élogos

PROGRAMACION ORIENTADA A OBJETOS CON JAVA

élogos

PROGRAMACION ORIENTADA A OBJETOS CON JAVA

Diseño orientado a objetos

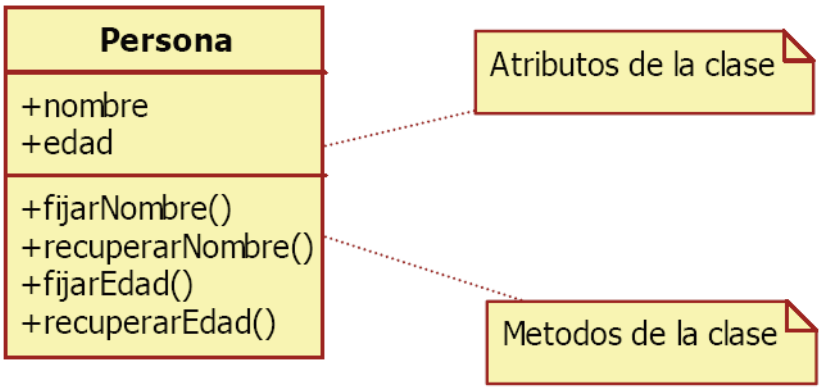
Conceptos básicos de Orientación a Objetos

A continuación vamos a analizar algunos conceptos básicos de orientación a objetos aprovechando la notación UML que estudiaremos en detalle mas adelante.

Clase

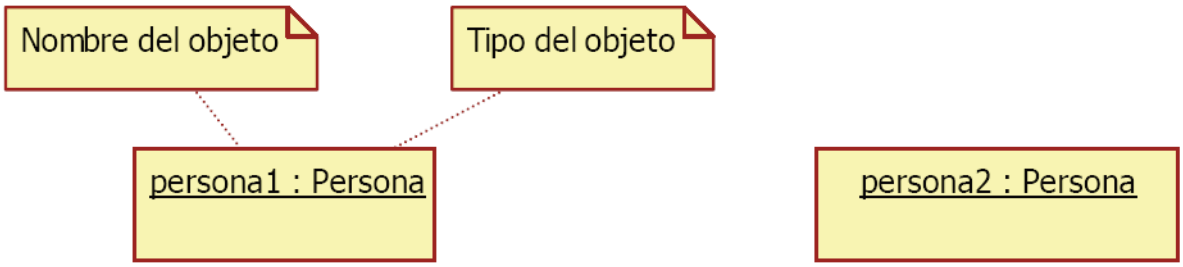
Es una abstracción de la realidad, un elemento del presente en el dominio del problema que pasará a formar parte de nuestro S.I. en una forma simplificada. Las características que definen a la clase son los atributos y los métodos.

- Los atributos también son llamados campos, variables de clase o propiedades de la clase.
- Los métodos también son llamados operaciones o funciones.



Objeto

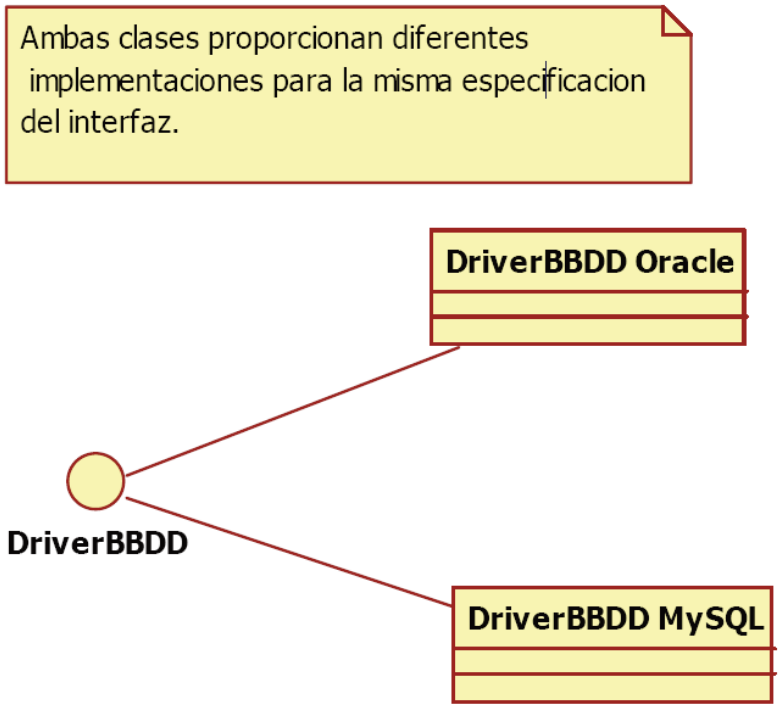
Es una instancia de la clase, un caso concreto donde la clase toma valores concretos.



Los objetos pueden tener opcionalmente un nombre que los identifica, aunque en ocasiones no es necesario indicarlo.

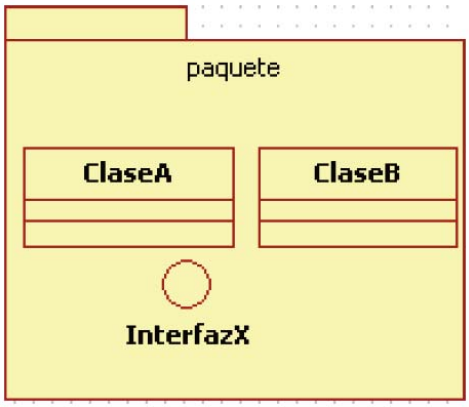
Interfaz

Define un contrato que deberá ser implementado por una clase (u otro elemento) y permitirá el uso de dicha implementación sin tener que conocer el detalle de la misma.



Paquete lógico

Agrupación lógica de clases, interfaces y otros elementos de Programación orientada a objetos.



EL LENGUAJE UML

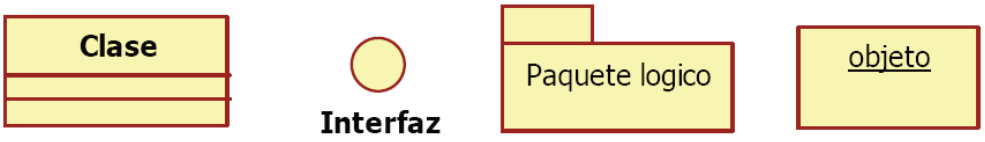
UML significa Lenguaje Unificado de Modelado.

El lenguaje UML es visual, más que programar en UML se modela visualmente.

Muchos de los elementos de UML tienen su origen en la Orientación a Objetos, otros vienen de fuentes diferentes.

