Introducción a UML

Autor: Pere Martra

Introducción

UML es una especificación de notación orientada a objetos. Se basa en las anteriores especificaciones BOOCH, RUMBAUGH y COAD-YOURDON. Divide cada proyecto en un número de diagramas que representan las diferentes vistas del proyecto. Estos diagramas juntos son los que representa la arquitectura del proyecto.

Con UML nos debemos olvidar del protagonismo excesivo que se le da al diagrama de clases, este representa una parte importante del sistema, pero solo representa una vista estática, es decir muestra al sistema parado. Sabemos su estructura pero no sabemos que le sucede a sus diferentes partes cuando el sistema empieza a funcionar. UML introduce nuevos diagramas que representa una visión dinámica del sistema. Es decir, gracias al diseño de la parte dinámica del sistema podemos darnos cuenta en la fase de diseño de problemas de la estructura al propagar errores o de las partes que necesitan ser sincronizadas, así como del estado de cada una de las instancias en cada momento. El diagrama de clases continua siendo muy importante, pero se debe tener en cuenta que su representación es limitada, y que ayuda a diseñar un sistema robusto con partes reutilizables, pero no a solucionar problemas de propagación de mensajes ni de sincronización o recuperación ante estados de error. En resumen, un sistema debe estar bien diseñado, pero también debe funcionar bien.

UML también intenta solucionar el problema de propiedad de código que se da con los desarrolladores, al implementar un lenguaje de modelado común para todos los desarrollos se crea una documentación también común, que cualquier desarrollador con conocimientos de UML será capaz de entender, independientemente del lenguaje utilizado para el desarrollo.

UML es ahora un estándar, no existe otra especificación de diseño orientado a objetos, ya que es el resultado de las tres opciones existentes en el mercado. Su utilización es independiente del lenguaje de programación y de las características de los proyectos, ya que UML ha sido diseñado para modelar cualquier tipo de proyectos, tanto informáticos como de arquitectura, o de cualquier otro ramo.

UML permite la modificación de todos sus miembros mediante estereotipos y restricciones. Un estereotipo nos permite indicar especificaciones del lenguaje al que se refiere el diagrama de UML. Una restricción identifica un comportamiento forzado de una clase o relación, es decir mediante la restricción estamos forzando el comportamiento que debe tener el objeto al que se le aplica.

Diagramas. Vistazo general

La explicación se basará en los diagramas, en lugar de en vistas o anotación, ya que son estos la esencia de UML. Cada diagrama usa la anotación pertinente y la suma de estos diagramas crean las diferentes vistas. Las vistas existentes en UML son:

- Vista casos de uso: Se forma con los diagramas de casos de uso, colaboración, estados y actividades.
- Vista de diseño: Se forma con los diagramas de clases, objetos, colaboración, estados y actividades.
- Vista de procesos: Se forma con los diagramas de la vista de diseño. Recalcando las clases y objetos referentes a procesos.
- Vista de implementación: Se forma con los diagramas de componentes, colaboración, estados y actividades.
- Vista de despliegue: Se forma con los diagramas de despligue, interacción, estados y actividades.

Se Dispone de dos tipos diferentes de diagramas los que dan una vista estática del sistema y los que dan una visión dinámica. Los diagramas estaticos son:

- Diagrama de clases: muestra las clases, interfaces, colaboraciones y sus relaciones. Son los más comunes y dan una vista estática del proyecto.
- Diagrama de objetos: Es un diagrama de instancias de las clases mostradas en el diagrama de clases. Muestra las instancias y como se relacionan entre ellas. Se da una visión de casos reales.
- Diagrama de componentes: Muestran la organización de los componentes del sistema. Un componente se corresponde con una o varias clases, interfaces o colaboraciones.
- Diagrama de despliegue.: Muestra los nodos y sus relaciones. Un nodo es un conjunto de componentes. Se utiliza para reducir la complejidad de los diagramas de clases y componentes de un gran sistema. Sirve como resumen e índice.
- Diagrama de casos de uso: Muestran los casos de uso, actores y sus relaciones. Muestra quien puede hacer que y relaciones existen entre acciones(casos de uso). Son muy importantes para modelar y organizar el comportamiento del sistema.

Lo diagramas dinámicos son:

- Diagrama de secuencia, Diagrama de colaboración: Muestran a los diferentes objetos y las relaciones que pueden tener entre ellos, los mensajes que se envían entre ellos. Son dos diagramas diferentes, que se puede pasar de uno a otro sin perdida de información, pero que nos dan puntos de vista diferentes del sistema. En resumen, cualquiera de los dos es un Diagrama de Interacción.
- Diagrama de estados: muestra los estados, eventos, transiciones y actividades de los diferentes objetos. Son útiles en sistemas que reaccionen a eventos.
- Diagrama de actividades: Es un caso especial del diagrama de estados. Muestra el flujo entre los objetos. Se utilizan para modelar el funcionamiento del sistema y el flujo de control entre objetos.

