muestra la Figura 1.

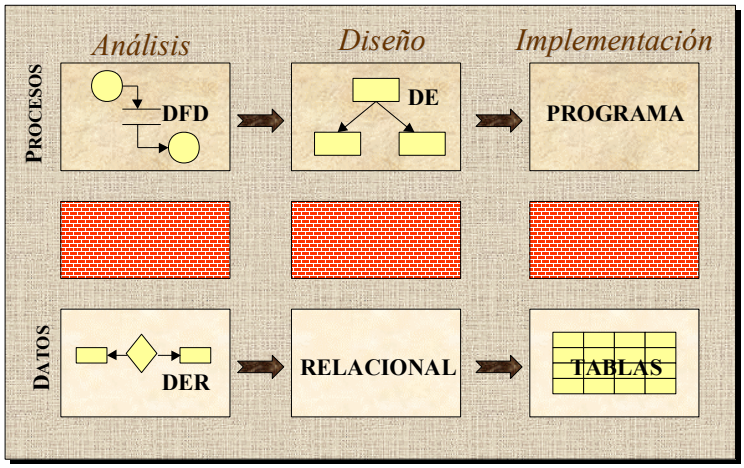

Figura 1. Desarrollos estructurados.

Por su parte, en los desarrollos orientados al objeto se tiene una mayor continuidad entre las diferentes fases, con unas fronteras entre fases muy poco marcadas que dan lugar a desarrollos más iterativos e incrementales. Todo esto es posible gracias a una característica de vital importancia, el modelo subyacente a todas las fases es el mismo, el modelo objeto, que tiene como elemento central al objeto, que es la entidad que encapsula elementos estructurales y de comportamiento. De esta forma, los objetos semánticos identificados en la fase de análisis se refinarán durante la fase de diseño e implementación, a la vez que se añaden objetos de interfaz y de utilidad para constituir la aplicación final, como se puede apreciar en la Figura 2.

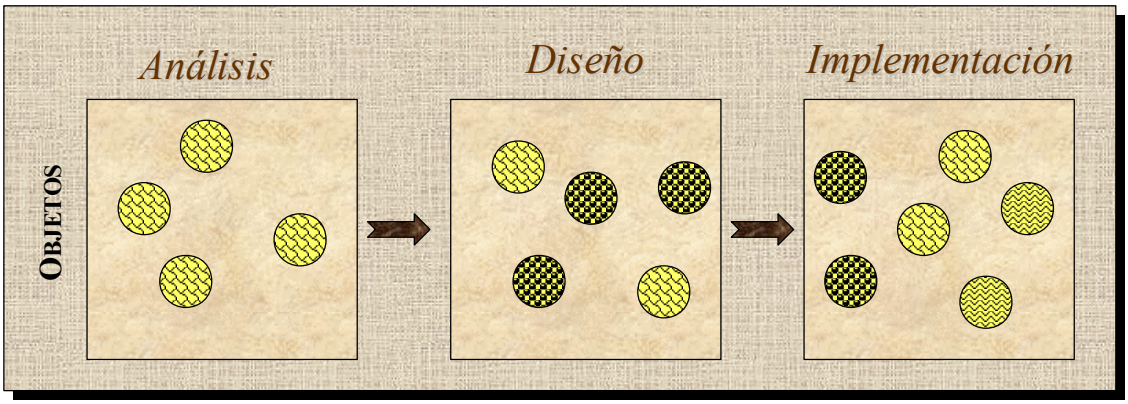

Figura 2. Desarrollos orientados a objeto.

Como referencia de modelos objeto concretos puede hacerse mención al modelo de objeto de OMT, definido por James Rumbaugh en [3], o al modelo objeto de UML [4], entre muchos otros.