Notaciones gráficas

Son los elementos visuales existentes en UML, hemos visto algunos en la sección de orientación a objetos:

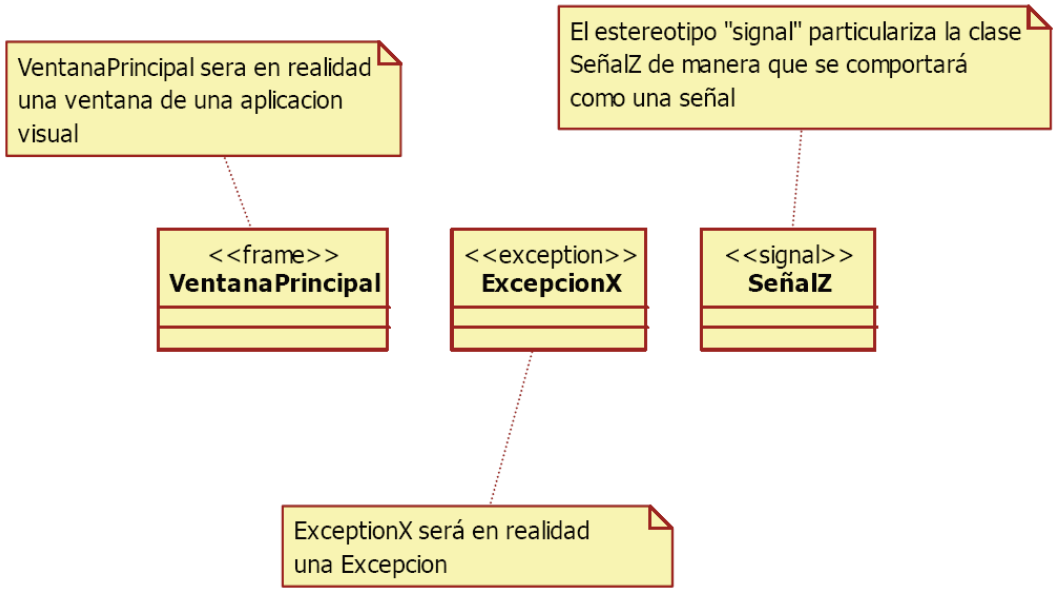


Otros sin embargo son específicos de UML, como:

Estereotipo

Es una cadena de texto que permite particular las características de un determinado elemento de UML.

Por ejemplo, una clase UML puede ser estereotipada, representar elementos diferentes a las clases de la P.O.O.



Aunque algunos estereotipos tienen un significado prácticamente universal, otros se pueden encontrar con diferentes acepciones dependiendo del entorno laboral.

Un analista, diseñador o arquitecto puede crear sus propios prototipos si una determinada situación lo aconseja.

UML reconoce la posibilidad de que un estereotipo modifique la representación visual de dicho elemento.

Relaciones

Enlaces realizados entre las notaciones gráficas con el fin de mostrar la interacción existente entre dichas notaciones. Existen diversas notaciones

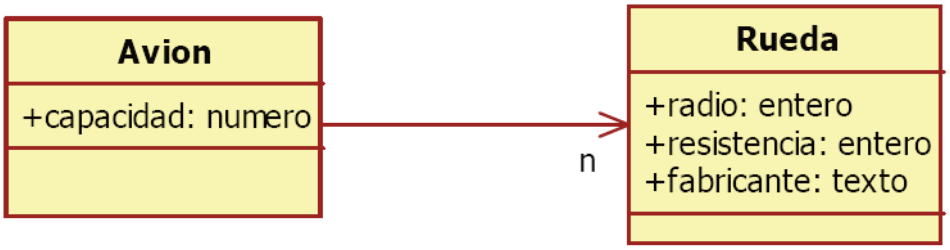
Asociación

Indica que existe una relación de uso entre los elementos vinculados con la asociación.

La relación puede tener direccionalidad indicando que elemento referencia al otro.

En ocasiones se simplifican los diagramas indicando que una asociación implica la presencia de un atributo en la clase origen de la siguiente manera.

El avión tiene un número indeterminado de ruedas



Agregación

Es una asociación donde los elementos relacionados presentan un vínculo de todo-parte.

El uso de la agregación frente a la asociación dependerá en gran medida del caso concreto que estemos modelando y de las características del sistema.

El avión tiene inexorablemente 2 alas, solo podremos emplear aviones en nuestro S.I. cuando las alas estén disponibles.

Composición

Es una asociación donde los elementos relacionados presentan un vinculo más fuerte que en la asociación y agregación.

La creación de instancias de una clase compuesta implicará la creación de instancias de las otras clases compuestas. De la misma manera la destrucción de instancias será simultánea.

Asociación versus Agregación versus Composición

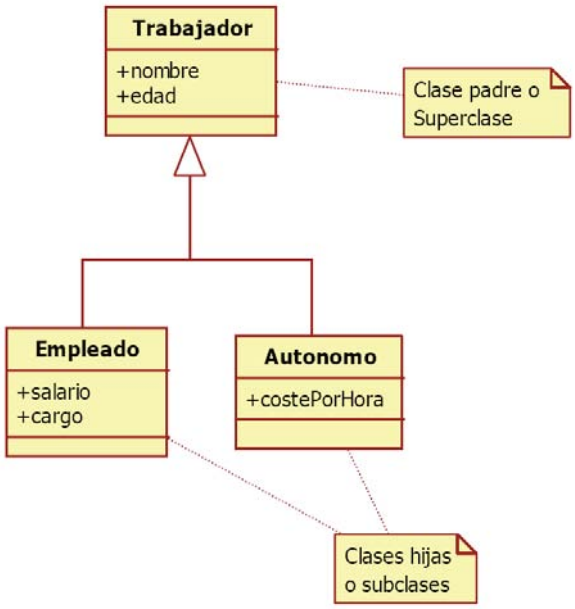
Existen diferentes teorías sobre dónde usar una asociación, una agregación o una composición.

La falta de estandarización desaconseja que se haga un uso gratuito de unas u otras si no se respeta alguna de las numerosas interpretaciones existentes y a ser posible se documente la motivación de su uso.

En caso de duda, la asociación representa tanto a la agregación como a la composición y siempre se podrá detallar en mayor medida en fases posteriores del modelado.

Herencia

Indica que existe una relación padre-hijo entre clases, de manera que la clase hija hereda las propiedades del padre.



El trabajador es la superclase o clase base. Empleado y autónomo son dos subclases.

Dependencia

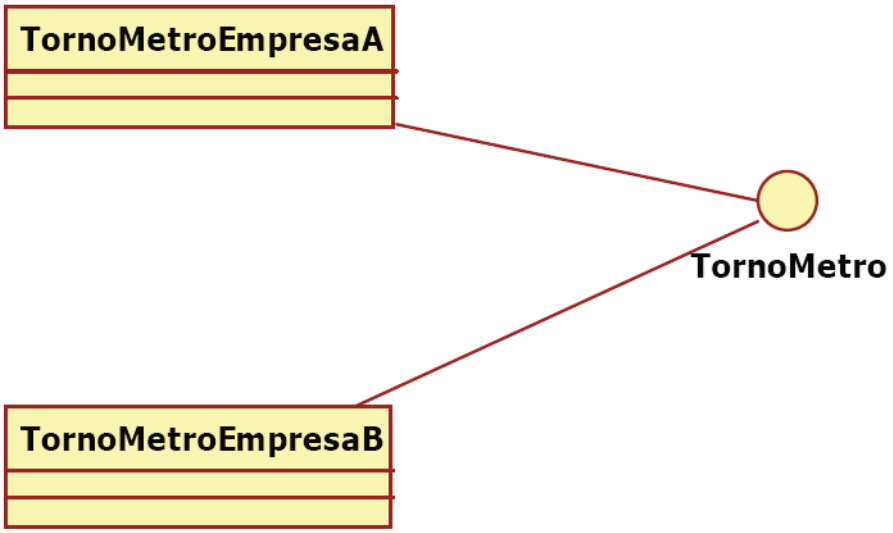
Indica que el elemento origen de la flecha precisa de alguna de las propiedades del elemento destino de la flecha para poder funcionar.

Realización

El elemento origen de la flecha proporciona una implementación para el elemento destino de la flecha.

En el siguiente ejemplo vemos que la clase ConexionBBDDOracle proporciona una implementación concreta de ConexionBBDDGenerica.