Como podemos ver el número de diagramas es muy alto, en la mayoría de los casos excesivos, y UML permite definir solo los necesarios, ya que no todos son necesarios en todos los proyectos. En el documento se dará una breve explicación de todos, ampliándose para los más necesarios.

Diagramas recomendados

Los diagramas a representar dependerán del sistema a desarrollar, para ello se efectuan las siguientes recomendaciones dependiendo del sistema. Estas recomendaciones se deberán adaptar a las características de cada desarrollo, y seguramente será la practica lo que nos diga las cosas que echamos en falta o los diagramas que parecen ser menos necesarios.

- Aplicación monopuesto
    - Diagrama de casos de uso.
    - Diagrama de clases.

    - Diagrama de interacción.
- Aplicación monopuesto, con entrada de eventos:
    - Añadir: Diagrama de estados.
- Aplicación cliente servidor:
    - Añadir: Diagrama de despliegue y diagrama de componentes, dependiendo de la complejidad.
- Aplicación compleja distribuida:
    - Todos.

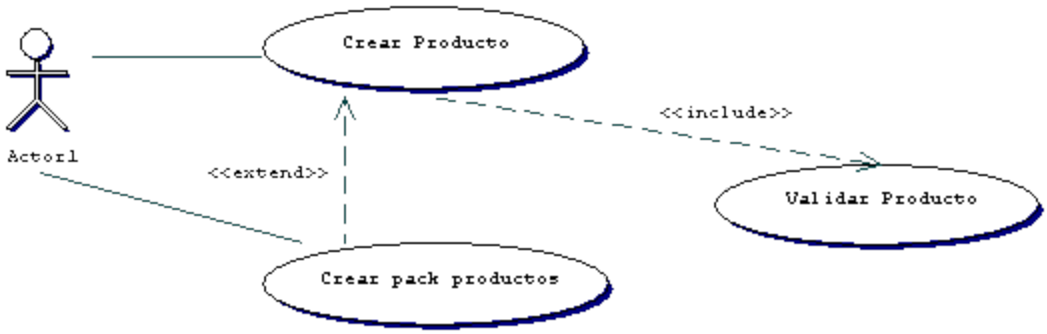
Así tenemos que para una aplicación sencilla debemos realizar entre tres y seis tipos de diagramas, y para una aplicación compleja unos nueve tipos. ¿Es esto demasiado trabajo? En un principio no lo parece, ya que el tiempo dedicado a la realización de los diagramas es proporcional al tamaño del producto a realizar, no entraremos en la discusión de que el tiempo de diseño no es tiempo perdido si no ganado. Para la mayoría de los casos tendremos suficiente con tres o cuatro diagramas. Debemos pensar que UML esta pensado para el modelado tanto de pequeños sistemas como de sistemas complejos, y debemos tener en cuenta que los sistemas complejos pueden estar compuestos por millones de líneas de código y ser realizados por equipos de centenares de programadores. Así que no debemos preocuparnos, el mayor de nuestros proyectos, desde el punto de vista de UML no deja de ser un proyecto mediano tirando a pequeño.

Diagrama de casos de uso

Se emplean para visualizar el comportamiento del sistema, una parte de el o de una sola clase. De forma que se pueda conocer como responde esa parte del sistema. El diagrama de uso es muy útil para definir como debería ser el comportamiento de una parte del sistema, ya que solo especifica como deben comportarse y no como están implementadas las partes que define. Por ello es un buen sistema de documentar partes del código que deban ser reutilizables por otros desarrolladores. El diagrama también puede ser utilizado para que los expertos de dominio se comuniquen con los informáticos sin llegar a niveles de complejidad. Un caso de uso especifica un requerimiento funcional, es decir indica esta parte debe hacer esto cuando pase esto.

En el diagrama nos encontramos con diferentes figuras que pueden mantener diversas relaciones entre ellas:

- Casos de uso: representado por una elipse, cada caso de uso contiene un nombre, que indique su funcionalidad. Los casos de uso pueden tener relaciones con otros caso de uso. Sus relaciones son:
    - Include: Representado por una flecha, en el diagrama de ejemplo podemos ver como un caso de uso, el de totalizar el coste incluye a dos casos de uso.
    - Extends: Una relación de una caso de Uso A hacia un caso de uso B indica que el caso de uso B implementa la funcionalidad del caso de uso A.
    - Generalization: Es la típica relación de herencia.
- Actores: se representan por un muñeco. Sus relaciones son:
    - Communicates: Comunica un actor con un caso de uso, o con otro actor.
- Parte del sistema (System boundary): Representado por un cuadro, identifica las diferentes partes del sistema y contiene los casos de uso que la forman.