Análisis y diseño orientados a objetos (ADOO)

Como se ha comentado en el apartado anterior, la transición entre las fases de análisis y diseño en la orientación al objeto es mucho más suave que en las metodologías estructuradas, no habiendo tanta diferencia entre las etapas. Esto implica que es difícil determinar donde acaba el Análisis Orientado a Objeto (AOO) y donde comienza el Diseño Orientado a Objeto (DOO), siendo la frontera entre ambas fases totalmente inconsistente, de forma que lo que algunos autores incluyen en el AOO otros lo hacen en el DOO. Esto conduce a que sea frecuente el uso de las siglas ADOO para hacer referencia a ambas fases de forma conjunta.

El objetivo del AOO es modelar la semántica del problema en términos de objetos distintos pero relacionados. Por su parte, el DOO conlleva reexaminar las clases del dominio del problema, refinándolas, extendiéndolas y reorganizándolas, para mejorar su reutilización y tomar ventaja de la herencia. El análisis casa con el dominio del problema y el diseño con el dominio de la solución; por lo tanto el AOO enfoca el problema en los objetos del dominio del problema y el DOO en los objetos del dominio de la solución.

Los objetos del dominio del problema representan cosas o conceptos utilizados para describir el problema, recibiendo el nombre de objetos semánticos porque ellos representan las abstracciones que encierran el significado del dominio del problema.

El análisis se centra en la representación del problema, es decir, en la identificación de las abstracciones que representen el significado de las especificaciones y de los requisitos del sistema.

El énfasis del diseño se centra en la definición de la solución. Los objetos semánticos serán refinados durante la fase de análisis y de diseño, siendo completados con los objetos propios del dominio de la solución.

Los objetos del dominio de la solución incluyen: *objetos de interfaz, objetos de aplicación* y *objetos base o de utilidad***.** Éstos no forman parte directamente de los objetos del dominio problema, pero representan la vista del usuario de los objetos semánticos.

Una vez realizada esta introducción al AOO y al DOO se puede proceder a dar una definición más concreta de ambos procesos.

Se puede definir AOO como *el proceso que modela el dominio del problema identificando y especificando un conjunto de objetos semánticos que interaccionan y se comportan de acuerdo a los requisitos del sistema*.

Se puede definir DOO como *el proceso que modela el dominio de la solución, lo que incluye a las clases semánticas con posibles añadidos, y las clases de interfaz, aplicación y utilidad identificadas durante el diseño*.

No debiera darse por terminado este apartado sin enunciar cuales son las actividades propias del AOO y del DOO. Este es un asunto comprometido por la diversidad existente en el mundo de las metodologías orientadas a objeto, como se puso de manifiesto en [5], pero de forma general podría afirmarse que:

- En el AOO deben llevarse a cabo las siguientes actividades:

- *La identificación de clases semánticas, atributos y servicios*
- *Identificación de las relaciones entre clases (generalizaciones, agregaciones y asociaciones).*
- *El emplazamiento de las clases, atributos y servicios.*
- *La especificación del comportamiento dinámico mediante paso de mensajes.*

- En el DOO deben llevarse a cabo las siguientes actividades:
  - *Añadir las clases interfaz, base y utilidad.*
  - *Refinar las clases semánticas.*

Como resumen final, se podría afirmar que el AOO y el DOO no deben verse como fases muy separadas, siendo recomendable llevarlas a cabo concurrentemente, así el modelo de análisis no puede completarse en ausencia de un modelo de diseño, ni viceversa. Uno de los aspectos más importantes del ADOO es la sinergia entre los dos conceptos.

Identificación de objetos

La identificación de objetos es la clave o el cuello de botella a la hora de aplicar tanto el diseño como el análisis orientados al objeto. Existen varios enfoques en cuanto a las técnicas a aplicar para identificar cuales son las abstracciones que mejor representan o recogen la semántica del problema que se desea resolver. Aquí se van a comentar brevemente las tres técnicas más difundidas para llevar a cabo esta actividad. Estas son:

- ***El análisis textual.***
- ***Las tarjetas CRC.***
- ***Las formas de utilización.***

Análisis textual

Es el más simple e intuitivo de las técnicas que aquí se van explicar. Parte de una descripción del problema en lenguaje natural (*en nuestro caso sería una descripción realizada en español*) y consistiría en extraer los objetos, los atributos, los servicios y las relaciones entre los objetos mediante un análisis lingüístico del enunciado del problema de acuerdo a la tabla representada en la Figura 3.

