Arquitecturas Software

Juan José Moreno Navarro
(Curso de Software basado en Componentes, junto a Lars-Ake Fredlund)

Arquitecturas Software

- Motivación:
  - Complejidad creciente de aplicaciones.
  - Sistemas distribuidos (segunda coordenada de complejidad).
  - Sistemas abiertos y basados en componentes (tercera coordenada de complejidad).
- Idea principal:

  Estructura de alto nivel de un sistema software y sus propiedades globales.

Arquitecturas Software

- Características:
  - Parte del diseño de software.
  - Nivel del diseño de software donde se definen la estructura y propiedades globales del sistema.
  - Incluye sus componentes, las propiedades observables de dichos componentes y las relaciones que se establecen entre ellos.
  - Un aspecto crítico: Una arquitectura errónea puede llevar a problemas incontables.

Arquitecturas Software

- Características:
  - Representación de alto nivel de la estructura del sistema describiendo las partes que lo integran.
  - Puede incluir los patrones que supervisan la composición de sus componentes y las restricciones al aplicar los patrones.
  - Trata aspectos del diseño y desarrollo que no pueden tratarse adecuadamente dentro de los módulos que forman el sistema.

Arquitecturas Software

- Nueva disciplina:
  - Toda aplicación tiene una arquitectura, aunque no sea explícita.
  - Tradicionalmente ha habido un repertorio de técnicas, patrones y expresiones para estructuras sistemas software complejos.
- Arquitectura de Software:
  - Hace explícito con rigor lo anterior.
  - Incluye modelos, lenguajes y herramientas para la descripción y desarrollo práctico de arquitecturas software.

Arquitecturas Software

- Objetivos:
  - Comprender y mejorar la estructura de las aplicaciones complejas.
  - Reutilizar dicha estructura (o partes de ella) para resolver problemas similares.
  - Planificar la evolución de la aplicación, identificando las partes mutables e inmutables de la misma, así como los costes de los posibles cambios.
  - Analizar la corrección de la aplicación y su grado de cumplimiento respecto a los requisitos iniciales.
  - Permitir el estudio de algunas propiedad específica del dominio.

Arquitecturas Software

- ¿De qué se ocupa?
  - Diseño preliminar o de alto nivel.
  - Organización a alto nivel del sistema, incluyendo aspectos como la descripción y análisis de propiedades relativas a su estructura y control global, los protocolos de comunicación y sincronización utilizados, la distribución física del sistema y sus componentes, etc.
  - Otros aspectos relacionados con el desarrollo del sistema y su evolución y adaptación al cambio: composición, reconfiguración, reutilización, escalabilidad, mantenibilidad, etc.

Arquitecturas Software

- ¿De qué no se ocupa?
  - Diseño detallado.
  - Diseño de algoritmos.
  - Diseño de estructuras de datos.

Arquitecturas Software

- ¿Qué elementos intervienen?
  - Componentes.
  - Realizan cómputos y almacenamiento de datos.
  - Interaccionan unos con otros durante la ejecución.
- ¿Qué elementos no intervienen?
  - Módulos (fuente) del sistema.
  - Interacción: relaciones de definición/uso.
  - Importación o exportación de elementos definidos en el código fuente.

Arquitecturas Software

- ¿Cómo interviene en la consecución de una solución?
  - Métodos arquitectónicos.
  - Espacio de los diseños arquitectónicos: propiedades de los diferentes diseños arquitectónicos y su capacidad para resolver diferentes problemas.
- ¿Cómo no interviene?
  - Métodos de desarrollo de software: Proporcionan un camino entre el espacio del problema y la solución.

Arquitecturas Software

- Puntos en común:
  - Los métodos de desarrollo suelen basarse en un estilo arquitectónico preferido.
  - Nuevos estilos arquitectónicos proponen nuevos métodos de desarrollo.