En este grafico encontramos tres casos de usos Crear producto utiliza Validar producto, y Crear pack productos es una especialización de Crear productos.

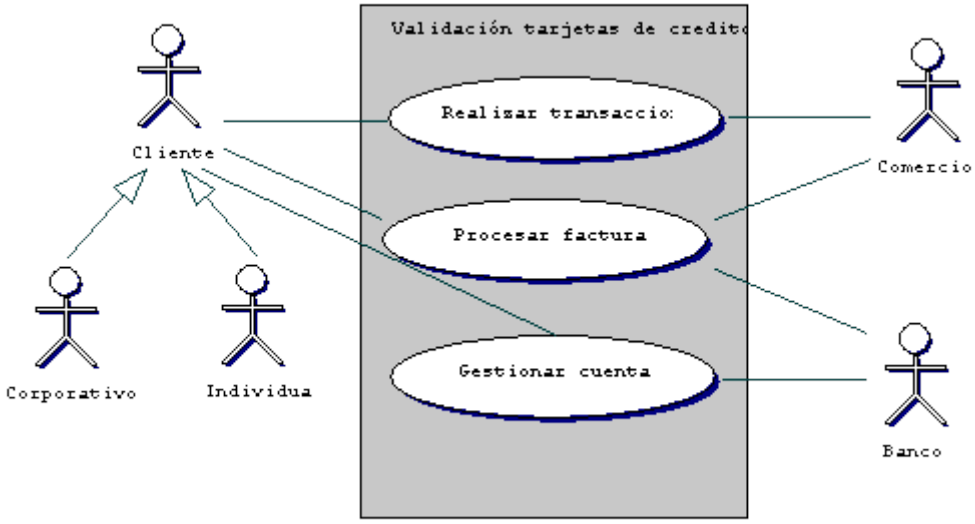
Podemos emplear el diagrama de dos formas diferentes, para modelar el contexto de un sistema, y para modelar los requisitos del sistema.

Modelado del contexto

Se debe modelar la relación del sistema con los elementos externos, ya que son estos elementos los que forman el contexto del sistema.

Los pasos a seguir son:

- Identificar los actores que interactúan con el sistema.
- Organizar a los actores.
- Especificar sus vías de comunicación con el sistema.



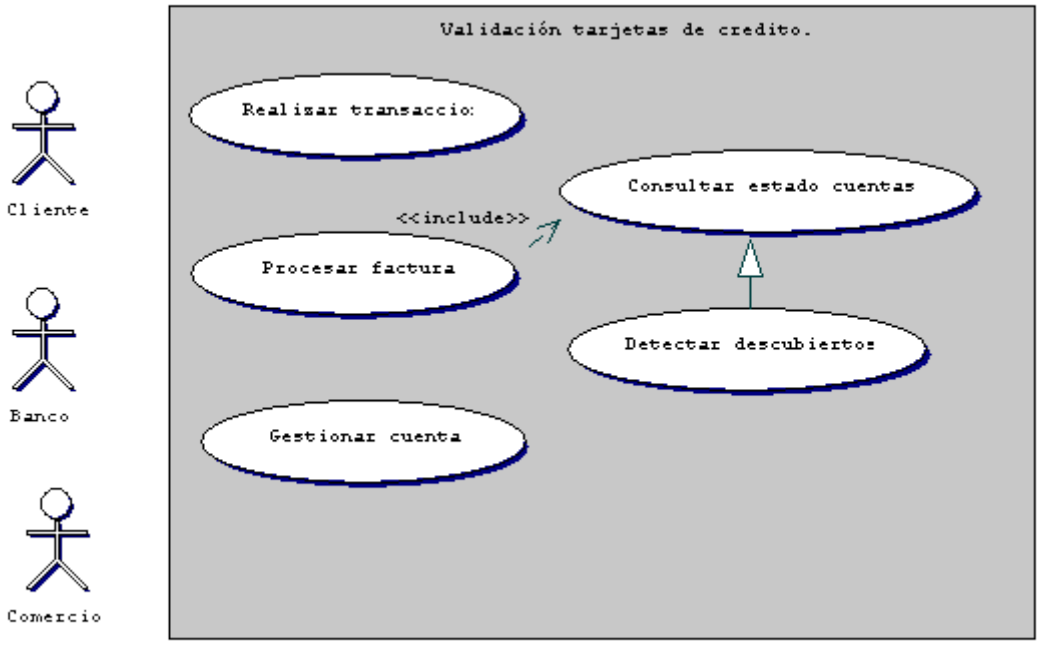
Modelado de requisitos

La función principal, o la mas conocida del diagrama de casos de uso es documentar los requisitos del sistema, o de una parte de el.