Parte de la oración	*Componente del modelo de objetos*	*Ejemplo*
Nombre propio	*Instancia*	Puzzle
Nombre común	*Clase*	Juguete
Verbo acción	*Operación*	Guardar
Verbo de existencia	*Clasificación*	Es un
Verbo de posesión	*Composición*	Tiene un
Verbo afirmativo	*Condición de invarianza*	Posee
Adjetivo	*Valor o clase (atributo)*	Inadecuado
Frase adjetiva	*Asociación*	Cliente con niños
	Operación	Cliente que compró el puzzle
Verbo transitivo	*Operación*	Entrar
Verbo intransitivo	*Excepción o suceso*	Depende

Figura 3. Criterios para el análisis lingüístico.

Tarjetas CRC

Las tarjetas CRC (*Class, Responsability and Collaboration- Clases, Responsabilidades, y Colaboraciones*), también denominadas tarjetas de clase, constituyen una forma simple y efectiva de analizar escenarios.

Esta técnica consiste en elaborar una ficha o tarjeta por cada clase en la que se anotan los siguientes datos: *el nombre de la clase, la lista de sus superclases, la lista de sus subclases, sus responsabilidades* y *sus colaboraciones*, como se puede apreciar en la Figura 4.

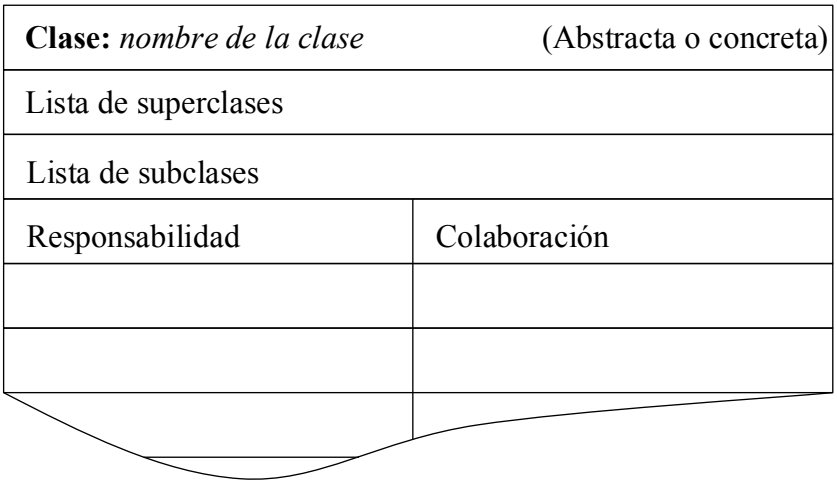

Figura 4. Tarjeta de clase.

Las responsabilidades de una clase son todos los servicios que proporciona la clase a sus clientes potenciales. Por su parte, las colaboraciones de una clase representan las peticiones que hace una clase a ciertos servidores para poder cumplir sus responsabilidades.

La forma de trabajo con las tarjetas de clase consiste en identificar las primeras clases semánticas. Estas clases se examinan para determinar cómo envían mensajes a otras y cuáles son sus responsabilidades, comprobando si cada clase posee todas las características necesarias para atender las peticiones que le llegan. Para la validación de las clases, cada miembro del equipo de desarrollo toma el papel de una o varias clases realizando una simulación de las responsabilidades y colaboraciones de éstas.

Formas de utilización

Las formas de utilización o casos de uso se deben a Ivar Jacobson y fueron presentadas en el OOPSLA'87. Un caso de uso es una forma, patrón o ejemplo concreto de utilización, un escenario que comienza con algún usuario del sistema que inicia alguna transacción o secuencia de eventos interrelacionados [6].

Las formas de utilización se introducen en la etapa de análisis para representar escenarios concretos que ayudan a identificar los objetos que intervienen en dichos escenarios.

