Diseño Basado en Componentes

UML aplicado al diseño basado en componentes

Ingeniería Informática
Universidad Carlos III de Madrid

Tabla de contenidos

Introducción a UML. Definición e historia

- UML es un lenguaje gráfico y formal para el modelado de sistemas.
  - Visualizar, especificar, construir y documentar los artefactos de un sistema.
  - Lenguaje, no método – ligado a USDP, pero sirve para otros métodos.
  - Orientado a objetos – toma sus conceptos de los lenguajes OO.
- Origen histórico.
  - Antes de 1994: la “guerra de los métodos”.
  - 1994: **Booch** y **Rumbaugh** en Rational – 1995: **Jacobson** se les une.
  - 1997: primera versión estándar certificada por el OMG.

Introducción a UML. Elementos principales (i)

- Cosas – representan distintos aspectos del modelo:
  - **Estructura**: son los “sustantivos” de UML, tales como clase, interfaz, atributo, componente, nodo...
  - **Comportamiento**: son los “verbos” de UML, tales como acción, actividad, interacción, estado, mensaje...
  - **Agrupamiento**: son los **paquetes**, que se usan para agrupar elementos relacionados semánticamente en unidades coherentes.
  - **Anotación**: son las “notas”, que pueden añadirse en cualquier parte del modelo para capturar información no gráfica.

Introducción a UML. Elementos principales (ii)

- **Relaciones** – representan conexiones entre las cosas:
  - **Asociación**
  - **Dependencia**
  - **Generalización**
  - Otros: ***include*** y ***extend***
- **Mecanismos de extensión**: **estereotipos**, **valores etiquetados** (*tagged values*) y **restricciones** (*constraints).*
- **Diagramas** – representan un conjunto de cosas y relaciones.

Introducción a UML. Elementos principales (iii)

<table>
<tr><td>Modelado de requisitos</td><td colspan="2">Diagrama de casos de uso</td></tr>
<tr><td rowspan="2">Modelado estático</td><td colspan="2">Diagrama de clases</td></tr>
<tr><td colspan="2">Diagrama de objetos</td></tr>
<tr><td rowspan="4">Modelado dinámico</td><td rowspan="2">Interacción</td><td>Colaboración</td></tr>
<tr><td>Secuencia</td></tr>
<tr><td colspan="2">Diagrama de estados</td></tr>
<tr><td colspan="2">Diagrama de actividad</td></tr>
<tr><td rowspan="2">Modelado físico</td><td colspan="2">Diagrama de componentes</td></tr>
<tr><td colspan="2">Diagrama de despliegue</td></tr>
</table>

Paquetes en UML (i)

- Sirven para **organizar** los elementos de un modelado en diferentes **grupos** (incluidos los diagramas). Podrían darse estructuras recursivas, donde un paquete incluye otro paquete.
- También sirven para **organizar modelos grandes**, **agrupar elementos relacionados** o **crear *namespaces***.
- Concepto clave: "**Fuerte cohesión interna** y **bajo acoplamiento externo**" (mostrar).
- Un paquete puede ser incluido en cualquier tipo de diagrama y, a su vez, puede contener también otros diagramas.

Paquetes en UML (ii)

- Cada paquete constituye un "espacio de nombres": **unicidad** de nombres (no se permiten dos elementos del mismo tipo con el mismo nombre).
- Se pueden incluir dependencias entre paquetes para indicar que el contenido de uno de ellos necesita de algún modo parte del contenido del otro paquete.

Paquetes en UML. Visibilidad

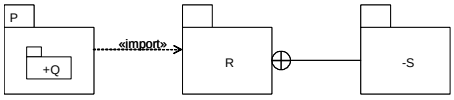
- El nombre completo cualificado de un elemento *x* que resida dentro del paquete *Pq1* es: Pq1::x
- Los elementos de un paquete pueden tener visibilidad **pública** (+), **privada** (-) o **protegida** (#).
- Un elemento de un paquete puede referenciar:
  - Elementos **públicos** de cualquier paquete al que tenga permisos: **nombre cualificado** (P::A).
  - Elementos (incluso **privados**) de su paquete, o cualquier paquete **que incluya al suyo**.
  - Elementos **protegidos** de sus paquetes *padre* (relación de generalización).

Paquetes en UML. Accesibilidad (i)

- Para que un elemento de un paquete tenga acceso (permisos) a otro elemento de otro paquete se debe crear una relación entre ambos paquetes.
- Se debe tener este tipo de permiso por ejemplo, para crear una asociación entre dos elementos de distintos paquetes, o para definir el tipo de dato de un atributo.

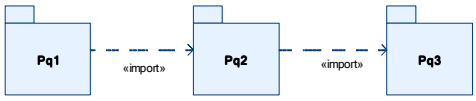
Paquetes en UML. Accesibilidad (ii)

- Las relaciones de permiso son:
  - <<import>>: Incorpora el espacio de nombres de un paquete a otro (ojo a duplicados). No hace falta cualificar nombres. Puede darse un alias a cada elemento importado y definir una nueva visibilidad.
  - <<access>>: permite acceder al espacio de nombres de otro paquete sin llegar a incorporar sus elementos. Los elementos accedidos se indican con su nombre cualificado.