Los requisitos establecen un contrato entre el sistema y su exterior, definen lo que se espera que realice el sistema, sin definir su funcionamiento interno. Es el paso siguiente al modelado del contexto, no indica relaciones entre autores, tan solo indica cuales deben ser las funcionalidades (requisitos) del sistema. Se incorporan los casos de uso necesarios que no son visibles desde los usuarios del sistema.

Para modelar los requisitos es recomendable:

- Establecer su contexto, para lo que también podemos usar un diagrama de casos de uso.
- Identificar las necesidades de los elementos del contexto (Actores).
- Nombrar esas necesidades, y darles forma de caso de uso.
- Identificar que casos de uso pueden ser especializaciones de otros, o buscar especializaciones comunes para los casos de uso ya encontrados.



Como podemos ver se incluyen nuevos casos de uso que no son visibles por ninguno de los actores del sistema, pero que son necesarios para el correcto funcionamiento.

Diagrama de clases

Forma parte de la vista estática del sistema. En el diagrama de clases como ya hemos comentado será donde definiremos las características de cada una de las clases, interfaces, colaboraciones y relaciones de dependencia y generalización. Es decir, es donde daremos rienda suelta a nuestros conocimientos de diseño orientado a objetos, definiendo las clases e implementando las ya típicas relaciones de herencia y agregación.

En el diagrama de clases debemos definir a estas y a sus relaciones.

La Clase

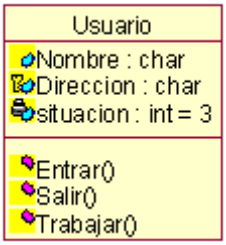
Una clase esta representada por un rectángulo que dispone de tres apartados, el primero para indicar el nombre, el segundo para los atributos y el tercero para los métodos.

Cada clase debe tener un nombre único, que las diferencie de las otras.

Un atributo representa alguna propiedad de la clase que se encuentra en todas las instancias de la clase. Los atributos pueden representarse solo mostrando su nombre, mostrando su nombre y su tipo, e incluso su valor por defecto.

Un método o operación es la implementación de un servicio de la clase, que muestra un comportamiento común a todos los objetos. En resumen es una función que le indica a las instancias de la clase que hagan algo.

Para separar las grandes listas de atributos y de métodos se pueden utilizar estereotipos.



Aquí vemos un ejemplo. La clase usuario contiene tres atributos. Nombre que es public, dirección que es protected y situación que es private. Situación empieza con el valor 3. También dispone de tres métodos Entrar, Salir y Trabajar.

Relaciones entre clases

Existen tres relaciones diferentes entre clases, Dependencias, Generalización y Asociación. En las relaciones se habla de una clase destino y de una clase origen. La origen es desde la que se realiza la acción de relacionar. Es decir desde la que parte la flecha, la destino es la que recibe la flecha. Las relaciones se pueden modificar con estereotipos o con restricciones.

Dependencias

Es una relación de uso, es decir una clase usa a otra, que la necesita para su cometido. Se representa con una flecha discontinua va desde la clase utilizadora a la clase utilizada. Con la dependencia mostramos que un cambio en la clase utilizada puede afectar al funcionamiento de la clase utilizadora, pero no al contrario. Aunque las dependencias se pueden crear tal cual, es decir sin ningún estereotipo (palabreja que aparece al lado de la línea que representa la dependencia) UML permite dar mas significado a las dependencias, es decir concretar mas, mediante el uso de estereotipos.

- Estereotipos de relación Clase-objeto.
  - Bind: La clase utilizada es una plantilla, y necesita de parámetros para ser utilizada, con Bind se indica que la clase se instancia con los parámetros pasándole datos reales para sus parámetros.
  - Derive: Se utiliza al indicar relaciones entre dos atributos, indica que el valor de un atributo depende directamente del valor de otro. Es decir el atributo edad depende directamente del atributo Fecha nacimiento.
  - Friend: Especifica una visibilidad especial sobre la clase relacionada. Es decir podrá ver las interioridades de la clase destino.
  - InstanceOF: Indica que el objeto origen es una instancia del destino.
  - Instantiate: indica que el origen crea instancias del destino.
  - Powertype: indica que el destino es un contenedor de objetos del origen, o de sus hijos.
  - Refine: se utiliza para indicar que una clase es la misma que otra, pero mas refinada, es decir dos vistas de la misma clase, la destino con mayor detalle.