Una forma de utilización consta de dos elementos principales, los actores y las formas de utilización. Un actor representa cualquier elemento que intercambia información con el sistema, por lo que está fuera del sistema, y se representan por el típico monigote. Una forma de utilización representa

Figura 5. Elementos de una forma de utilización

una secuencia de transacciones en un diálogo con el sistema que se encuentran relacionadas por su comportamiento, las formas de utilización se representan por elipses.

Modelado de objetos

Los objetos identificados en la etapa de análisis se plasman en diferentes modelos, combatiendo de esta forma la complejidad inherente del problema al que nos estamos enfrentando.

Para la construcción de modelos de objetos, y modelos software en general, se debe contar con un lenguaje de modelado, es decir, un lenguaje para especificar, visualizar, construir y documentar los componentes de los sistemas software.

En [5] se presentó a UML 1.1 como el lenguaje de modelado que se había convertido en estándar de facto de las notaciones en orientación a objeto, estando a la espera del reconocimiento por parte de OMG como el estándar oficial de las notaciones para el modelado de sistemas software. En el presente artículo, ya podemos anunciar que esta espera ha terminado, y UML 1.1 es ya el estándar oficial aceptado por OMG como lenguaje de modelado.

El hecho de la estandarización de la notación de UML nos conduce hacia una notación única en la que expresar las abstracciones identificadas en análisis, así como sus relaciones. Además, esos mismos modelos serán refinados durante la fase de análisis y diseño hasta que estén preparados para su implementación en el lenguaje de programación elegido.

A la hora de realizar un modelo de un sistema software, éste debe hacerse desde diferentes puntos de vista, de forma que recojan tanto la dimensión estática y estructural de los objetos como su componente dinámica.

Un lenguaje de modelado debe aportar una serie de mecanismos gráficos que permitan captar la esencia de un sistema desde diversas perspectivas. Estos mecanismos gráficos suelen recibir el nombre de diagramas en la mayoría de los lenguajes de modelado.

UML 1.1 cuenta con una amplia gama de diagramas para que el analista pueda reflejar las diferentes dimensiones de un sistema. Estos diagramas se dividen en las siguientes categorías:

- Diagramas estáticos de estructura
- Diagramas de formas de utilización (*casos de uso*)
- Diagramas de interacción
- Diagramas de transición de estados
- Diagramas de actividad
- Diagramas de implementación

Diagramas estáticos de estructura

Estos diagramas proporcionan una notación gráfica para el modelado conceptual del sistema, es decir, permiten representar las abstracciones identificadas en forma de clases, objetos y sus interrelaciones. Existen dos tipos de diagramas estáticos de estructura: *los diagramas de clases* y *los diagramas de objetos*.

Los diagramas de clases representan un esquema conceptual que describe muchas instancias de datos posibles, por tanto, los diagramas de clases van a describir clases de objetos.

Durante el análisis se usan los diagramas de clase para indicar los roles comunes y las responsabilidades de las entidades que sustentan el comportamiento del sistema, mientras que durante el diseño, se utilizan para capturar la estructura de las clases que forman la arquitectura del sistema.

Por su parte un diagrama de objetos estático es una instancia del diagrama de clases, mostrándose como una instantánea del estado del sistema en un momento dado. Los diagramas de objetos describen, por tanto, la forma en que un cierto conjunto de objetos se relacionan entre sí.

De lo expresado aquí se puede afirmar que un diagrama de clases dado se corresponde con un conjunto infinito de diagramas de objetos que serían instancias de dicho diagrama de clases.

Diagramas de formas de utilización

Ya presentamos anteriormente las formas de utilización como una forma de identificación de objetos en el análisis orientado a objetos. UML 1.1 tiene soporte para las formas de utilización, y dado que su inventor Ivar Jacobson es uno de los creadores destacados de UML, la notación de estos diagramas es muy similar a la notación original.

Como resumen de lo qué es un diagrama de formas de utilización, podíamos decir que se trata de un diagrama donde una serie de entidades externas al sistema (*los actores*) interacciona con el sistema, comunicándose con una cierta forma de utilización.

