ÍNDICE

Unidad didáctica 5

LENGUAJE UNIFICADO DE MODELADO (UML)

OBJETIVOS:

Esta unidad didáctica tiene como objetivo introducir al alumno en el estudio del lenguaje unificado de modelado (UML) usado en el análisis y diseño orientado a objetos.

INTRODUCCIÓN

UML es una **notación estándar para desarrollo de sistemas** usando el enfoque orientado a objetos.

UML **comenzó en 1994** como un esfuerzo de Grady Booch y James Rumbaugh para combinar sus metodologías definiendo una notación estándar para ellas. Después, en 1995, Ivar Jacobson (técnica OOSE que es un *software* de ingeniería orientado a objetos), incluyendo casos de uso se unió al equipo.

Paralelamente a este esfuerzo, el Object Management Group (OMG, www.omg.org) hizo una llamada para propuestas de notación y metamodelo orientado a objetos. En Enero de 1997 UML fue entregado al OMG en respuesta a su convocatoria. Existían otras propuestas remitidas, pero al final, se formó un solo equipo en torno a UML.

UML **es solo una notación, no dicta estándares para el proceso de desarrollo**. Sin embargo, UML condiciona dicho proceso de desarrollo al establecer los diagramas e información asociada que debe representarse.

Los diagramas incluidos en UML, agrupados según su ámbito de aplicación, son:

- **Análisis en el contexto organizacional:** diagramas de casos de uso.
- **Análisis y diseño desde la perspectiva estática:** diagrama de clase y diagrama de objetos.
- **Análisis y diseño desde la perspectiva dinámica:** diagrama de secuencia, diagrama de colaboración y diagrama de estados.
- **Implementación:** diagrama de componentes y diagrama de despliegue.

5.1. CARACTERÍSTICAS.

UML tiene origen en **Simula 67**, que es un lenguaje diseñado para hacer simulaciones, y fueron refinados más tarde en el conocido **lenguaje de programación orientado a objetos Smalltalk**. Supone una nueva filosofía frente a la programación estructurada porque los datos y el código se agrupan en objetos y los datos y las instrucciones se agrupan en clases.

Las **ventajas** más importantes del **UML** son:

- Adecuada para **entornos de BBDD** (bases de datos).
- Adecuada para **prototipos y simulación**.
- Adecuada para **tratamientos de interfaces de usuario**.
- **Costos del HW** (*hardware*) decrecen.
- **Portabilidad** de los objetos.
- **Reusabilidad del código** (abstracción y herencia).

Los **inconvenientes** más importantes del **UML** son:

- **Curvas de aprendizaje** son largas.
- **Dependencia** del lenguaje.
- **Determinación** de las clases.
- Existe ***perfomance*** (en un sistema donde todo es un objeto y toda interacción es mediante mensajes).

A continuación, se incluyen una serie de **definiciones** necesarias para entender lo que es el UML y su proceso posterior:

Abstracción se basa en las características principales de un objeto las cuales le hacen diferente de otro.

Encapsulación es el proceso mediante el cual el estado de un objeto no puede ser manipulado por acción directa sobre los datos miembros.

Clase es un plantilla que permite la creación de objetos y contiene la descripción de las características comunes de esos objetos.

Mensaje es el medio por el que los objetos se comunican entre sí (es una petición de un objeto a otro al que le solicita ejecutar uno de sus métodos y donde el objeto que envía es el emisor y el que recibe el receptor).

Herencia consiste en reutilizar código que permite a los programadores crear nuevas clases a partir de clases existentes.

Polimorfismo es la posibilidad de que una misma operación se puede comportar de distintas formas en diferentes objetos.

En este *blog* se amplía más información sobre las características de UML: http://daliags-programacion.blogspot.com.es/2011/10/caracteristicas-del-uml.html.

La filosofía del UML es distinta a la programación estructurada, ya que todo se basa en los objetos y en la conocida "programación orientada a objetos".

5.2. VERSIONES.

Los antecedentes de UML se sitúan en la década de los 90 con distintos estándares para modelado de *software*. No obstante, podemos hablar de dos grandes versiones aunque se espera que aparezca una tercera versión.

Seguidamente, se enumeran las **versiones de UML** que se han ido usando a lo largo de la historia:

- **UML 1.X** (comprende UML 1.1, 1.2, 1.3, 1.4, 1.5): desde finales de los 90 se empezó a trabajar con el estándar UML. En los años sucesivos fueron apareciendo nuevas versiones que introducían mejoras o ampliaban a las anteriores.
- **UML 2.X** (comprende UML 2.1 hasta UML 2.5, 2.6, etc.): en torno a 2005 se difundió una nueva versión de UML a la que podemos denominar UML 2.X. Comprenden varias revisiones.
- **UML 3.X:** evolución que se espera para UML 2.X.

Hay que tener en cuenta que UML es un conjunto muy amplio de normas. Prácticamente nadie las conoce todas. Según la empresa o universidad, institución o centro de trabajo, se usan determinados programas para crear diagramas y se conocen ciertas partes de UML, pero no el conjunto de UML.