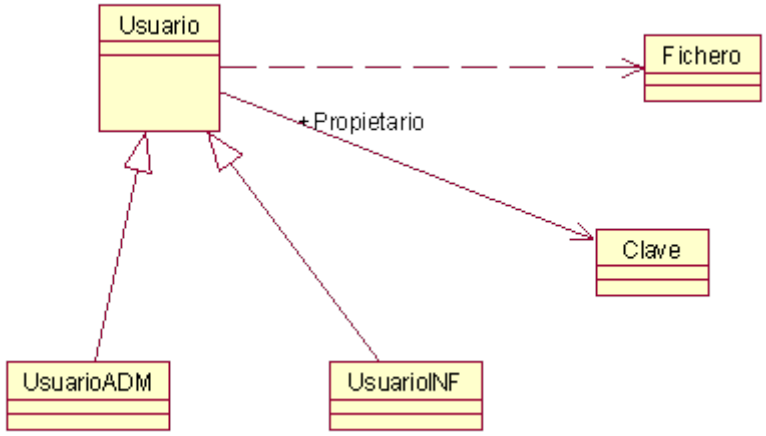
Generalización

Pues es la herencia, donde tenemos una o varias clases padre o superclase o madre, y una clase hija o subclase. UML soporta tanto herencia simple como herencia múltiple. Aunque la representación común es suficiente en el 99.73% de los casos UML nos permite modificar la relación de Generalización con un estereotipo y dos restricciones.

- Estereotipo de generalización.
  - Implementation: El hijo hereda la implementación del padre, sin publicar ni soportar sus interfaces.
- Restricciones de generalización.
  - Complete: La generalización ya no permite mas hijos.
  - Incomplete: Podemos incorporar mas hijos a la generalización.
  - Disjoint: solo puede tener un tipo en tiempo de ejecución, una instancia del padre solo podrá ser de un tipo de hijo.
  - Overlapping: puede cambiar de tipo durante su vida, una instancia del padre puede ir cambiando de tipo entre los de sus hijos.

Asociación

Especifica que los objetos de una clase están relacionados con los elementos de otra clase. Se representa mediante una línea continua, que une las dos clases. Podemos indicar el nombre, multiplicidad en los extremos, su rol, y agregación.

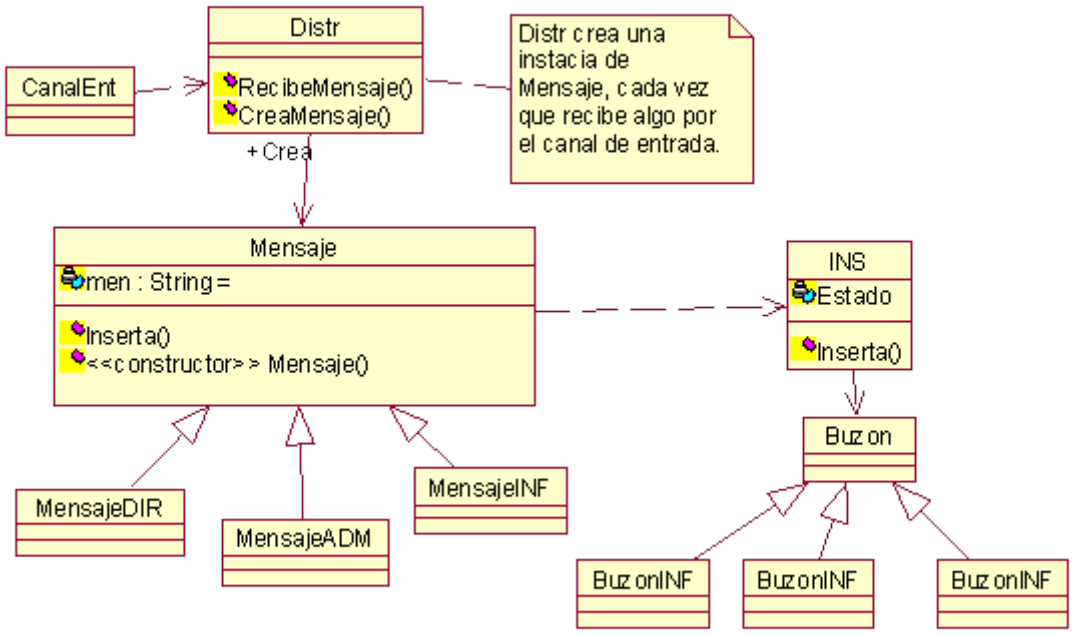


En este diagrama se han creado cuatro clases. La clase principal es Usuario, que tiene dos clases hijas UsuarioADM y UsuarioINF. El usuario mantiene una relación de asociación con la clase Clave, se indica que es propietario de una clave, o de un número indeterminado de ellas. Se le crea también una relación de dependencia con la clase Perfil, es decir las instancias de usuario contendrán como miembro una instancia de Perfil.

