# **VISTA DE MÓDULOS – ESTILOS**

## **ESTILO DE DESCOMPOSICIÓN (es parte de)**

Descomponemos el sistema en **unidades de implementación** y de esta forma organizamos el código en módulos y sub-módulos.

### **ESTILOS**

* CRITERIOS para descomponer

Acoplamiento

Distribuir trabajo (equipos)

Atributos de calidad

Documentar el criterio

* Heurísticas

Módulo es pequeño si se puede reimplementar en caso de abandono de un miembro.

* Para qué?

Nuevas incorporaciones, navegación, distribución de trabajo.

* Restricciones 🡪 no hay bucles, y el módulo máximo pertenece a un agregado
* Notación informal 🡪 grafico/textual
* UML 🡪 paquete=módulo
* Relación con otras vistas: C&C Y asignación de trabajo

## **ESTILO DE USO (uso)**

* Nos dice que otros módulos usa un módulo. A usa B,si para que A funcione ok necesita a B.
* Para qué?

Desarrollo incremento radual, pruebas, estimar efectos de un cambio determinado.

* No hay restricciones. Hay bucles cadenas largas de dependencia…
* Notación informal 🡪 matriz de dependencia
* Notación UML
* Relación con otras vistas: Capas, debe precederlo.

## **ARQUITECTURA POR CAPAS**

Obtenemos módulos con **acoplamiento bajo**. Cohesión en cada capa. Dividimos el software.

Las **interfaces** hacen públicas las capas.

Arquitecturas **cerradas** o **abiertas.**

**Restricciones:**

* Al menos dos capas
* Cada módulo en una capa
* No hay restricciones circulares

Las capas no aparecen en código, si examinamos esta sacamos las relaciones **de uso**.

Nos sirven para:

* Crear equipos de diferentes capacidades
* Re-utilizar capas

Para qué?:

* **Modificabilidad – Portabilidad – Mantenibilidad**
* Cada capa **oculta información.** Los cambios de implementación quedan ocultos por las interfaces, promueven portabilidad y propagan únicamente cambios en el rendimiento.
* Crear y comunicar arquitectura

Las capas e ifaces promueven mantenibilidad de sistemas complejos.

Con la vista de descomposición entendemos el código y asignamos trabajo.

Documentar

* para que sirve cada capa. Lista de módulos y como se ha implementado.
* Que capas puede usar cada una
* Criterio a seguir

UML🡪 paquete esteriotipado con layer

Realacionado con la vista de descomposición:

* Establece correspondencia entre módulos de vistas
* Dos submódulos de un mismo módulo pueden pertenecer a diferentes capas.
* Un módulo pude aparecer en más de una capa

## **ASPECTOS (aislar)**

Se usa para aislar en la arquitectura los módulos que implenentan aspectos transversales.

Algunos mezclan negocios con transversa, pej banca: modulo cliente, cuenta atm…

**Natural** en OOP, lo hace mas fácil de modifcar.

En UML clases estereotipadas <<aspect>>

## **GENERALIZACIÓN (is a)**

En OO es entre clases, ahora módulos.

Podemos representar herencia de implementación y de interfaces

# **VISTA DE COMPONENTE Y CONECTORES C&C**

## **VISTA**

Muestran funcionamiento del sistema.

Especifican estructura y comportamiento de los elementos en tiempo de ejec.

Nos muestra los principales componentes y su interacción, las partes del sistema replicadas y las veces, el progreso de los datos por el sistema en t.exec, protocolos de comunicación, exec paralela…

## **COMPONENTES**

**Unidades de procesamiento principal y almacenes de datos.**

**Puertos🡪**  permite interactuar con otros componentes mediante conectores

Puede tener varios puertos (server atiende n clientes).

Propiedades: Nombre, tipo, fiabilidad, prestaciones, requisitos, seguridad, concurrencia, tasa de peticiones max…

## **PUERTOS**

**Interfaz de un componente, un punto de interacción con su entorno.**

Suelen tener un tipo explícito. Define comportamiento en la iteracción. **Hay que documentarlos.** **Los puertos se pueden replicar, las interfaces no.**

Si hay subarquitectura, documentar relación entre puertos internos y externos.

## **CONECTORES**

**Caminos de interacción entren los componentes.** (tubería, invocar, cola de mensajes…).

Tienen **ROLES** 🡪 indican cómo los componentes pueden usarlos en sus interacciones. Es la **iface** de un conector.

Asociaciones de puerto a puerto con los roles, podemos incluir un tag.

Podemos crear un componente como mediador. Pej, Pub/Sub

Propiedades: Nombre, tipo, fiabilidad, prestaciones, requisitos, seguridad, concurrencia, tasa de peticiones max…

## **RELACIONES**

**Acoplamiento:** asociación puerto-rol y crea un grafo de C&C. Valido si son compatibles (documentarlo).

**Delegación de interfaces:** puertos externos asociados a los internos, también para roles.

## **RESTRICCIONES**

Componente solo se asocia a conectores y viceversa.