Cuando se representa una implementación entre clases e interfaces la realización se simplifica visualmente con una flecha continua sin cabeza, lo que puede provocar confusión con una relación de asociación bidireccional.



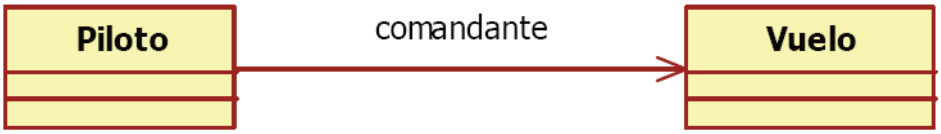
Adornos de una relación

Los adornos aportan información adicional sobre las relaciones existentes entre diferentes elementos. El uso de dichos adornos dependerá de la labor específica que estemos desempeñando en un momento dado.

En el siguiente ejemplo hemos añadido una cardinalidad (valor numérico) a la relación.

El avión tiene de n ruedas

Podemos también indicar un nombre a la relación.



El piloto es el comandante de un vuelo

O si necesitamos mayor detalle en la definición del papel que cada clase juega en la relación podemos usar los roles de las clases.

El cliente alquila un coche que él mismo conducirá

DIAGRAMAS

Diagrama de casos de uso

Modela los casos de uso del sistema (requisitos) y representa las interacciones que dichas funcionalidades deben tener con los actores vinculados a los mismos.

Los actores pueden ser tanto seres humanos como maquinas con las que nuestro sistema debe interactuar.

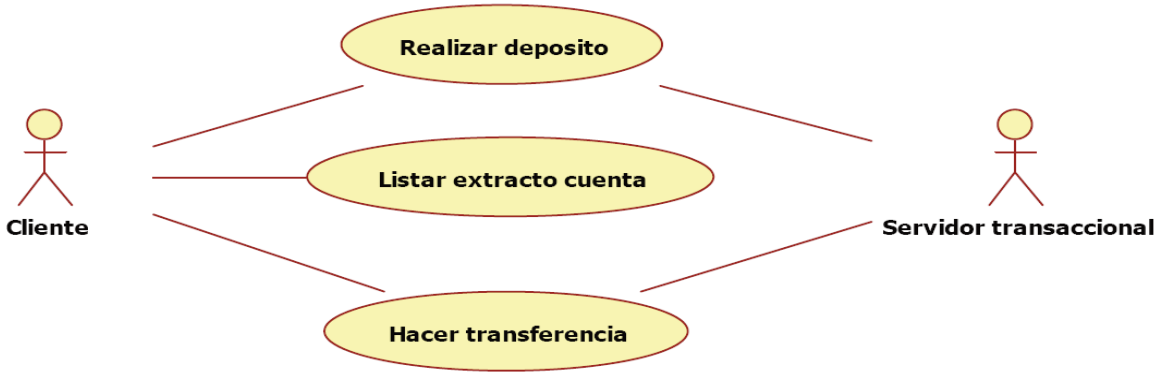


Diagrama de C.U. que modela parcialmente la funcionalidad de un S.I. de banca online.

Diagrama de clases

El diagrama de clases modela el comportamiento de las clases de un sistema.

En un sistema orientado a objetos cada clase representa un elemento que interactúa con otros a través de sus métodos para conseguir un comportamiento global.

La clase CuentaBanco usa a la clase Cliente para reflejar quién es el titular de la cuenta.

Diagrama de objetos

Muestra la interacción que se produce entre instancias de las clases (objetos).

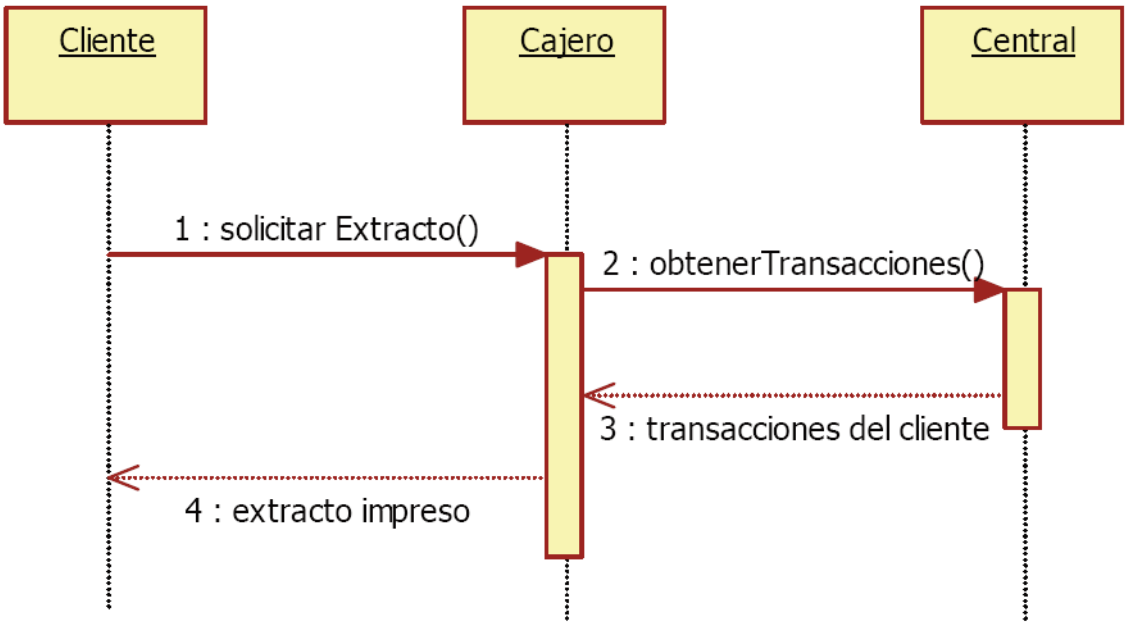
Habitualmente se emplea para modelar aspectos críticos del comportamiento dinámico del sistema.

En este caso los objetos tienen nombre, pero habitualmente se omiten.

Sin embargo de por sí sólo su utilidad es muy limitada. A veces se crea como base de un diagrama de secuencia o de comunicación.

Diagrama de secuencia

Modela la interacción que se produce entre objetos de un S.I. mediante el intercambio de mensajes.

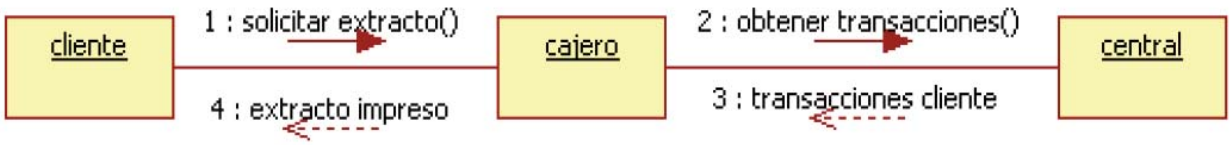


Interacción realizada por un cliente en un cajero para ver un extracto de operaciones

Diagrama de colaboración (uml 1) / comunicación (uml 2)

Al igual que el diagrama de secuencia modela la interacción que se produce entre objetos de un S.I. mediante el intercambio de mensajes.

El Diagrama de Colaboración es muy parecido al Diagrama de Secuencia, la principal diferencia es que el Diagrama de Secuencia se centra en la temporalidad existente entre diferentes mensajes entre objetos.



El cliente emplea el cajero para obtener un extracto de sus operaciones.

Diagrama de actividades

Muestra las interacciones existentes entre diferentes acciones de un mismo proceso.