Un mismo diseño arquitectónico puede servir para dos aplicaciones distintas (ej. los patrones de diseño)

Estilos arquitectónicos

- Clasificación de los sistemas software en grandes familias cuyos integrantes comparten un patrón estructural común.
- Ejemplos: Filtros y *pipes*, organizados en capas, cliente/servidor, etc.
- Intervienen: patrón de organización general, tipos de componentes presentes habitualmente, interacciones entre ellos.
- Ejemplos de componentes: clientes, servidores, filtros, niveles, bases de datos, ...
- Ejemplos de mecanismos de interacción: RPC, paso de mensajes, protocolos de comunicación, ...
- El campo de aplicación no es relevante.

Estilos arquitectónicos

Sistemas de flujo de datos	*Procesamiento por lotes*
	Filtros y pipes
Sistemas basados en llamada y retorno	*Programa principal y subrutinas*
	Orientados a objetos
	Organizados en capas
Sistemas de componentes independientes	*Comunicación entre procesos*
	Cliente/servidor
	Basados en eventos

Estilos arquitectónicos

Sistemas centrados en los datos	*Repositiorios*
	Pizarras
Máquinas virtuales	*Intérpretes*
	Basados en reglas
Sistemas heterogéneos	*Localmente heterogéneos*
	Jerárquicamente heterogéneos
	Simultáneamente heterogéneos

Estilos arquitectónicos

- Indica
  - Los tipos de componentes y conectores involucrados.
  - Patrones y restricciones de interconexión o composición entre ellos: *Invariantes* del estilo.
- Asociados a cada estilo hay una serie de propiedades que lo caracterizan, determinando sus ventajas e inconvenientes, condicionando la elección de uno u otro estilo.

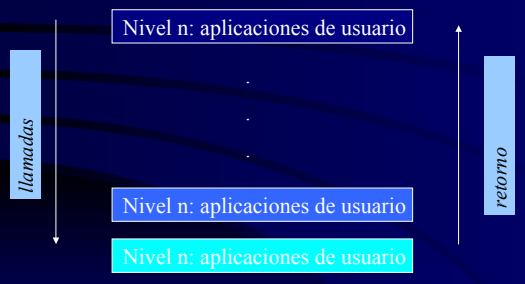
Ejemplo: Organización en capas

- Los componentes son las capas o niveles que pueden estar implementadas internamente por objetos o procedimientos.
- Cada nivel tiene asociado una funcionalidad:
  - Niveles bajos: Funciones simples, ligadas al hardware o al entorno.
  - Niveles altos: Funciones más abstractas.
- Mecanismos de interacción entre componentes:
  - Llamadas a procedimientos.
  - Llamadas a métodos.

Ejemplo: Organización en capas

- Restricciones:
  - Solo llamadas de niveles superiores a inferiores.
  - (Variante) Solo llamadas entre niveles adyacentes.
- Aplicación:
  - Torres de protocolos de comunicación,
  - Sistemas operativos,
  - Compiladores.

Ejemplo: Organización en capas



Ejemplo: Organización en capas

- Propiedades:
  - Facilita la migración. El acoplamiento con el entorno está localizado en las capas inferiores. Estas son las únicas a re-implementar en caso de transporte a un entorno diferente.
  - Cada nivel implementa unas interfaces claras y lógicas, lo que facilita la sustitución de una implementación por otra.
  - Permite trabajar en varios niveles de abstracción. Para implementar los niveles superiores no necesitamos conocer en entorno subyacente, solo las interfaces que proporcionan los niveles inferiores.

Modelos y arquitecturas de referencia

- Particularizan un estilo imponiendo una serie de restricciones sobre el mismo y realizando una descomposición y definición estándar de componentes.
- OSI (*Open System Interconection*) que particulariza el estilo de organización en capas, con 7 niveles.
- RM-ODP para el diseño de sistema distribuidos y abiertos (ISO/ITU-T, 1995). Hace transparente la heterogeneidad de plataformas, s.o., lenguajes, ...Se basa en el estilo de componentes independientes.
- CCM, COM, JavaBeans - Sistemas basados en el intercambio de eventos.

Marcos de trabajo/Frameworks

- Definen una arquitectura adaptada a las particularidades de un determinado dominio de aplicación, definiendo de forma abstracta una serie de componentes y sus interfaces y estableciendo las reglas y mecanismos de interacción entre ellos.
- Suele incluirse la implementación de algunos de los componentes e incluso varias implementaciones alternativas.
- El usuario
  - Selecciona, instancia, extiende y reutiliza los componentes del marco.
  - Completa la arquitectura con nuevos componentes específicos dentro de las relaciones estructurales del marco.