¿Qué versión usar? Para generar diagramas UML se usan programas informáticos. Usa un programa actualizado, pero no te preocupes en exceso por qué versión de UML usar, lo importante es que en tu grupo de trabajo o personas a las que se les vaya a enviar documentación sobre un proyecto *software* sepan interpretar lo que se les envía.

A nivel profesional no se le presta demasiada atención a que se cumpla estrictamente con las normas de una determinada versión de UML, sino a que los esquemas estén bien construidos y razonados.

UML tiene dos versiones actuales, UML1.X y UML 2.X, y se espera que haya una tercera versión, UML 3.X. Pero realmente lo que nos interesa es saber interpretar la versión de UML que estemos usando.

Para el que desee ampliar información sobre UML, incluimos **dos enlaces que os aportarán información muy interesante**:

- En el siguiente *blog* se describen las versiones de UML más conocidas: http://bit.ly/LKpjDy.
- La siguiente presentación expone muy bien todo lo referente a las versiones de UML: http://bit.ly/1gIGUJZ.

5.3. DIAGRAMAS UML.

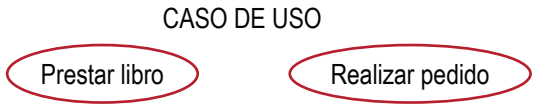
Son una representación gráfica de un conjunto de elementos y las conexiones entre ellos.

Veremos tres **tipos de diagramas UML**: diagramas de caso de uso, de clases y de secuencia.

Diagramas de caso de uso

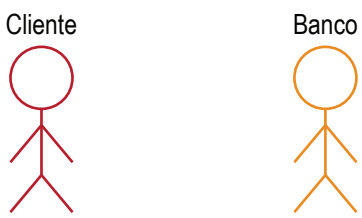
Su función principal es **describir el comportamiento del *software* a desarrollar**. Sus elementos principales son el caso de uso, los actores, las relaciones y la descripción.

Un **caso de uso** es un conjunto de acciones que ejecuta un sistema. Se representa mediante una elipse de borde continuo (un "uso típico del sistema", como por ejemplo una feria de subastas).



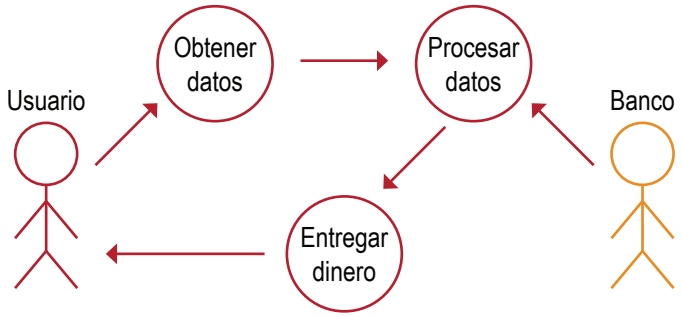
Describe bajo la forma de acciones y reacciones el comportamiento de un sistema desde el punto de vista de los usuarios. Describe la funcionalidad del futuro sistema independiente de la implementación (representa un requisito funcional del sistema).

Un **actor** es un elemento externo al sistema con el que interactúa, es decir, es un rol que desempeña un "usuario" respecto de un caso de uso con el que interactúa (base de datos, impresora, pantalla, una persona, un conjunto de personas, un sistema *hardware*, un reloj, etc).



Las **relaciones** son asociación, dependencia y generalización y la **descripción** son las plantillas textuales para cada caso de uso.

En el siguiente ejemplo se describe el **caso de uso de un sencillo sistema de cajeros**:



- El sistema es un rectángulo.
- Actores: son los agentes situados fuera del sistema y que interaccionan con él.
- Casos de uso: son procesos o funcionalidades que el sistema debe manejar u ofrecer.

Un diagrama de casos de uso consiste en realizar un análisis funcional de una aplicación, es decir, muestra las distintas funciones.

Diagramas de clases

Representa las **clases** del sistema **y las relaciones** entre ellas.

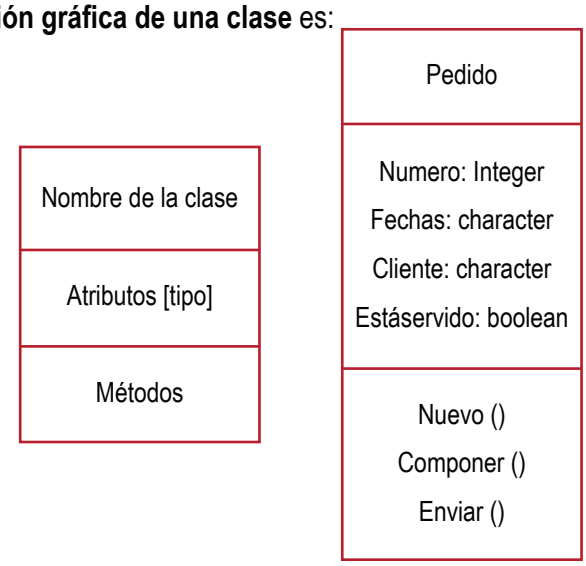
Hay **dos elementos principales**: las clases (que son un conjunto de objetos con un comportamiento común, es decir, con atributos y métodos) y las relaciones (asociación, dependencias, generalización y herencia).