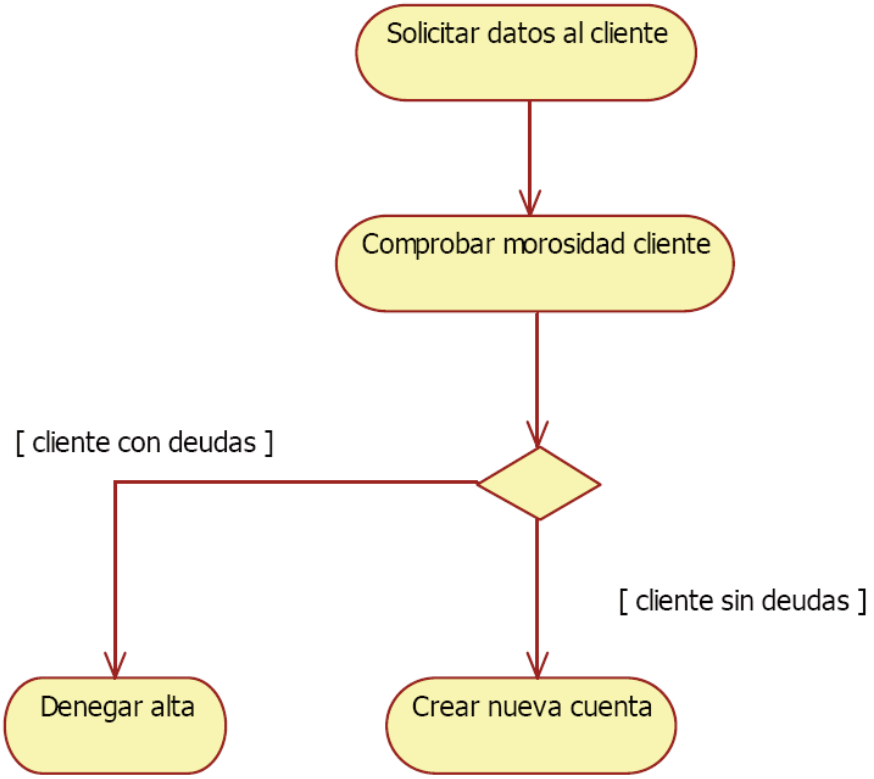


Diagrama de estados

Muestra el ciclo de vida de un determinado elemento de software o concepto de lógica de negocio.

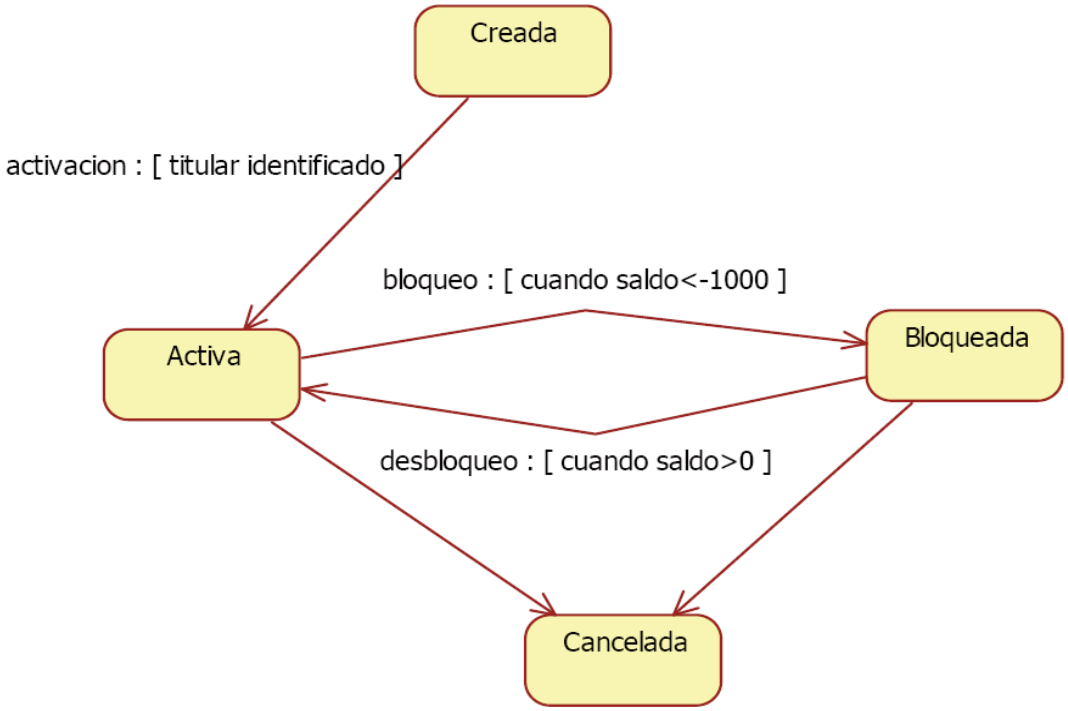


Diagrama de estados simple del ciclo de vida de una cuenta de banco

Entre llaves se ubica la condición de guarda de las transiciones. Cuando se cumpla dicha transición se transita a otro estado.

Diagrama de componentes

Modela los elementos de software que van a integrar nuestro sistema.

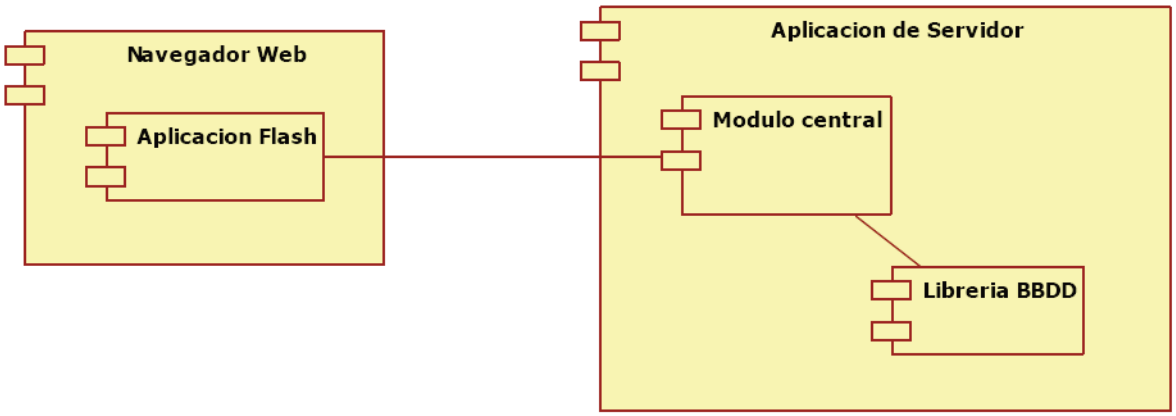


Diagrama de componentes de un S.I. para Web que emplea una aplicación flash para la presentación al cliente final

Diagrama de despliegue

Modela los elementos de hardware que van a integrar el sistema que se está desarrollando. En ocasiones los elementos del Diagrama de componentes se distribuyen sobre los nodos del Diagrama de Despliegue.