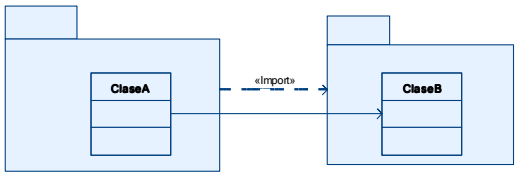
Paquetes en UML. Accesibilidad (iii)

- Las relaciones de permiso **no** son **simétricas** ni **transitivas**.
  - Pq1 puede acceder al contenido público de Pq2 y Pq2 al de Pq3 pero:
    - Pq2 no puede acceder a ningún contenido de Pq1.
    - Pq1 no puede acceder a ningún contenido de Pq3.



Paquetes en UML. Accesibilidad (iv)

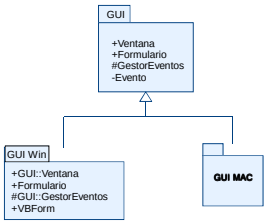
- Debido a que no son simétricas, la siguiente asociación es poseída por el paquete de la izquierda, y sólo puede ser navegable en el sentido A→B.



Paquetes en UML. Generalización (i)

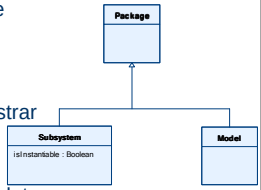
- UML define el concepto de **generalización** entre paquetes.
- Conceptos como **"principio de sustitución"** y **visibilidad** también son utilizados a nivel paquete.
- Un contenido también puede ser **sobrescrito** en un paquete hijo.
- El paquete hijo hereda los contenidos públicos y protegidos del padre, incluyendo los contenidos importados (no accedidos por el padre).

Paquetes en UML. Generalización (ii)



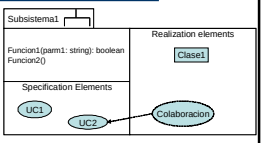
Paquetes en UML. Tipos de paquetes (i)

- Según el metamodelo de UML, existen dos tipos especiales de paquetes
- **Modelo** (raíz): sirve para representar el primer nivel de agrupamiento. Otros modelos pueden ser utilizados para mostrar un contenido desde distintos puntos de vista.
- **Subsistema**: sirve para descomponer un sistema completo a un elevado nivel de abstracción. También es un tipo de clasificador, por lo que puede proporcionar interfaces y definir operaciones (servicios) de alto nivel.



Paquetes en UML . Tipos de paquetes (ii)

- Subsistema:
  - Puede ser mostrado con tres compartimentos:
    - **Operaciones**: se implementarán mediante una clase pública del subsistema. Son los servicios del subsistema.
    - Elementos de **especificación**: casos de uso e interfaces que se muestran al exterior ocultando su verdadera realización.
    - Elementos de **realización**: contiene los elementos que realizan los casos de uso, o las clases que implementan las operaciones del subsistema.
  - Son el origen de los futuros componentes.

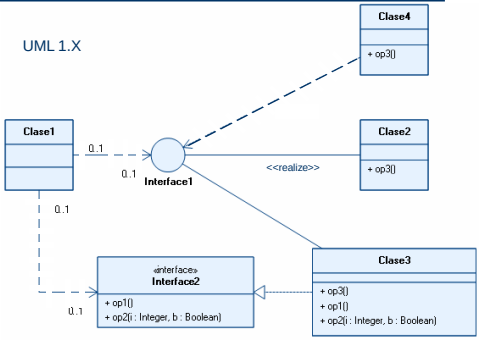


Implementación de Interfaces (i)

- Una interfaz (<<interface>>) es una colección de un conjunto de **operaciones** que se utiliza para especificar un **servicio** de una **clase** o **componentes**.
- Una interfaz no puede tener atributos ni asociaciones navegables.
- Separan la **especificación** de una funcionalidad de su **implementación**.
- La **realización** de una interfaz es una dependencia con el estereotipo ***<<realize>>***.
- Una interfaz puede ser vista en su forma de icono (un círculo) o su forma extendida incluyendo sus operaciones.