Marcos de trabajo

- Básicamente se presentan como un diseño reutilizable de todo o parte de un sistema, representado por un conjunto de componentes abstractos y la forma en que los componentes interactuan.
- Una alternativa es verlos como un esqueleto de una aplicación que debe ser adaptado por el programador según sus necesidades concretas.
- Un marco de trabajo define el patrón arquitectónico que relaciona los componentes de un sistema.

Marcos de trabajo

- Presentan dos niveles:
  - Especificación de la arquitectura marco.
  - Implementación del marco de trabajo (normalmente un lenguaje orientado a objetos).
- Al desarrollo de aplicaciones a partir de un marco de trabajo se le denomina *extensión del marco*:
  - Puntos de entrada (hot spots): Componentes o procedimientos cuya implementación final depende de la aplicación concreta.
  - Definidos por el diseñador del marco para que sean la forma natural de la extensión del mismo.

Marcos de trabajo

- Dependiendo de la extensión tenemos:

Marcos de trabajo de caja blanca

Que se extienden mediante herencia, concretamente mediante la implementación de las clases y métodos abstractos definidos como puntos de entrada. Se tiene acceso al código del marco y se permite reutilizar la funcionalidad de sus clases mediante herencia y redefinición de métodos.

Marcos de trabajo de caja de cristal

Admiten la inspección de su estructura e implementación, pero no su modificación ni extensión, excepto en los puntos de entrada.

Marcos de trabajo

Marcos de trabajo de caja gris

Los puntos de entrada no son simplemente métodos abstractos, de los que se declara meramente su signatura, sino que se definen por medio de un lenguaje de especficación de alto nivel los requisitos que deben cumplirse a la hora de implementarse.

Marcos de trabajo de caja negra

Su instanciación se realiza por medio de composición y delegación, en lugar de utilizar la herencia. Los puntos de entrada se definen por medio de interfaces que deben implementar los componentes que extiendan el marco.

Marcos de trabajo

- A favor de la caja negra:
  - Se evitan las dependencias entre la implementación del marco y sus clientes.
- A favor de la caja gris:
  - Más flexible que el anterior y similares ventajas.
- A favor de la caja blanca:
  - Problema de los anteriores: Clarividencia. Obliga a que todos los interfaces se definan de antemano contemplando todos los usos futuros del marco.
  - Es posible redefinir componentes y métodos implementados.
  - Exige estar familiarizado con la implementación.

Marcos de trabajo

- Dependiendo de su aplicabilidad tenemos:

Marcos de trabajo horizontales

- Válidos para todos los dominios de aplicación concretados en un aspecto del sistema.
- Infraestucturas de comunicación, interfaces de usuario, entornos visuales, etc.
- Marcos de trabajo distribuidos (*Middelware Application Frameworks*) o *Plataforma de Componentes Distribuidos* para integrar componentes distribuidas.
- Aíslan las dificultades conceptuales y técnicas del desarrollo de aplicaciones distribuidas basadas en componentes.
- CORBA, Active X/OLE/COM, JavaBeans, ACE, Hector, Aglets.

Marcos de trabajo

Marcos de trabajo verticales

- Desarrollados específicamente para un dominio de aplicación.
- Telecomunicaciones, fabricación, servicios telemáticos y multimedia, etc.
- TINA:
  - Marco de trabajo (a partir de RM-ODP) para el desarrollo de servicios avanzados en telecomunicación.
  - Tres niveles + componentes independientes del servicio concreto a implementar.
- MultiTEL (Servicios avanzados de telecomunicación y multimedia)

Marcos de trabajo

Marcos de trabajo para Componentes (*Component Frameworks*)

- Verticales u horizontales, pero exclusivamente realizados para el desarrollo de sistemas basados en componentes reutilizables.
- Características especiales para tratar problemas propios de los SBC: Composición tardía, extensibilidad, ...)
- Pueden verse como una implementación concreta de uno o más patrones de diseño mediante componentes reutilizables.
- OpenDoc, BlackBox (con Oberon)