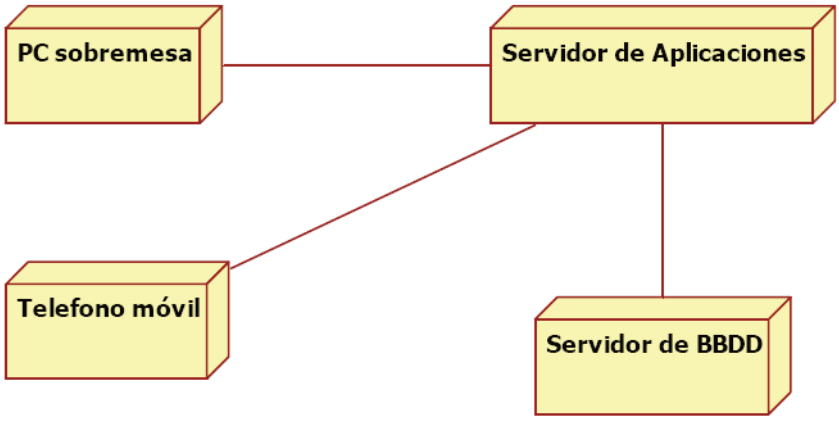
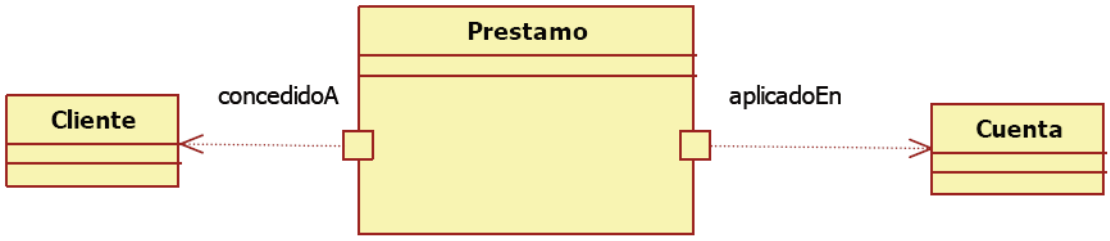


Diagrama de despliegue para una aplicación típica accesible desde un dispositivo móvil y un PC.

Diagrama de estructura compuesta

Modela los vínculos estructurales existentes entre clases con mayor detalle que un diagrama de clases convencional.

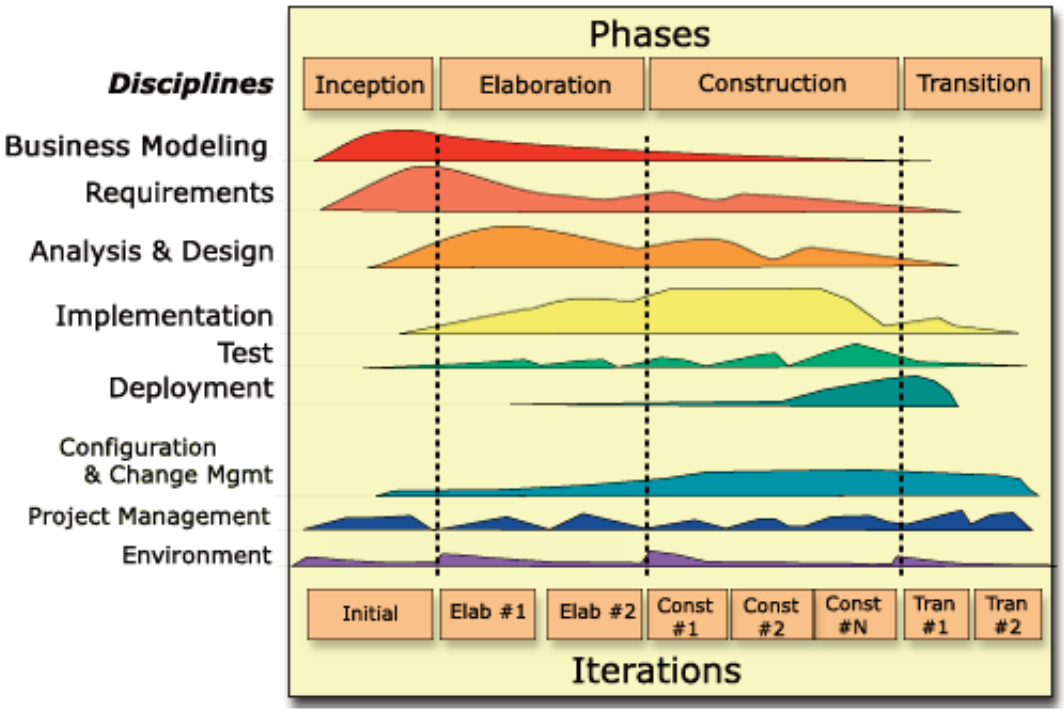


La clase Préstamo juega dos roles diferentes con dos clases diferentes

EL PROCESO UNIFICADO DE DESARROLLO

Es una metodología basada en el desarrollo iterativo del proyecto. El Proceso Unificado de desarrollo más conocido es el RUP (Racional Unified Process)

Se distingue del proceso iterativo clásico en que incluye una nueva variante: cada fase incluye el desempeño del trabajo en diferentes disciplinas del proyecto.



Fases del proceso

Definición de casos de caso

La definición de casos de uso consiste en extraer, recopilar, estructurar y modularizar los requisitos del S.I. que se debe desarrollar.

Habitualmente estos requisitos se extraen del cliente que solicita el sistema aunque en proyectos de reingeniería los requisitos pueden extraerse en gran medida de un S.I. preexistente.

La extracción se realiza en un diagrama de casos de uso (abreviado C.U.) y los C.U. se detallarán en mayor medida en una ficha de C.U.

La ficha de C.U. es un documento que detalla información sobre los C.U. definidos, esta información puede variar dependiendo del proyecto pero habitualmente tiene las siguientes características:

- Id: identificador que permite referenciar de forma única a un determinado C.U.
- Nombre: nombre del C.U.
- Descripción: descripción breve del C.U.
- Flujo principal: indica la secuencia de pasos que llevan a la consecución de la funcionalidad especificada por este C.U.
- Flujos secundarios: indican la secuencia de pasos que llevan a situaciones no deseables que se pueden presentar al intentar ejecutar este C.U.
- Precondiciones: condiciones que se deben cumplir antes de poder ejecutar la funcionalidad especificada en este C.U.
- Postcondiciones: condiciones que se deben cumplir al finalizar la ejecución de la funcionalidad especificada en este C.U.

Captura de requisitos de una tienda online:

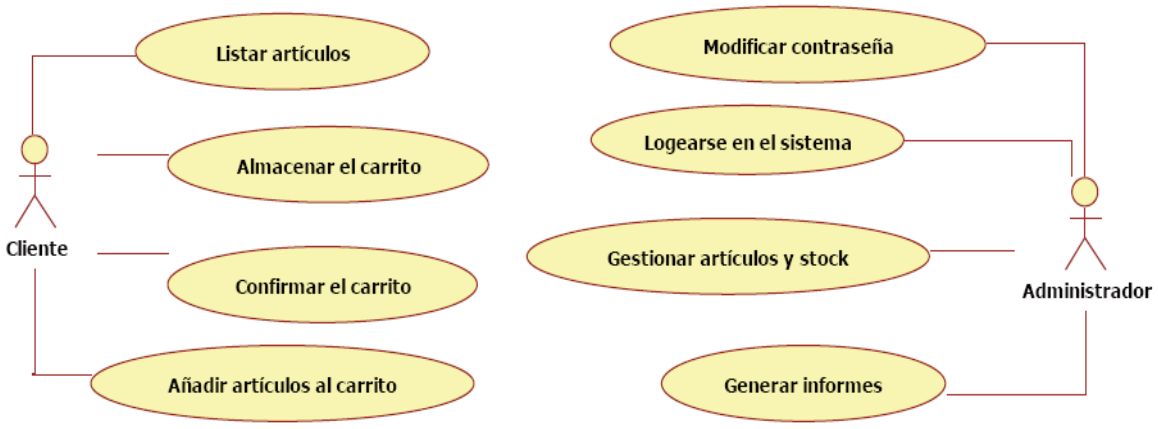
Vamos a desarrollar un caso práctico desde la captura de requisitos hasta el inicio de la codificación del sistema para ilustrar la aplicación de la metodología RUP.