Una **clase representa un elemento conceptual** como por ejemplo la clase "pedido", la clase "cliente" o la clase "empresa".

Los **pasos** para la **construcción del modelo orientado a objetos** mediante el diagrama de clases a partir de la descripción del problema son:

- Identificar objetos.
- Identificar clases.
- Identificar asociaciones entre clases.
- Identificar atributos de objetos y de enlace.
- Organizar y simplificar las clases con la herencia.
- Iterar y refinar el modelo.

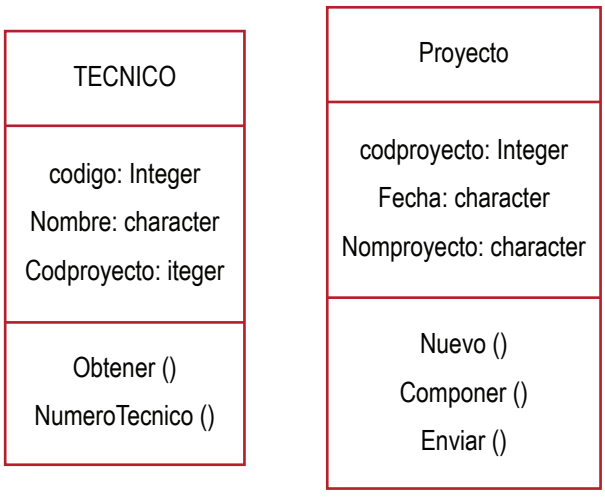
La **representación gráfica de una clase** es:



Habitualmente se usan los métodos get(), que quieren decir “obtener”, y el método set(), que quiere decir “poner, colocar”.

En el siguiente ejemplo estudiaremos la **relación entre técnico y proyecto**.

- Es una relación de **1 a n** de proyecto a técnico (porque en un proyecto puede haber 1 o más técnicos).
- Es una relación de **0 a n** de técnico a proyecto (porque cada técnico puede no estar asignado a ningún proyecto, a uno o a varios).



En un diagrama de clase se representan las clases junto con sus relaciones, así como que para cada clase hay que indicar sus métodos y atributos y que las relaciones entre las clases pueden ser de asociación, de dependencia y de generalización o herencia.

Diagramas de secuencia o de escenario

Un diagrama de **secuencia representa la interacción entre las clases del modelo** de estructuras estáticas, ordenadas temporalmente. Éste se lee de izquierda a derecha

y de arriba abajo. Normalmente cada caso de uso tiene asociado un diagrama de secuencia. Se usan principalmente cuando se trata de sistemas de tiempo real.

Cada escenario **representa el ciclo de vida de los objetos de una clase** permitiendo mostrar su relación con los objetos de otra clase, con los que se mantienen relaciones de agregación, relación e instanciación a través de la ejecución de sus respectivos métodos, en este modelo se representan los aspectos más importantes del ciclo de vida de cada objeto en cuanto a su relación con otros.

En un **diagrama de secuencia** intervienen los siguientes **elementos**:

- Las clases.
- Los actores.
- Los mensajes u operaciones, que es la **comunicación entre actores y clases**. Pueden ser de tres tipos:

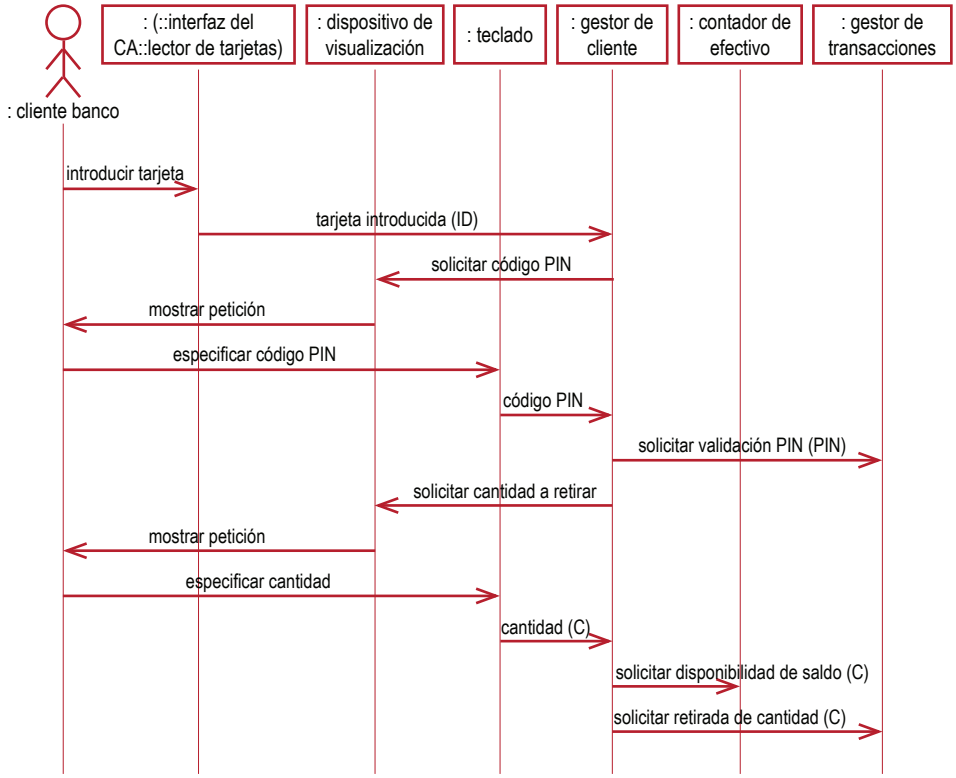
  - **Síncrono:** es el más utilizado, y significa que el expedidor del mensaje espera que la activación del método mencionado por el destinatario finalice antes de comunicar su actividad. Se representa por una flecha con la punta en negrilla.
  - **Asíncrono:** el expedidor no espera el término de la actividad invocada por el destinatario. Se representa por una flecha con punta normal.
  - **Retorno:** llama a un método que devuelven un resultado. Se representa por una flecha de trazo discontinuo.

- Muestra una visión gráfica de la interacción de los objetos con una referencia temporal, muestra los objetos participantes en la interacción y la secuencia de mensajes intercambiados.
- Los mensajes se representan como flechas dirigidas del objeto que los emite al objeto que los recibe, se etiquetan con el nombre del método invocado.
- Un retorno se representa con una flecha discontinua.
- El control de ejecución por un objeto se representa por una línea doble, este control se puede ver con diferente numerabilidad, representado por el anidamiento de líneas dobles.

Un diagrama de secuencia consiste en mostrar cómo las clases se interelacionan entre sí de izquierda a derecha y de arriba abajo.

En el siguiente *blog* se amplia más sobre todos los tipos de diagramas UML que existen, aunque hay que hacer hincapié que no todos se usan de manera habitual en las empresas: http://bit.ly/1IycUCy.

Siguiendo con el **ejemplo del sistema de cajeros**, a continuación describimos cómo sería el diagrama de secuencia o escenarios.



Ésta sería una posible solución al diagrama de secuencia o escenarios.

5.4. UTILIZACIÓN EN METODOLOGÍAS DE DESARROLLO ORIENTADO A OBJETOS.

En este apartado se explicará cómo se utiliza el lenguaje de modelo unificado (UML) en el desarrollo orientado a objetos para lo cual aplicaremos los tres tipos de diagramas UML que hemos conocido anteriormente, es decir, los **diagramas de caso de uso, los diagramas de clases y los diagramas de secuencia**.

A continuación se mostrarán unos **ejemplos** que nos ayudarán a ver el uso del UML en las metodologías de desarrollo orientado a objetos, para que de este modo se comprendan mejor los tres tipos de diagramas.

En el siguiente ejemplo queremos realizar un diagrama de caso de uso, de clases y de secuencia de un sencillo **"sistema de gestión de almacenes"** donde solo **aparecen un cliente y un proveedor**.

Diagrama de caso de uso

En este diagrama los **actores** son los clientes y proveedores así como los **casos de uso** serían encargar pedido, dar factura, recoger pedido y encargo.

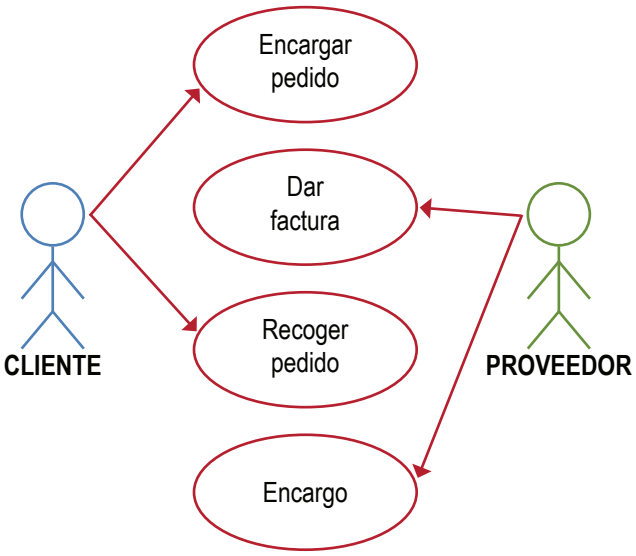


Diagrama de clases

En este diagrama las clases son clientes y proveedores, y la relación entre ellos sería de N a N que significa que un cliente puede tener muchos proveedores y que a la vez un proveedor puede tener muchos clientes.