Diagramas de interacción

Los diagramas de interacción son modelos que describen como grupos de objetos colaboran para conseguir algún fin. Se pueden distinguir los dos siguientes tipos: *diagramas de secuencia* y *diagramas de colaboración*.

Un diagrama de secuencia muestra las interacciones expresadas en función de secuencias de tiempo. En concreto muestra los objetos participantes y los mensajes que intercambian entre ellos a lo largo del tiempo. Un diagrama de secuencias tiene dos dimensiones, la vertical que representa el tiempo, y la horizontal que representa los distintos objetos. El tiempo avanza desde el comienzo hasta el final de la página, aunque se puede tomar el sentido contrario. Un ejemplo de un diagrama de secuencias se puede ver en la Figura 6.

Un diagrama de colaboración muestra cómo las instancias específicas de las clases trabajan juntas para conseguir un objetivo común. Implementa las asociaciones del diagrama de clases mediante el paso de mensajes de un objeto a otro.

La diferencia entre un diagrama de secuencias y un diagrama de colaboraciones es que este último muestra las relaciones sobre los objetos sin mostrar la dimensión temporal de dichas relaciones.

Diagramas de transición de estados

Un diagrama de transición de estados representa los estados que puede tomar un objeto mostrando, además, los eventos o circunstancias que implican el cambio de un estado a otro.

En UML, un estado es una condición que se verifica durante la vida de un objeto o de una interacción. Durante este período de tiempo se satisface alguna otra condición, se ejecuta alguna acción o se espera a la ejecución de algún evento. Un objeto permanece en un estado por un espacio finito y no instantáneo de tiempo. Cuando un objeto cambia de estado por la ocurrencia de un evento, se dice que se ha producido una transición.

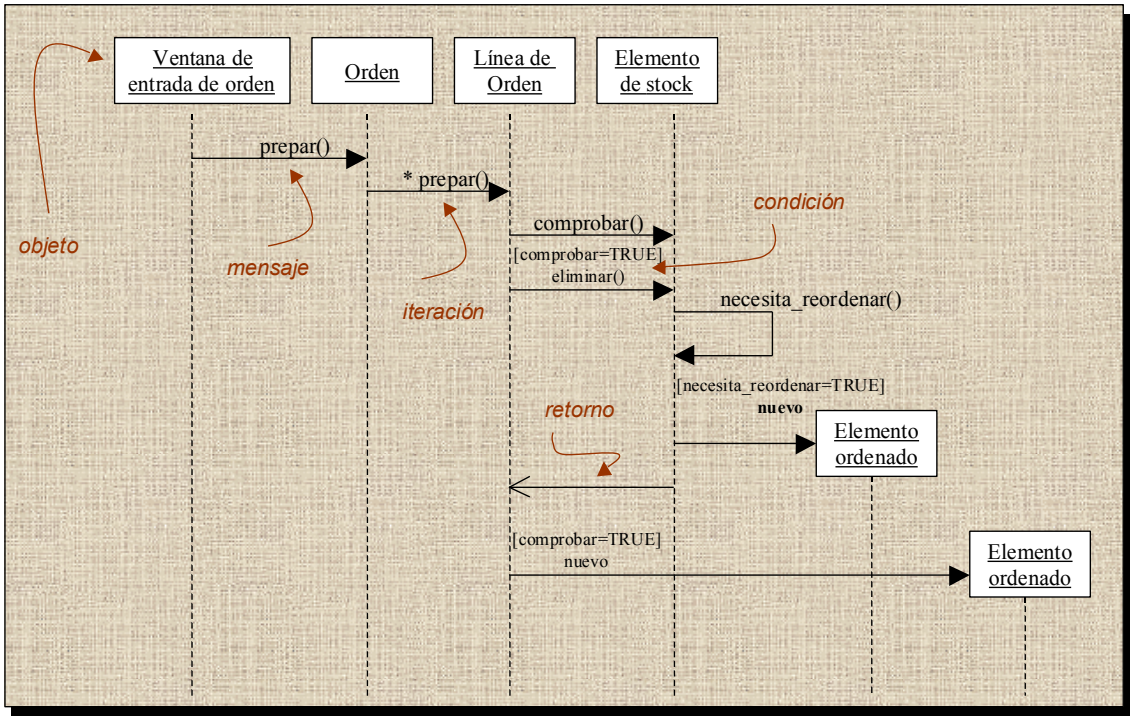

Figura 6. Ejemplo de un diagrama de secuencias de UML.