En primer lugar realizaremos la captura de requisitos con ayuda de reuniones con el cliente o, en el caso de que exista, de documentación o aplicaciones preexistentes.

Por ejemplo el cliente podría plantear los siguientes requisitos funcionales:

- Funcionalidad propia del cliente:
    * Listar los artículos con su stock
    * Añadir artículos al carrito de la compra
    * Confirmar el carrito de la compra
    * Almacenar el carrito de la compra en el sistema para confirmarlo posteriormente
    * Efectuar el pago de la compra (que interactúa con una pasarela bancaria).
- Funcionalidad propia del administrador:
    * Logearse en el sistema
    * Gestionar artículos y stock
    *- Generar informes sobre ventas
    * Modificar contraseña del administrador

Un diagrama UML que incluya la funcionalidad especificada podría ser el siguiente:



La metodología RUP admite múltiples variantes con respecto a la introducción de requisitos:

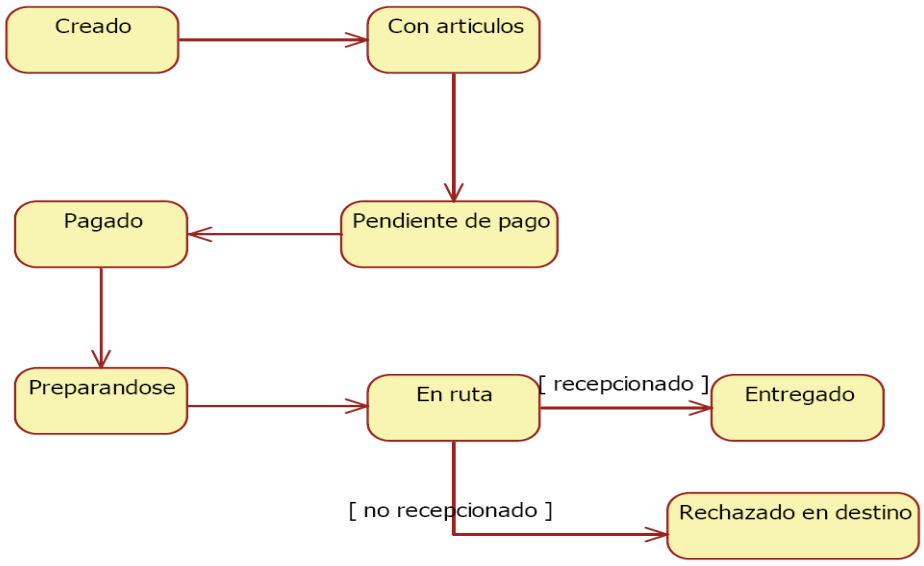
- Por ejemplo 'Logearse en el sistema' se puede considerar un prerrequisito de varios casos de uso, en vez de un caso de uso en sí mismo.

- Gestionar artículos y stock se podría dividir en dos casos de uso o dejar en un solo caso de uso que se explota en casos de uso más detallados.

- Podemos establecer dependencias entre casos de uso y además estereotiparlas para particular las propiedades de esa dependencia.

Una vez realizado el diagrama de C.U. para el caso práctico deberíamos elaborar la ficha de C.U. para todos los C.U. indicados en el diagrama y para los C.U. que estén en diagramas más detallados.

En esta fase del proyecto y en las posteriores podemos elaborar otros muchos diagramas de soporte que consideremos útiles para clarificar conceptos de negocio, como por ejemplo el ciclo de vida de un pedido.



Análisis (análisis funcional)

El análisis consiste en expandir la información elaborada en la definición de los C.U. y estructurar las clases de análisis siguiendo criterios funcionales con el fin de modelar la lógica del negocio.

Habitualmente en el análisis no se introducen conceptos específicos de una determinada arquitectura de Software. Es decir, el análisis debería ser independiente de Java, .NET, PHP o cualquier otra tecnología.

Las clases de análisis son clases UML que han sido estereotipadas para representar un determinado elemento del sistema. Hay tres estereotipos predefinidos:

- Boundary (frontera): establece el punto de interacción entre el actor y el interior del S.I.

- Control (control): identifica un elemento de gestión del sistema que permite el acceso a los datos del sistema a través de fronteras.

- Entity (entidad): es un almacén persistente de información, habitualmente una BBDD o una tabla de una BBDD.

Como se puede apreciar en la siguiente imagen, los estereotipos modifican la representación visual de las clases de análisis (algunas herramientas no modifican dicha representación o permiten ambas representaciones).

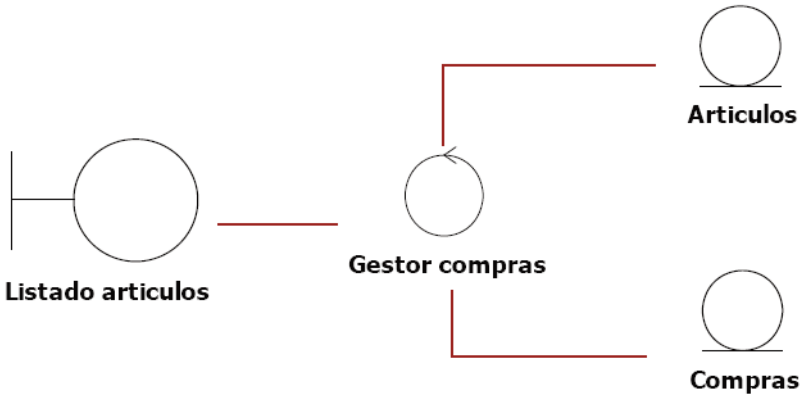
Modelado de análisis de las clases vinculadas al proceso de login.
De izq. a dcha. clases estereotipadas como boundary, control y entity.

Análisis del caso práctico

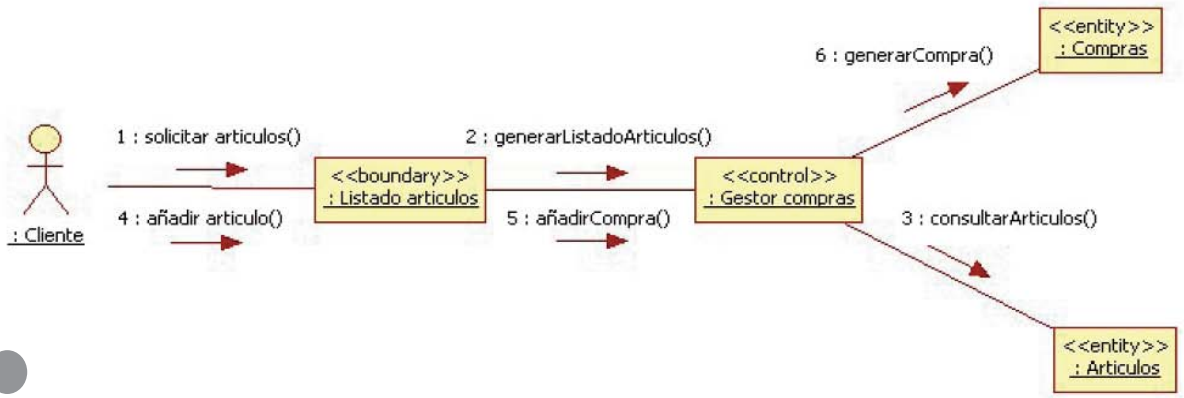
Para nuestra tienda online, debemos detallar los C.U. obtenidos en la captura de requisitos, en este punto ya deberíamos tener la ficha de C.U. detallada para todos los C.U. del sistema.