Marcos de trabajo

- El problema de la documentación del marco de trabajo.
  - El desarrollo es la extensión de un marco de trabajo, luego el usurio debe tener una buena documentación de los puntos de entrada y cómo adaptarlos.
  - Sugerencias para su correcta realización:
    - Patrones de diseño.
    - Diagramas de secuencia de mensajes.
    - Contratos de reutilización.
  - Solución aceptable: Un lenguaje de descripción de arquitecturas + entornos visuales (navegación)

Marcos de trabajo

- El problema de la composición de marcos de trabajo.
  - Usualmente se necesitan utilizar varios y cada uno resuelve un problema concreto (p.ej. un MT vertical y dos horizontales).
  - Gestión del control de la aplicación, ya que los MT suelen tomar el control de la aplicación.
  - Adaptación de los servicios ofrecidos por cada uno de los MTs: los servicios ofrecidos por dos MT pueden no ser compatibles.

Marcos de trabajo

- Falta de funcionalidad y servicios. Aparece cuando la unión de MTs no cubre las necesidades de la aplicación. Las extensiones han de ser compatibles con todos los servicios de los distintos MTs.
- Solapamiento de representaciones. Dos de los MTs utilizados representan de forma algo distinta una entidad del mundo real. (ejemplo, un reloj).
- Solapamiento de funcionalidad. Surge cuando dos MTs ofrecen la misma funcionalidad. Se debe asignar solo a uno y asegurarse que todos los MTs reciben información de que una función se ha efectuado (patrón observer).

Marcos de trabajo

- **Ventajas de la utilización de marcos de trabajo:**
  - **Son composicionales.**
  - **Son el grado más alto de reutilización dentro del desarrollo de software.**
  - **El diseño arquitectónico se reutiliza.**
  - **Reducción de costes/Mejora de la calidad.**
  - **Están intrínsecamente unidos a los componentes ya que además de proporcionar funcionalidad de los componentes permiten la composición entre ellos de forma consistente.**

Patrones de Diseño

- Un patrón es una solución probada que se puede aplicar con éxito a un determinado tipo de problemas que aparecen repetidamente en algún campo.
- Son útiles en cualquier fase del diseño:
  - Patrones estructurales para el diseño de bajo nivel.
  - Patrones arquitectónicos para el diseño de alto nivel.
- Propuestos por el *Gang of Four* (Gamma, Helm, Johnson y Vlissides).

Patrones de Diseño

- Descripción de los patrones de diseño + catalogo de los más comunes (libro de GoF).
- La edición original se ha completado con 2 libros más.
- Idea básica: Aunque la orientación a objetos facilita la reutilización de código, la reutilización efectiva sólo se produce a partir de un buen diseño basado en el uso de soluciones que han probado su utilidad en situaciones similares (los patrones).
- Son un esqueleto básico que cada diseñador adapta a las peculiaridades de su aplicación.

Patrones de Diseño

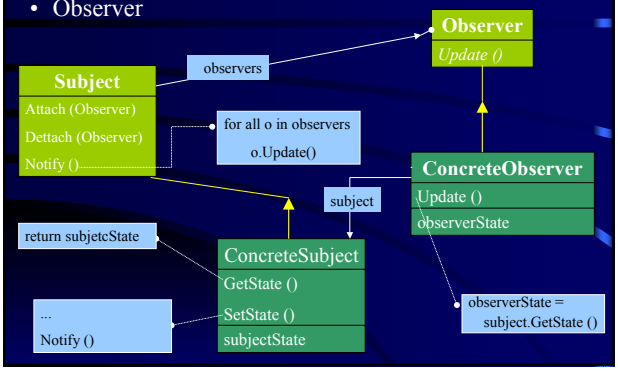
- Descripción de los patrones de diseño:
  - Name & Classification (Class/Object Creational/Structural/Behavoral).
  - Also known as
  - Motivation
  - Applicability
  - Structure
  - Participants
  - Collaborations
  - Implementation
  - Sample code
  - Known Uses
  - Related Patterns