Los acoples solo entre puertos y roles.

Delegar ifaces solo entre puertos o roles compatibles.

No hay conectores aislados.

## **TIPOS E INSTANCIAS**

Tipo es un componente o conector no definido completamente.

Instancia es el resultado de completar esa la definición.

## **INTERFACES**

Un componente las propone o las requiere.

Asociadas a un puerto.

## **RELACIONES**

Vista de módulos COMPLEJA: un componente que ejecuta código de varios módulos…

Vista de distirbución: vista C&C muestra elementos en t exec – relacionado con canales de comunicación donde se ejecutan.

# **CONECTORES**

Integración e interacción de los componentes.

Pueden:

Proporcionar servicios de perisistencia, mensajería, transacciones…

**Aka *facilities components***

A veces son componentes al ser tan complejos.

**Responsabilidad del arq. de software:**

* Entender interacción necesaria para cada componente.
* Identificar atributos relevantes
* Elegir conectores candidatos
* Analizar propiedades clave de la interacción.

Nos ayuda a ver como realiza tareas el sistema pero sin saber qué hace

## **CATEGORIAS**

Describen el tipo de servicio que pueden dar (tipo interacción). Puede tener mas de una.

1. **Comunicación**: transferir info
2. **Coordinación:** soporte a transferencia de control
3. **Conversión:** componentes no diseñados para interactuar que lo hagan (wrappers)
4. **Facilitación:** servicios para facilitar y optimizar interacción

## **TIPOS DE CONECTORES**

1. **Llamada a procedimiento:** más común, diferentes técnicas. Pej, RPC, fork..
2. **Evento:** desencadena el flujo, puede llevar info. Conectores virtuales entre componentes interesados en el mismo evento.
3. **Acceso a datos:** Permite acceder a datos mantenidos por un componente, incluso puede realizar conversiones.
4. **Linkage:** Une componentes durante su interacción habilitando canales. Tiene **granularidad** (unidad, sintáctica, semántica) **y cardinalita.**
5. **Stream:** Transferir gran cantidad de info entre procesos autónomos. Combina con acceso a datos y evento. Pej: sockets, pipes…
6. **Mediador:**  coordina y facilita, componente sabe de la existencia de otros pero no el estado, el conector organiza la ejecución del sistema, resuelve conflictos…
7. **Adaptador:** hacen posible la operabilidad entre compontes heterogéneos
8. **Distribuidor:**  Facilitan la identificación de caminos de interacción entre componente y establecen comunicación y distribución. (dns, routing…)

Podemos construir conectores compuestos más complejos.

# **VISTA DE DISTRIBUCIÓN**

Establece relaciones entre la arquitectura del software y los elementos no software.

**Entorno:** lugar donde se desarrolla, despliega y ejecuta el software.

**Elementos:**  hw, redes, FMS, equipos de desarrollo…

Un elemento de la arq se puede asignar a diferentes elementos del entorno.

Si la distribución cambia, la arquitectura es dinámica

**El objetivo: comparar las propiedades que requiere el elemetno sw con las propiedades que proporciona el elemento del entorno.**

## **VISTA DE DESPLIEGUE**

* **ELEMENTOS SW:** Entidades de exec.
  + **PROPIEDADES:**
    - Consumen recursos
    - Requisitos de recursos y restricciones a satisfacer
    - Condiciones de disponibilidad
* **ELEMENTOS DEL ENTORNO:** Hardware
  + **PROPIEDADES:**
    - CPU
    - MEMORIA
    - Discos
    - Ancho de banda
    - Tolerancia a fallos
* **RELACIONES**
  + Asignación: en que unidades físicas están los elementos software en el momento determinado
  + Si es **dinámico:** el componente se puede mover de un procesador a otro.
    - Migrar
    - Copiar-migrar
    - Ejecutar-migrar

**Asignación válida:** los requisitos expresados por los elementos sw son satisfechos por las características de los elementos hw.

## **VISTA DE INSTALACIÓN**

Si el sistema es grande, hay muchos ficheros instalados. Además, podemos tener versiones para diferentes plataformas.

Debemos agruparlos para instalarlos ene el entorno de producción.

Puede ser interesante tener varias versiones.

Ayuda ver que ficheros hay que usar y como se deben configurar para producir las versiones.

**Para qué?**

* Crear build tolos
* Mantener control e integridad del proceso de empaquetado y sistema desplegado
* Navegación por los ficheros
* Seleccionar y configurar ficheros para crear versiones
* Actualizar versiones
* Diseñar e implementar actualizaciones

## **VISTA DE ASIGNACIÓN DE TRABAJO**

Asigna módulos a equipos o personas que serán responsables de llevarlos a cabo. Se usa la vista módulos.

Debe ayudar y ser utilizada por el gestor del proyecto para planificar los recursos, asignar responsabilidades, planificar, presupuestos…