RUP propone que para este tipo de tarea desarrollemos un diagrama de clases de análisis y simultánea o posteriormente un diagrama de colaboración por cada flujo de cada C.U.

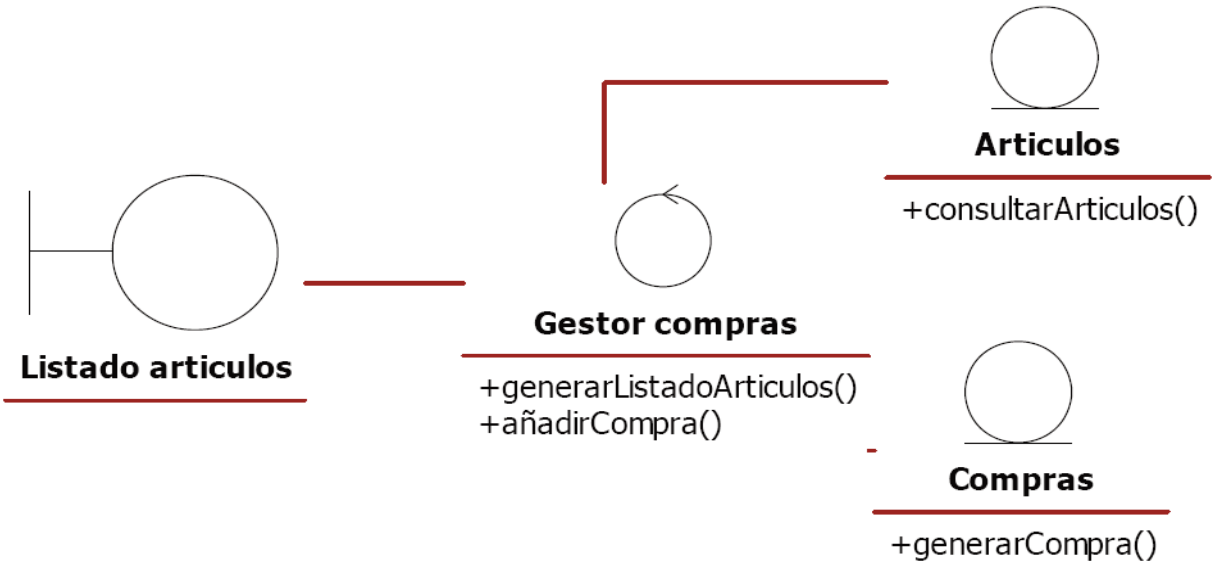
Por ejemplo si desarrollamos el C.U. 'Añadir al carrito' el diagrama de clases de análisis seria el siguiente



Y el diagrama de colaboración para el flujo primario seria el siguiente:



A continuación realizaríamos diagramas de colaboración o de secuencia para los flujos primarios y secundarios de todos los C.U. existentes. Muchos de los mensajes entre objetos se traducirán en métodos sobre las clases de análisis (en el diagrama de la parte superior serían por ejemplo los mensajes 2, 3, 5 y 6).



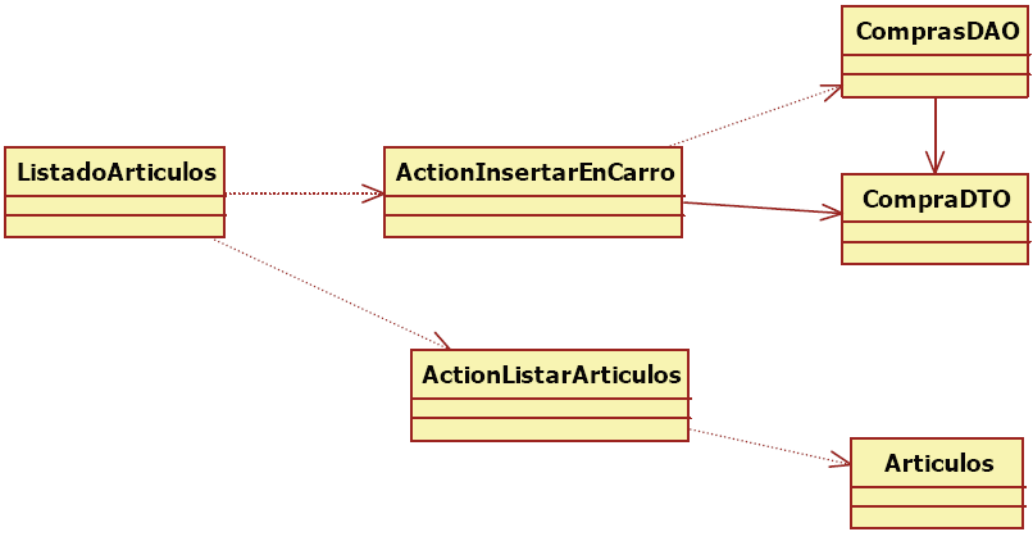
Al seguir poblando de métodos las clases de análisis y obteniendo otras nuevas estamos definiendo el elemento fundamental del análisis, las clases de análisis.

Diseño (análisis orgánico)

El diseño consiste en expandir la información elaborada en el análisis del sistema y generar diagramas que representen la misma información del análisis pero detallada en mayor medida y vinculada a la arquitectura concreta que se vaya a usar en el sistema.

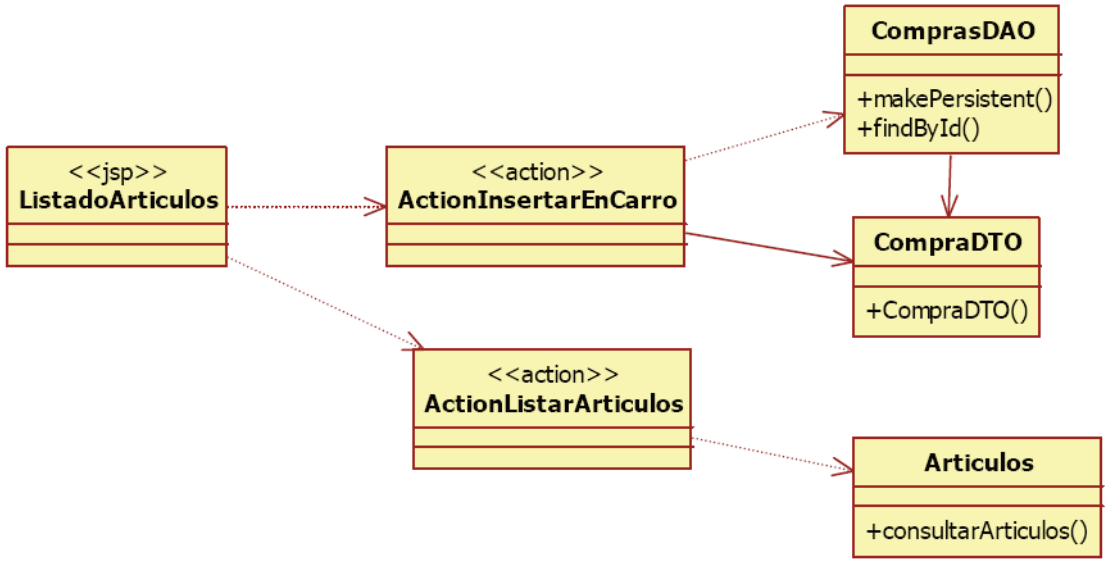
Diseño del caso práctico

Para el diagrama de clases de análisis que realizamos anteriormente tendremos ahora que generar diagramas de clases de diseño, es decir diagramas de clases específicos de la arquitectura desarrollada.