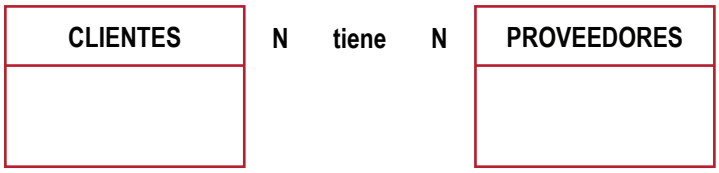
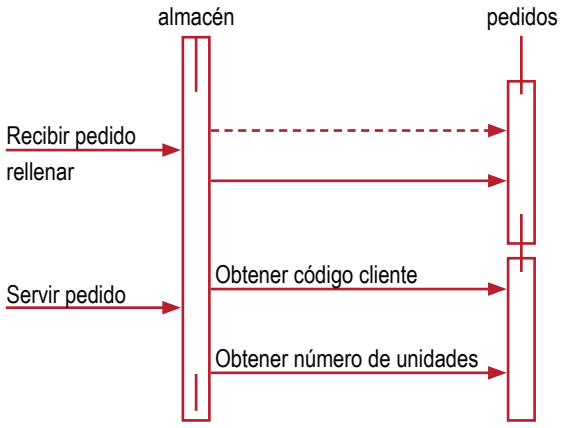


Diagrama de escenario o secuencia

Teniendo en cuenta el escenario "recibir y servir pedidos" la secuencia de acciones se reflejan en el siguiente diagrama:



Ésta sería una posible solución al diagrama de casos de uso, de clases y de secuencia o escenarios. No existe una solución única.

En el siguiente ejemplo se desea realizar un diagrama de caso de uso, de clases y de secuencia de un sencillo **"sistema de control de las notas"** de un alumnos por parte de un profesor donde solo existen alumnos y profesores.

Diagrama de caso de uso

En este diagrama los actores son los alumnos y el profesor, y los casos de uso serían matricularse, introducir notas y publicar notas.

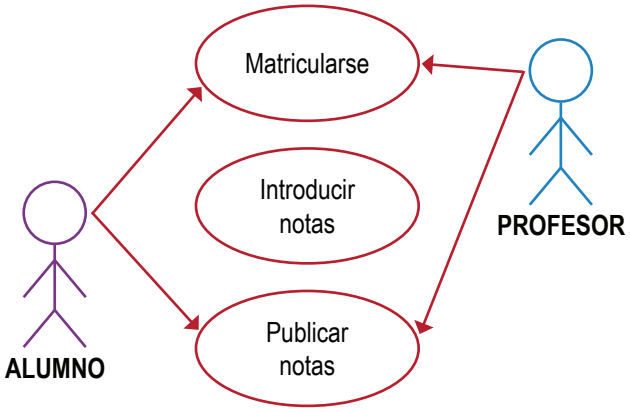
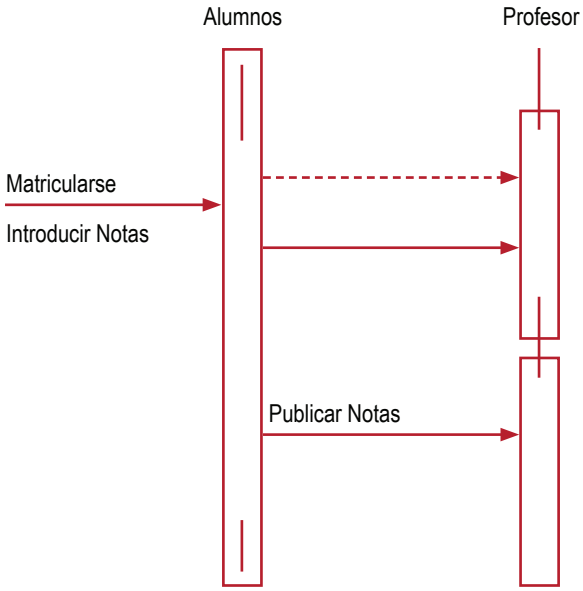


Diagrama de clases

En este diagrama las clases son alumno y profesor, y la relación entre ellos sería de n a n que significa que un alumno puede tener muchos profesores impartiendoles clase y que a la vez un profesor puede impartir clase a muchos alumnos.

Diagrama de escenario o secuencia



Ésta sería una posible solución al diagrama de casos de uso, de clases y de secuencia o escenarios. No hay una solución única.

1. Para hacer un diagrama de caso de uso hay que pensar cuáles son los actores y los casos de uso.
2. Para hacer un diagrama de clase hay que pensar en cuáles son las clases y sus relaciones.
3. Para resolver un diagrama de secuencia, hay que pensar en las posibles secuencias de acciones que se van a realizar a lo largo del proceso de desarrollo del *software*.

5.5. HERRAMIENTAS CASE CON SOPORTE UML.

En este apartado, veremos varios ejemplos de las herramientas Case más usadas en la actualidad, tales como **Rational Rose**, **ERwin**, **EasyCASE**, **PowerDesigner** y **System Arquitect**.