Diagramas de objetos

Forma parte de la vista estática del sistema. En este diagrama se modelan las instancias de las clases del diagrama de clases. Muestra a los objetos y sus relaciones, pero en un momento concreto del sistema. Estos diagramas contienen objetos y enlaces. En los diagramas de objetos también se pueden incorporar clases, para mostrar la clase de la que es un objeto representado.

En este diagrama se muestra un estado del diagrama de eventos. Para realizar el diagrama de objetos primero se debe decidir que situación queremos representar del sistema. Es decir si disponemos de un sistema de mensajería, deberemos decidir que representaremos el sistema con dos mensajes entrantes, los dos para diferentes departamentos, dejando un departamento inactivo. Para el siguiente diagrama de clases:



Tendríamos un diagrama de objetos con dos instancias de Mensaje, mas concretamente con una instancia de MensajeDIR y otra de MensajeADM, con todos sus atributos valorados. También tendríamos una instancia de cada una de las otras clases que deban tener instancia. Como CanalEnt, INS, Distr, y el Buzon correspondiente a la instancia de mensaje que se este instanciando. En la instancia de la clase INS se deberá mostrar en su miembro Estado, que esta ocupado realizando una inserción.

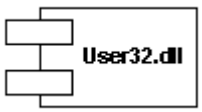
En un diseño no podemos encontrar con multitud de diagramas de objetos, cada uno de ellos representando diferentes estados del sistema.

Diagrama de componentes

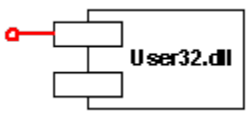
Se utilizan para modelar la vista estática de un sistema. Muestra la organización y las dependencias entre un conjunto de componentes. No es necesario que un diagrama incluya todos los componentes del sistema, normalmente se realizan por partes. Cada diagrama describe un apartado del sistema.

En el situaremos librerías, tablas archivos, ejecutables y documentos que formen parte del sistema.

Uno de los usos principales es que puede servir para ver que componentes pueden compartirse entre sistemas o entre diferentes partes de un sistema.



Aquí tenemos un componente del sistema de Windows. En el diagrama de componentes de Windows debe salir este componente, ya que sin el sistema no funcionaría.



En esta otra figura tenemos el mismo componente, pero indicamos que dispone de un interface. Al ser una Dll el interface nos da acceso a su contenido. Esto nos hace pensar que la representación anterior es incorrecta, pero no es así solo corresponde a un nivel diferente de detalle.

Como ya hemos indicado antes todo objeto UML puede ser modificado mediante estereotipos, los standard que define UML son:

- Executable
- Library
- Table
- File
- Document

Aunque por suerte no estamos limitados a estas especificaciones. Que pasa si queremos modelar un proyecto de Internet donde nuestros componentes son ASP, HTML, y Scripts, y queremos marcarlo en el modelo. Pues utilizamos un estereotipo. Existe ya unos definidos WAE (Web Applications Extensión).

Podemos modelar diferentes partes de nuestro sistema, y modelar diferentes entidades que no tiene nada que ver entre ellas.

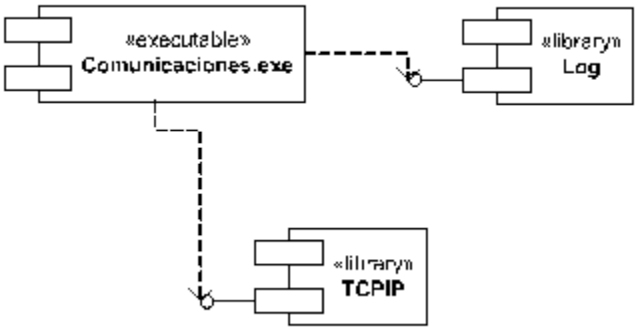
- Ejecutables y bibliotecas
- Tablas
- API
- Código fuente
- Hojas HTML

Ejecutables

Nos facilita la distribución de ejecutables a los clientes. Documenta sus necesidades y dependencias. Si disponemos de un ejecutable que solo se necesita a el mismo para funcionar no necesitaremos el diagrama de componentes.

Los pasos a seguir para modelar, a priori no a posteriori, son:

- Identificar los componentes, las particiones del sistema, cuales son factibles de ser reutilizadas. Agruparlos por nodos y realizar un diagrama por cada nodo que se quiera modelar.
- Identificar cada componente con su estereotipo correspondiente.
- Considerar las relaciones entre componentes.