Por ejemplo en este diagrama podemos apreciar cómo hemos 'realizado' (expandido) el GestorCompras del análisis en un par de clases de diseño para adaptarlo a un determinado framework (Struts de Java). Podemos apreciar también en la parte derecha del diagrama el uso de dos patrones de diseño, el DAO y el DTO.

Corresponde al diseñador poblar estas clases con métodos de manera que reflejen toda la funcionalidad obtenida en el análisis. Podemos estereotipar algunas clases si consideramos que es pertinente para detallar en mayor medida sus características.



Dependiendo de la capacidad y experiencia de los análisis y diseñadores el sistema será más o menos modular, reutilizable, mantenible, estructurado y por lo tanto, económico en el corto, medio y largo plazo.

Las decisiones de análisis y diseño que hayamos tomado afectarán a toda la vida del sistema.

Implementación

La implementación es la fase del proyecto donde se transforman en código los diagramas generados durante las fases anteriores.

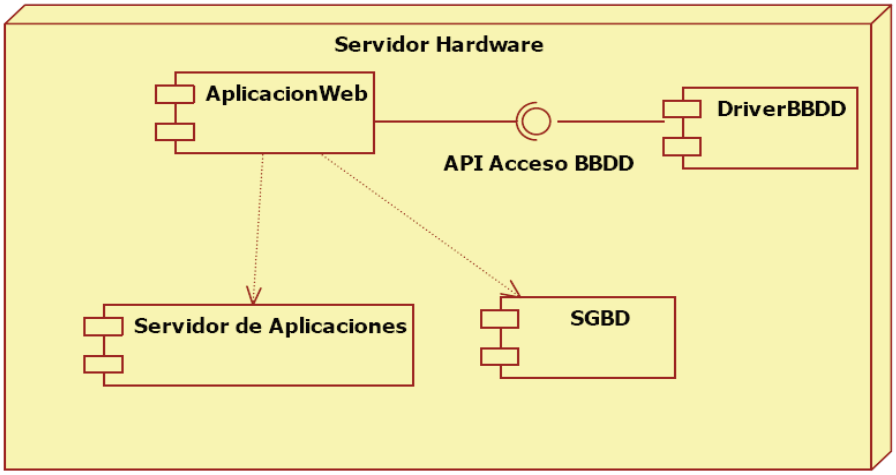
Además se pueden emplear diagramas de componentes y de despliegue para indicar los elementos de software y hardware que van a integrar el sistema. Incluso se pueden combinar cuando el número de nodos de hardware

En la mayoría de sistemas el mayor volumen de trabajo con UML no se encuentra en la implementación, sino que se usan herramientas específicas de cada arquitectura (habitualmente java o .net).

Implementación del caso práctico

Existen herramientas que a partir de un diagrama de clases de diseño nos generan el esqueleto de las clases para codificar los métodos de las mismas. Estas herramientas sin embargo no se suelen adaptar a las necesidades concretas de cada proyecto.

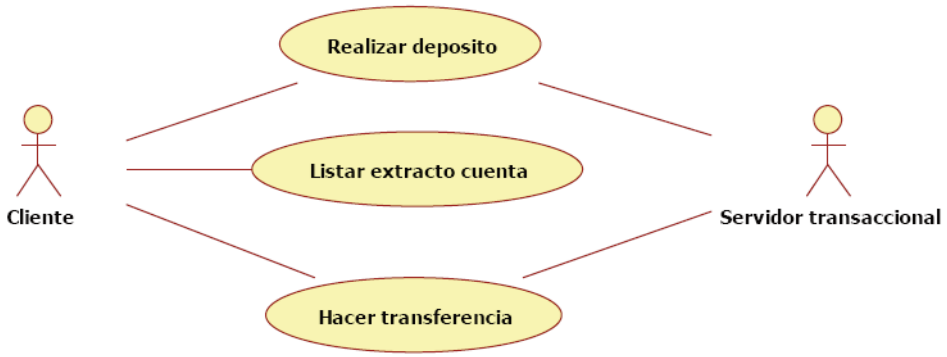
Un diagrama de componentes y otro u otros de despliegue suelen ayudar en esta fase del proyecto, a veces si el número de nodos de hardware es reducido se combinan en uno solo como es el caso de la tienda online.



Resumen de la aplicación práctica de RUP

Aunque lo hemos visto en el desarrollo del caso práctico vamos a resumir los pasos principales de la metodología RUP.

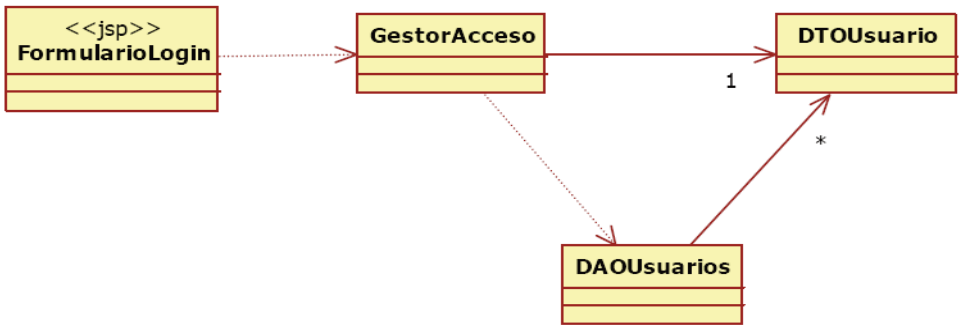
En un proyecto que siga la metodología RUP, el punto de partida son los C.U. que reflejan en detalle los requisitos del sistema.



Cada C.U. se detallará en una ficha con sus características, una de ellas es el flujo primario. El flujo primario se empleará para elaborar un diagrama de clases de análisis y el diagrama de comunicación o secuencia que modele el flujo primario del C.U. en cuestión.

En teoría también deberían modelarse en diagramas de secuencia o comunicación los flujos alternativos o secundarios, pero por restricciones de tiempo se suelen obviar muchos flujos secundarios.

Una vez elaborados los diagramas de clases de análisis y de secuencia o de colaboración que el analista considere oportunos podemos pasar al diseño, elaborando diagramas de clases de diseño que reflejarán elementos específicos de la arquitectura seleccionada.



Los diagramas de diseño habitualmente se emplean para generar la estructura de clases con los atributos y métodos que formarán el esqueleto del sistema que los programadores deberán codificar. Las herramientas de modelado suelen aportar opciones para generar esos esqueletos de forma automática.

Muchas empresas disponen de métricas especificas que son variaciones de RUP o versiones simplificadas o más complejas que varia el tipo de entregables generados.

Herramientas existentes

Para modelado con UML se pueden emplear entre otras las siguientes herramientas:

- StarUML: herramienta de software libre con soporte de UML 2.0. Los diagramas de este curso han sido desarrollados con esta herramienta.
- Rational Rose: una herramienta desarrollada por Rational antes de que fuera comprado por IBM, desactualizada desde 2003 pero todavía en uso en algunas empresas.
- Rational IBM Software Modeller: herramienta de IBM con soporte para UML 2.0.
- MyEclipse: versión de pago del IDE Eclipse con un plug-in para algunos diagramas de UML.