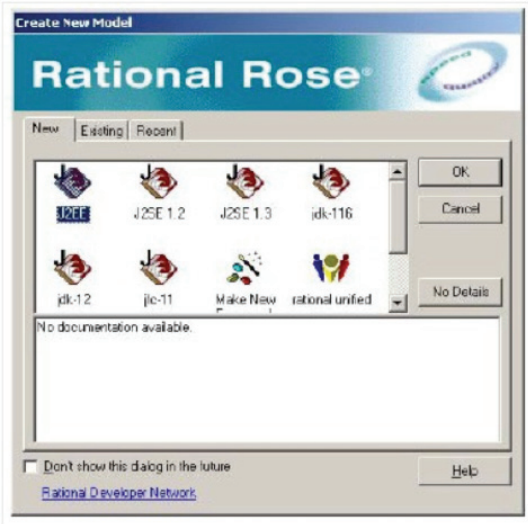
Rational Rose

¿Qué es Rational Rose? Es una herramienta para "modelado visual", que forma parte de un conjunto más amplio de herramientas que juntas cubren todo el ciclo de vida del desarrollo de *software*.

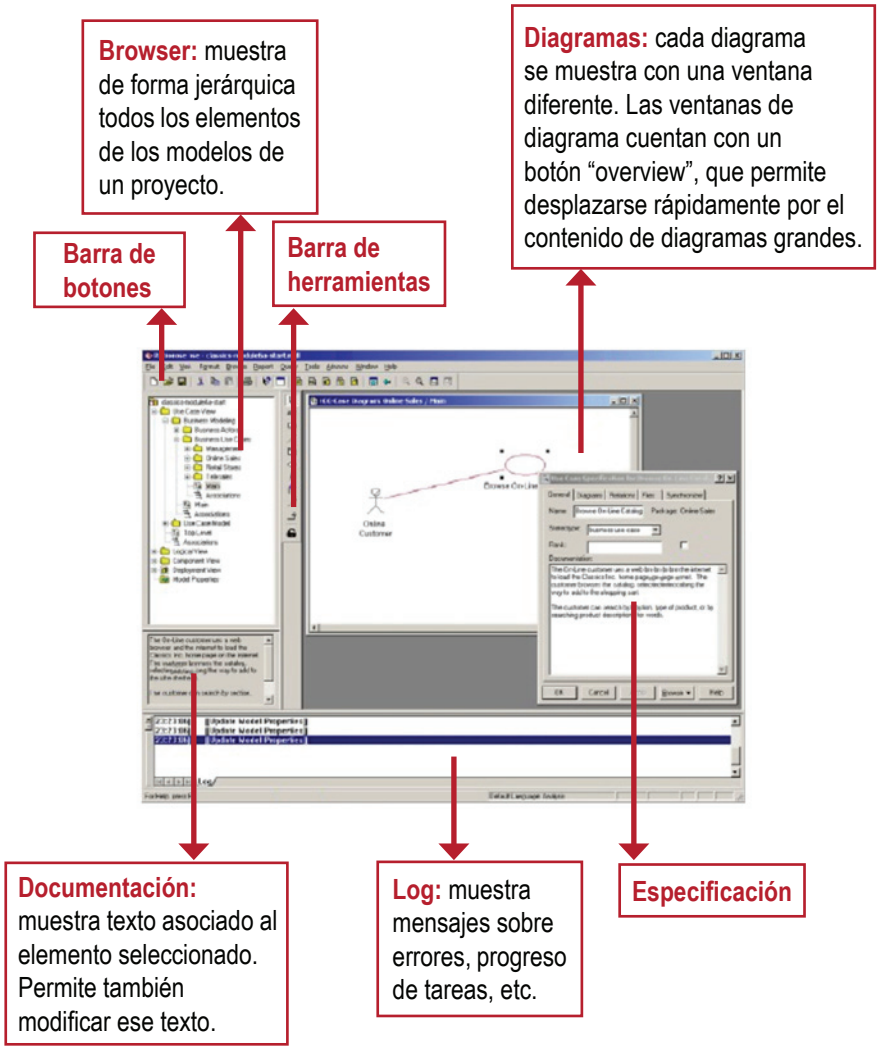
Rational Rose permite completar una gran parte de las disciplinas (flujos fundamentales) del proceso unificado de Rational (RUP), en concreto:

- Modelado del negocio.
- Captura de requisitos (parcial).
- Análisis y diseño (completo).
- Implementación (como ayuda).
- Control de cambios y gestión de configuración (parte).

La primera pantalla que aparece es un selector de *framework* (esqueleto). Este cuadro de diálogo permite elegir modelos que contienen elementos predefinidos para distintos tipos de proyectos.



Al cancelar el anterior cuadro de diálogo aparece la ventana principal de Rose. Esta ventana **tiene los siguientes componentes**: **"Browser"**; **"Documentación"**; **"Log"**; **"Diagramas"**.