Patrones de Diseño: Ej

- Observer
  - Define una dependencia 1-N entre objetos, de manera que cuando un objeto cambia todas sus dependencias se actualizan automáticamente.
  - Interacción también conocida como *publish-subscribe*
  - Objeto a controlar: Subject. Dependencia: Observadores
  - La idea es tener un objeto para mantener los observadores que los almacena en una colección. Cuando se modifican se indica esto a todo los observadores.
  - La herencia se usa para asegurarse que todos los observadores son del mismo tipo y con una operación *Update*.
  - También sirve como interfaz abstracto del Subject

Patrones de Diseño: Ej

- Observer

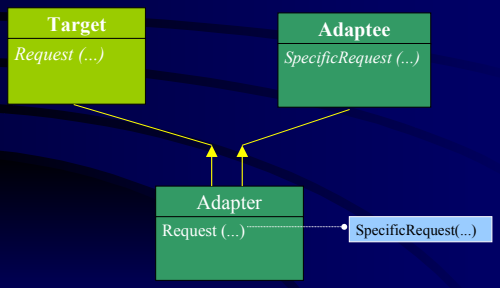


Patrones de Diseño: Ej

- Adapter
  - Convierte el interfaz de una clase en otro que es el que el cliente espera. Se permite así que las clases colaboren juntas lo que no podría hacerse en otro caso por la incompatibilidad de los interfaces.
  - Interacción también conocida como *wrapper*
  - Se crea una clase nueva con acceso a las clases del interfaz actual y la del deseado (supuestamente ya existente).
  - La herencia se usa para hacer que la nueva clase herede de ambas luego todos sus objetos pertenecen a ambos tipos.

Patrones de Diseño: Ej

- Adapter



Patrones de Diseño: Catálogo

Ambito				
	Clases	Factory Method	Adapter (class)	Interpreter Templete Method
	Objetos	Abstract Factory Builder Prototype Singleton	Adapter (object) Bridge Composite Decorator Facade Flyweight Proxy	Chain of Responsability Command Iterator Mediator Memento Observer State Strategy Visitor

Patrones de Diseño

- Ventajas:
  - Son soluciones simples y técnicas.
  - Muy prácticos (deslumbran) y útiles.
- Desventajas:
  - Son soluciones concretas a problemas concretos.
  - Libro de recetas, lo que hace difícil averiguar el patrón adecuado ante un problema concreto. (LePlus)
  - No dejan huella: En una implementación es difícil saber que patrón se utilizó. (Ingeniería inversa).
  - Facilitan la reutilización del diseño pero no tanto la de la implementación.
  - No están formalizados (al menos no de forma simple).

Lenguajes de descripción de arquitecturas

- Sirven para expresar la estructura de las aplicaciones.
- Proporcionan los modelos, notaciones y herramientas que permiten describir los componentes y sus enlaces específicos.
- Arquitecturas estáticas y/o dinámicas.
- En ocasiones sirven para el prototipado rápido y verificación formal de propiedades.

Lenguajes de descripción de arquitecturas

- Misiones:
  - Gestionar los diseños de alto nivel, adaptarlos a implementaciones específicas y permitir la selección de patrones o paradigmas de arquitecturas especificados previamente.
  - Representar nuevos patrones de arquitecturas y nuevas formas de interacción entre los componentes, de forma que puedan ser reutilizados en diseños futuros.
  - Aportar un conjunto de herramientas formales para demostrar propiedades sobre los sistemas diseñados (seguridad, viveza, ...)
  - Aportar otro conjunto de herramientas de desarrollo para realizar implementaciones parciales de los sistemas a partir de su descripción.

Lenguajes de descripción de arquitecturas

- Lenguaje Unificado de Modelado (UML)
- Lenguajes de interconexión de módulos y de descripción de interfaz (CORBA-IDL)
- Lenguajes de descripción de arquitectura:
  - Unicon (Mary Shaw y colaboradores - CMU)
  - Wright (Allen y Garlan)
  - Darwin (Magee y Kramer - IC)
  - Rapide (Luckham)
  - C2 (Medvidovic)
  - LEDA (U. Málaga)