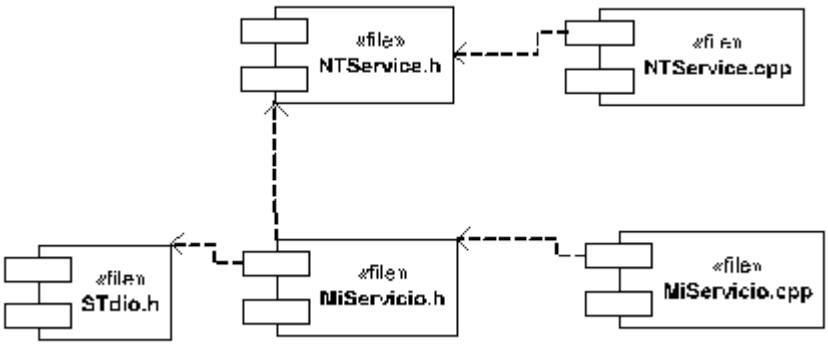


En este grafico se muestra un ejecutable que utiliza dos librerías, estas dos librerías disponen de su interface con el que ofrecen el acceso a sus servicios. Se puede ver que estas librerías son componentes que pueden ser reutilizados en otras partes del sistema.

Código fuente

Se utiliza para documentar las dependencias de los diferentes ficheros de código fuente. Un ejecutable, o librería es una combinación de estos ficheros, y al mostrar la dependencia entre ellos obtenemos una visión de las partes necesarias para la creación del ejecutable o librería.

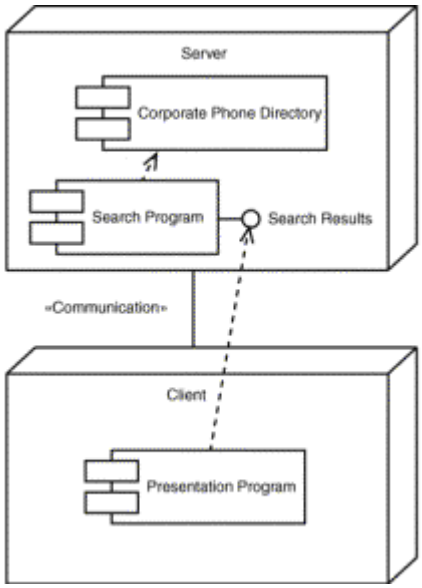
Al tener documentadas las relaciones se pueden realizar cambios en el código de un archivo teniendo en cuenta donde se utiliza, y que otros ficheros pueden verse afectados por su modificación.



Aquí tenemos la relación entre los diferentes ficheros de un sistema. Cada fichero Cpp utiliza su fichero .h correspondiente, y MiServicio.h utiliza NTService.h u Stdio.h.

Diagramas de despliegue

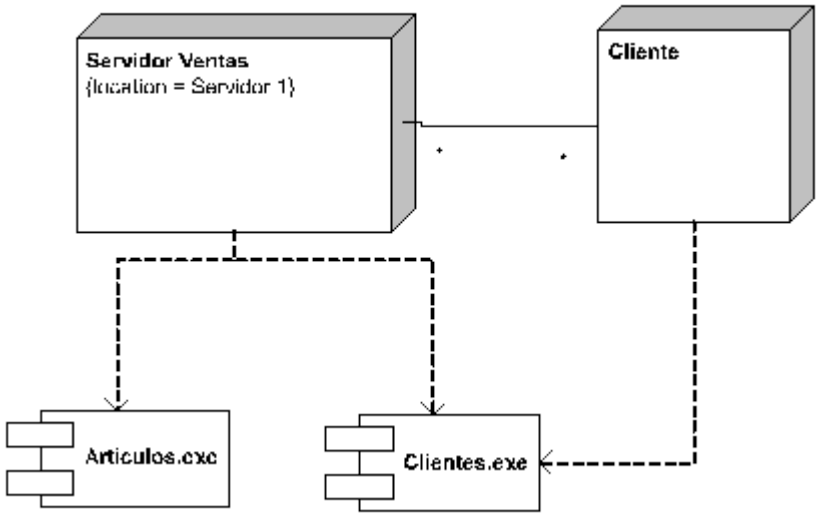
En el diagrama de despliegue se indica la situación física de los componentes lógicos desarrollados. Es decir se sitúa el software en el hardware que lo contiene. Cada Hardware se representa como un nodo.



Un nodo se representa como un cubo, un nodo es un elemento donde se ejecutan los componentes, representan el despliegue físico de estos componentes.

Aquí tenemos dos nodos, el cliente y el servidor, cada uno de ellos contiene componentes. El componente del cliente utiliza un interface de uno de los componentes del servidor. Se muestra la relación existente entre los dos Nodos. Esta relación podríamos asociarle un estereotipo para indicar que tipo de conexión disponemos entre el cliente y el servidor, así como modificar su cardinalidad, para indicar que soportamos diversos clientes.