Como se puede comprobar por la explicación anterior, los diagramas son **vistas gráficas del modelo**. Rose mantiene automáticamente la consistencia entre los diagramas y las especificaciones correspondientes: si se modifica el diagrama se cambia la especificación, y viceversa.

Tras este primer acercamiento a la herramienta Case Rational Rose, ahora es el momento de saber **qué diagramas son soportados por Rose**.

- De clases.
- De casos de uso.
- De colaboración.
- De secuencia.
- De componentes.
- De estados.
- De actividad.
- De despliegue.

Ya conocemos los diagramas soportados por Rational Rose, de modo que en el siguiente listado veremos resumido cada uno de los siguientes diagramas:

- **Caso de uso:** esta vista define la interacción entre actores y casos de uso.
- **Diagramas principales:** casos de uso, colaboración, secuencia, actividad.
- **Lógica:** esta vista define fundamentalmente las clases del sistema y sus relaciones. Diagramas principales: clases, estados.
- **Componentes:** esta vista contiene información sobre ficheros, ejecutables y librerías del sistema. Diagrama de componentes.
- **Despliegue:** esta vista muestra la asignación de procesos al *hardware*.

La siguiente presentación nos muestra que es una herramienta Rational Rose y para que sirve: http://es.slideshare.net/vivi_jocadi/rational-rose.

Rational Rose es una herramienta que se usa para hacer un modelo visual de un proceso de desarrollo de *software* orientado a objetos y que constituyen todo el ciclo de vida del desarrollo de *software*.

ERwin

PLATINUM ERwin es una herramienta de diseño de base de datos con la cual podemos **diseñar, generar y mantener aplicaciones de una manera muy eficaz y productiva**. Además, ERwin permite visualizar la estructura, los elementos importantes, y optimizar el diseño de la base de datos desde un punto de vista lógico y físico generando automáticamente las tablas, y disparadores para los principales tipos de base de datos.

Las **características principales del ERwin** son las siguientes:

1. ERwin hace fácil el diseño de una base de datos.
2. ERwin automatiza el proceso de diseño de una manera inteligente.
3. Se mantienen las vistas de la base de datos como componentes integrados al modelo, permitiendo que los cambios en las tablas sean reflejados automáticamente en las vistas definidas.
4. ERwin establece una conexión entre una base de datos diseñada y una base de datos, permitiendo transferencia entre ambas y la aplicación de ingeniería reversa.
5. ERwin soporta principalmente bases de datos relacionales SQL y bases de datos que incluyen Oracle, Microsoft SQL Server, Sybase, DB2, e Informix.

El siguiente enlace nos describe más exhaustivamente qué es ERwin: http://bit.ly/LOTF7l.

ERwin es una herramienta de diseño de base de datos que sirve para **diseñar, generar y mantener aplicaciones** de una manera muy eficaz y productiva, así como para diseñar una base de datos de un modo fácil.

EasyCASE

EasyCASE Profesional **es una herramienta que se usa para generar esquemas de base de datos e ingeniería reversa,** de modo que podamos tener una solución comprensible para el diseño, consistencia y documentación del sistema en conjunto.

Las **principales caracterísicas** de esta herramienta son:

1. Permite automatizar las **fases de análisis y diseño** dentro del desarrollo de una aplicación.
2. Permite capturar los **detalles de diseño de un sistema** y comunicar las ideas gráficamente para que sean fáciles de ver y entender.
3. Posee **herramientas de corrección** avanzadas que permiten revisiones generales en poco tiempo.
4. Soporta una gama amplia de **metodologías estructuradas**, permitiendo escoger los métodos más apropiados para realizar las tareas.
5. Determina los **tipos de esquemas** según la metodología del proyecto seleccionada.
6. El verdadero poder de EasyCASE se encuentra en el **soporte comprensivo al modelado de datos, procesos y eventos**.

El siguiente vídeo nos muestra un tutorial sobre EasyCase:
http://www.youtube.com/watch?v=55atKxSS9uA.

EasyCASE es una herramienta que se usa para generar esquemas de base de datos e ingeniería reversa, automatizando las **fases de análisis y diseño** dentro del desarrollo de una aplicación, siendo su principal característica que tiene un **gran soporte en el modelado de datos, procesos y eventos**.

PowerDesigner

¿Qué es PowerDesigner? Es una *suite* de aplicaciones de la empresa **Powersoft** para la construcción, diseño y modelado de datos a través de diversas aplicaciones.

Es la herramienta para el análisis, diseño inteligente y construcción sólida de una base de datos y un desarrollo orientado a modelos de datos a nivel físico y conceptual.

Cuenta con los **siguientes productos**:

PowerDesigner ProcessAnalyst. Permite analizar el flujo de datos de toda la empresa, a través de los departamentos hasta el usuario final.

PowerDesigner DataArchitect. Provee a los diseñadores de las bases de datos una manera eficiente para la creación inteligente, depuración e ingeniería de inversa del modelado, tanto conceptual como físico, de los datos.