Diagramas de actividad

Los diagramas de actividad es una de las partes más inesperadas de UML. A diferencia del resto de las técnicas, los diagramas de actividad no tienen un origen claro en los trabajos previos de los creadores de UML. Este diagrama combina las ideas de varias técnicas: los diagramas de eventos de Jim Odell, las técnicas de modelado de estados de SDL, Redes de Petri.

Un diagrama de actividad es un tipo especial de diagrama de estados en el que todos o la mayoría de los estados son estados de acción, y todas o la mayoría de las transiciones son disparadas porque se ha completado la acción de los estados fuente.

El diagrama de actividad se adjunta a través del modelo a una clase, a un método o a una forma de utilización. El propósito de estos diagramas es centrarse en el flujo de los procesos internos, con independencia de los eventos externos. Por ello los utilizaremos fundamentalmente para flujos de operaciones, mientras que los diagramas de estado ordinarios serán más propios de sistemas donde puedan ocurrir eventos asíncronos.

Diagramas de implementación

Estos diagramas muestran los aspectos físicos del sistema, diferenciándose dos tipos: *los diagramas de componentes* y *los diagramas de despliegue*.

Los diagramas de componentes representan las relaciones de dependencia entre el código fuente, componentes binarios y ejecutables.

Los diagramas de despliegue se utilizan para representar los procesos y objetos en tiempo de ejecución, así como los componentes hardware en el que estos se ejecutan.

Conclusiones

Se ha intentado transmitir en este artículo la idea de que un desarrollo de software conlleva más actividades que las de implementación, plasmando someramente las principales actividades que se llevan a cabo en el ADOO y centrándonos especialmente en la tarea de identificación de los objetos que capturan la semántica de la aplicación, y en la necesidad de plasmarlos en una serie de modelos que representen el sistema.

En cuanto al tema de modelado, se ha incidido de nuevo en el uso de UML 1.1 (*que ya se introdujo en [5]*) pero haciendo especial hincapié en los diferentes tipos de diagramas que este lenguaje de modelado presenta. Somos conscientes de que la presentación de dichos diagramas ha sido demasiado superficial, pero el estudio de cada uno de ellos por separado es un tema demasiado extenso que ya se hace en [4]. No obstante, y dado la especial repercusión que tiene sobre la implementación, se realizará un estudio más detallado del diagrama de clases en un próximo artículo.

Referencias bibliográficas

[1] **Piattini, M., Calvo Manzano, J. A., Cervera, J. y Fernández, L.** "*Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión*". Ra-ma. 1996.

[2] **Pressman, Roger S.** "*Ingeniería del Software: Un Enfoque Práctico*". 4ª Ed. McGraw-Hill, 1997.

[3] **Rumbaugh, James, Blaha, Michael, Premerlani, William, Eddy, Frederick and Lorensen, William**. "*Object-Oriented Modeling and Design*". Prentice-Hall, 1991.

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

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

[6] **Jacobson, Ivar, Christerson, M., Jonsson, P., Overgaard, G.** "*Object-Oriented Software Engineering: A Use Case Driven Approach*". Addison-Wesley. 1992.

[7] **García Peñalvo, Francisco José, Marqués Corral, José Manuel y Maudes Raedo, Jesús Manuel**. "*Análisis y Diseño Orientado al Objeto para Reutilización*". Technical Report (TR-GIRO-01-97V2.1), Universidad de Valladolid. Octubre 1997.

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