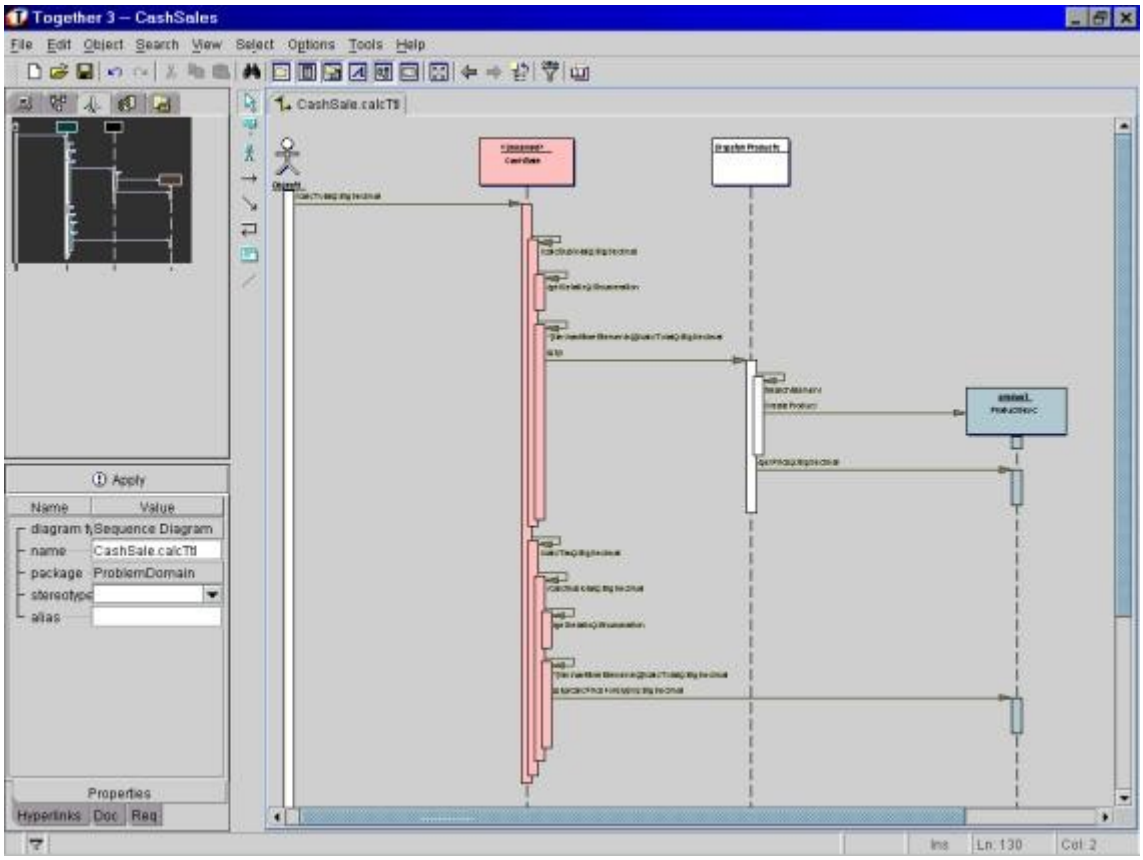
Como los componentes pueden residir en mas de un nodo podemos situar el componente de forma independiente, sin que pertenezca a ningún nodo, y relacionarlo con los nodos en los que se sitúa.



Diagramas secuencia

El diagrama de secuencia forma parte del modelado dinámico del sistema. Se modelan las llamadas entre clases desde un punto concreto del sistema. Es útil para observar la vida de los objetos en sistema, identificar llamadas a realizar o posibles errores del modelado estático, que imposibiliten el flujo de información o de llamadas entre los componentes del sistema.

En el diagrama de secuencia se muestra el orden de las llamadas en el sistema. Se utiliza un diagrama para cada llamada a representar. Es imposible representar en un solo diagrama de secuencia todas las secuencias posibles del sistema, por ello se escoge un punto de partida. El diagrama se forma con los objetos que forman parte de la secuencia, estos se sitúan en la parte superior de la pantalla, normalmente en la izquierda se sitúa al que inicia la acción. De estos objetos sale una línea que indica su vida en el sistema. Esta línea simple se convierte en una línea gruesa cuando representa que el objeto tiene el foco del sistema, es decir cuando el esta activo.



En esta pantalla tenemos tres objetos, y un actor, situado a la izquierda que es el que inicia la acción. Como podemos ver el objeto de mas a la derecha aparece mas abajo que los otros existentes. Esto se debe a que recibe una llamada de creación. Es decir el objeto no existe en el sistema hasta que recibe la primera petición.

Direccion web http://programacion.com/tutorial/uml/

AUTOR

Nombre:
Pere Martra

Email:
oopere@yahoo.com

URL:
http://usuarios.lycos.es/oopere/