PowerDesigner AppModeler. Posibilita el diseño y ajuste de los componentes de objetos y datos en aplicaciones de uso común como PowerBuilder, Power++, Visual Basic y Delphi, ajustando el modelo de base de datos.

PowerDesigner WarehouseArchitect. Tiene un poderoso *datawarehousing* (almacén de datos) para el diseño e implementación de una base de datos y un soporte para bases de datos tradicionales DBMS y bases de datos en plataformas de sistemas analítico.

PowerDesigner MetaWorks. Permite ver y compartir la información del modelado de datos con una definición constante de objetos.

PowerDesigner Viewer. Crea reportes de los modelos físicos, conceptuales y procesos del modelado de la base de datos, así como permite generar reportes para internet en HTML.

Además de todas estas características, PowerDesigner **ofrece las posibilidades** de:

- **Soporte para tipos de datos abstractos.** PowerDesigner soporta la identificación de tipos de datos abstractos con ingeniería inversa de aplicaciones para Oracle8.
- **Soporte para usuarios de bases de datos.** Los usuarios de bases de datos pueden ser recogidos de una base de datos existente y luego almacenados en un modelo físico de datos. Ahora, es posible añadir nuevos usuarios y también asignar usuarios como propietarios y vistas.
- **Mayor selectividad en ingeniería inversa.** PowerDesigner permite seleccionar no solo las tablas que se desean cargar, sino todo tipo de objetos de la base de datos.
- **Cálculo del tamaño de las bases de datos.** Puede calcular y definir el tamaño definitivo de bases de datos de nuevo diseño y construcción, incluyendo tamaños detallados de índices y tablas.

El siguiente vídeo nos muestra un tutorial sobre PowerDesigner:
http://www.youtube.com/watch?v=Qr7DHhSKZwY.

PowerDesigner es una herramienta para el análisis, diseño inteligente y construcción sólida de una base de datos, así como un desarrollo orientado a modelos de datos a nivel físico y conceptual, a través de diversas aplicaciones.

System Arquitect

Las principales características que tiene System Architect son:

1. Tiene un único repositorio que **integra todas las herramientas, y metodologías usadas**.
2. Tiene un **control automático de diagramas y datos**, normalizaciones y balanceamiento entre diagramas "Padre e Hijo".

3. Traduce **modelos de entidades** en esquemas para base de datos, como por ejemplo Sybase, DB2, Oracle u Oracle 7, Ingress, SQL Server, RDB, XDB, Progress, Paradox, SQL Base, AS400, Interbase, OS/2, DBMS, Dbase 111, Informix, entre otros. Genera también Windows DDL, definiciones de datos para lenguaje C/C++ y estructuras de datos en Cobol.
4. Tiene **esquemas de seguridad e integridad** a través de contraseñas que posibilitan el acceso al sistema en diversos niveles.
5. Tiene un **módulo específico para Ingeniería Reversa desde las bases de datos** SQL más conocidas como Sybase, DB2, Informix, Oracle y SQL Server.
6. Tiene **muchas metodologías para diseño y análisis**, las cuales proporcionan amplio soporte para la construcción de los modelo conceptual, funcional y operacional.
7. Es una herramienta CASE de última generación, creada específicamente para la arquitectura **cliente/servidor**, donde los servidores hacen peticiones que son atendidas por el cliente.

El siguiente enlace nos muestra información sobre System Arquitect:
http://ibm.co/1e0cv1P.

System Architect es una herramienta CASE que tiene un único repositorio que **integra todas las herramientas y metodologías usadas,** que se creó específicamente para la arquitectura **cliente/servidor**.

5.6. RESUMEN.

En esta unidad, hemos podido ver todo el **proceso inicial de modelado unificado,** de modo que **lo principal que no hay que olvidar es**:

Abstracción: consiste en que los objetos son diferentes unos a otros según sus características.

Encapsulación: consiste en que estado de un objeto no se pueda manipular por acción directa sobre los datos miembros.

Clase: consiste en la posibilidad de poder crear objetos además de poder contener la descripción de las características comunes de esos objetos.

Mensaje: es el medio por el que los objetos se comunican entre sí, de modo que el emisor es el que envía el mensaje y el receptor es el que lo recibe.

Herencia: consiste en que los programadores puedan reutilizar código, así como crear nuevas clases a partir de clases existentes.

Polimorfismo: consiste en la posibilidad de que una misma operación se puede comportar de distintas formas en diferentes objetos.

Diagramas de caso de uso: está formado por los caso de uso (es un conjunto de pasos que se deben realizar dentro del diseño de un sistema) y los actores (son los elementos que interactúan con el sistema).

Diagramas de clases: está formado por las clases (que son un conjunto de objetos que tienen atributos y métodos) y las relaciones (que nos dicen la forma de conexión entre las clases).

Diagramas de secuencia o de escenario: representan el modo de relacionarse las clases entre sí y la secuencia que deben seguir ordenadas de izquierda a derecha y de arriba abajo.