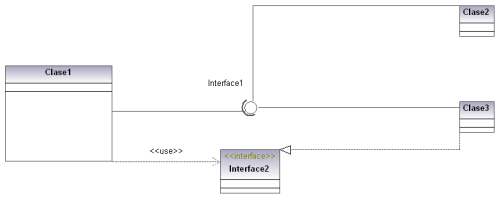
Implementación de Interfaces (ii)

UML 1.X



Implementación de Interfaces (iii)

UML 2.X



Diagramas de componentes (i)

- Definición de **componente**:
  - Un componente es una unidad de '**despliegue**' independiente.
  - Un componente podría ser utilizado por cualquier organización sin necesidad de ningún conocimiento previo.
  - Un componente no tiene estado persistente en sí mismo, lo tienen (opcionalmente) sus objetos.
  - Implementa una serie de **interfaces** y utiliza otras.
  - Encierran otros elementos de modelado como por ejemplo clases, permitiendo a otros componentes acceder a ellos.
  - Ejemplos: ficheros fuente, controles ActiveX, JavaBeans, Assemblies .Net, documentos,...

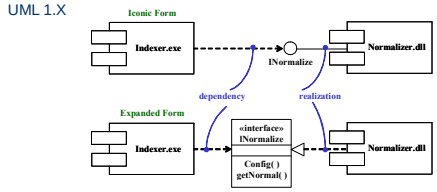
Diagramas de componentes (ii)

- Filosofía: **"Alta cohesión interna, bajo acoplamiento externo"**.
- Las interfaces **implementadas** o **utilizadas** por un componente son realmente utilizadas y/o implementadas por su contenido.
- Unos componentes hacen **uso** de otros. Esto se denota mediante una relación de **dependencia**.
- Para bajar el acoplamiento, no se hace que un componente dependa de otro, sino de una o varias de sus **interfaces**.

Diagramas de componentes (iii)

- Diagrama de componentes:
  - Representa la estructura física de **implementación**.
  - Un componente puede ser un ejecutable, librería dll, documento...

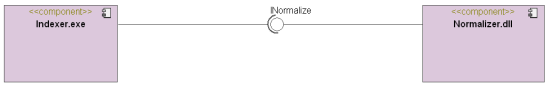
UML 1.X



Diagramas de componentes (iv)

UML 2.X

Iconic form



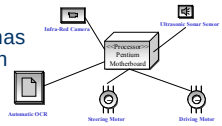
Expanded form

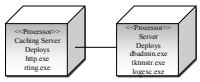
Diagramas de despliegue (i)

- **Nodo**:
  - Representa una parte **física** (hardware) de un sistema.
  - No tiene porqué ser un elemento computacional, podría ser un sensor.
  - Sobre cada nodo se ejecutan los distintos **componentes** del sistema.
  - Pueden usarse estereotipos para diferencias sus diferentes tipos y *taggedValues* para indicar sus cualidades.
- **Asociaciones** entre nodos:
  - Representan conexiones físicas entre nodos (redes...).

Diagramas de despliegue (ii)

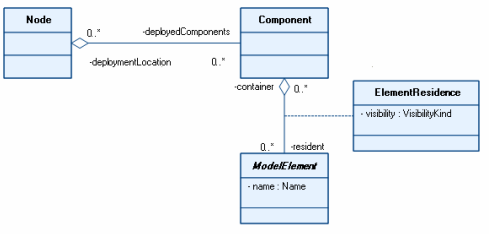
- **Diagramas de despliegue**:
  - Capturan la **topología física** del sistema.
  - No suelen utilizarse con sistemas monolíticos donde sólo se usan periféricos estándar.
  - Pueden ser vistos de diversas formas:
    - Sólo nodos en el diagrama (y sus asociaciones o dependencias).
    - Nodos con dependencias a sus respectivos componentes.
    - Nodos que incluyen dentro sus componentes.





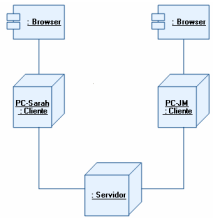
Diagramas de despliegue (iii)

- Vista de **Metamodelo** de UML:



Diagramas de despliegue (iv)

- En un diagrama de despliegue se pueden incluir tanto **instancias de componentes** como **instancias de nodos**.
- Esto permite diseñar escenarios concretos de ejecución.



Bibliografía

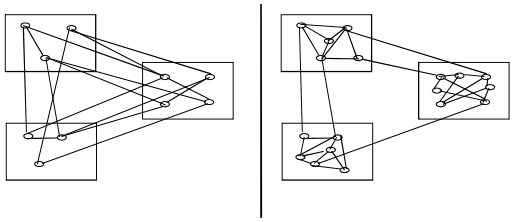
- UML and the Unified Process (Capítulos 11, 17, 21, 22, 23)
  Jim Arlow
  Addison Wesley, 2002
- Utilización de UML en Ingeniería del Software con Objetos y Componentes (Capítulos 13 y 14)
  Perdita Stevens
  Addison Wesley, 2000
- El Lenguaje Unificado de Modelado
  Grady Booch, James Rumbaugh, Ivar Jacobson
  Addison Wesley, 1999
- El Estándar de UML v. 1.5: http://www.omg.org

Diseño Basado en Componentes

UML aplicado al diseño basado en componentes

Ingeniería Informática
Universidad Carlos III de Madrid

Acoplamiento



Alto acoplamiento

Bajo acoplamiento

Volver