

Tema -1

Actividad No. 17

Tipo de clase: Conferencia #4

Título: Arquitectura

Semana: 5

Lugar: Aula.

Contenido

- Características de la arquitectura
- Los casos de uso y la arquitectura
- Descripción de la arquitectura
- Vistas de la arquitectura

Objetivo:

- Describir las diferentes vistas arquitectónicas a partir de los distintos flujos de trabajo

Introducción

El arquitecto crea la arquitectura junto con otros desarrolladores. Trabajan para conseguir un sistema que tendrá un alto rendimiento y una alta calidad, y será completamente funcionad, verificable, amigable para el usuario, fiable, de alta disponibilidad, preciso, extensible, tolerante a cambios, robusto, mantenible, portable, confiable, seguro, y económico. Ellos saben que han de convivir con esas restricciones y que tendrán que tomar soluciones de compromiso entre ellas - éste es el motivo por el que hay un arquitecto-. El arquitecto posee la responsabilidad técnica más importante en estos aspectos y selecciona entre patrones de arquitectura y entre productos para establecer las dependencias entre subsistemas para cada uno de esos distintos intereses. Aquí la separación de intereses significa la creación de un diseño donde un cambio en un subsistema no retumba en otros varios subsistemas.

Desarrollo

RUP Centrado en la Arquitectura. Diapo 1.

El concepto de arquitectura software incluye los aspectos estáticos y dinámicos más significativos del sistema. La arquitectura surge de las necesidades de la empresa, como las perciben los usuarios y los inversores, y se refleja en los casos de uso. Sin embargo, también se ve influida por muchos otros factores, como la plataforma en la que tiene que funcionar el software (arquitectura hardware, sistema operativo, sistema de gestión de base de datos, protocolos para comunicaciones en red), los bloques de construcción reutilizables de que se dispone

La función corresponde a los casos de uso y la forma a la arquitectura. Debe haber interacción entre los casos de uso y la arquitectura. Es un problema del tipo "el huevo y la gallina". Por un lado, los casos de uso deben encajar en la arquitectura cuando se llevan a cabo. Por otro lado, la arquitectura debe permitir el desarrollo de todos los casos de uso requeridos, ahora y en el futuro. En realidad, tanto la arquitectura como los casos de uso deben evolucionar en paralelo.

Por tanto, los arquitectos moldean el sistema para darle una forma. Es esta forma, la arquitectura, la que debe diseñarse para permitir que el sistema evolucione, no sólo en su desarrollo inicial, sino también a lo largo de las futuras generaciones. Para encontrar esa forma, los arquitectos deben trabajar sobre la comprensión general de las funciones clave, es decir, sobre los

casos de uso claves del sistema. Estos casos de uso clave pueden suponer solamente entre el 5 y el 10 por ciento de todos los casos de uso, pero son los significativos, los que constituyen las funciones fundamentales del sistema. Los casos de usos deben de poder desplegarse en una arquitectura estable. A su vez la arquitectura debe permitir el soporte de los casos de uso.

Que es la arquitectura ?. Diapo 2

La arquitectura de software es el conjunto de decisiones significativas sobre la organización de un sistema, la selección de los elementos estructurales y sus interfaces de los cuales el sistema está compuesto junto con su comportamiento. Describe los cimientos del sistema que son necesarios como base para comprenderlo, desarrollarlo y producirlo económicamente. La misma se representa en 4+1 vistas arquitectónicas.

La arquitectura paso a paso. (Diapo 3)

Como un edificio, un sistema software es una única entidad, pero al arquitecto del software y a los desarrolladores les resulta útil presentar el sistema desde diferentes perspectivas para comprender mejor el diseño. Estas perspectivas son vistas del modelo del sistema. Todas las vistas juntas representan la arquitectura.

La arquitectura software abarca decisiones importantes sobre:

- La organización del sistema software.
- Los elementos estructurales que compondrán el sistema y sus interfaces, junto con sus comportamientos, tal y como se especifican en las colaboraciones entre estos elementos.
- La composición de los elementos estructurales y del comportamiento en subsistemas progresivamente más grandes.
- El estilo de la arquitectura que guía esta organización: los elementos y sus interfaces, sus colaboraciones y su composición.

Sin embargo, la arquitectura software está afectada no sólo por la estructura y el comportamiento, sino también por el uso, la funcionalidad, el rendimiento, la flexibilidad, la reutilización, la facilidad de comprensión, las restricciones y compromisos económicos y tecnológicos, y la estética.

¿Donde se centra la arquitectura?

En la fase de Elaboración.

¿Por qué es necesaria la arquitectura? Diapo 6

Se necesita una arquitectura para:

- Comprender el sistema.
- Organizar el desarrollo.
- Fomentar la reutilización.
- Hacer evolucionar el sistema.

Comprensión del sistema.

Para que una organización desarrolle un sistema, dicho sistema debe ser comprendido por todos los que vayan a intervenir en él. El hacer que los sistemas modernos sean comprensibles es un reto importante por muchas razones:

- Abarcan un comportamiento complejo.
- Operan en entornos complejos.
- Son tecnológicamente complejos.

A menudo combinan computación distribuida, productos y plataformas comerciales (como sistemas operativos y sistemas gestores de bases de datos) y reutilizan componentes y marcos de trabajo.

Deben satisfacer demandas individuales y de la organización.

En algunos casos son tan grandes que la dirección tiene que dividir el trabajo de desarrollo en varios proyectos, que están, a menudo, separados geográficamente, añadiendo dificultades a la hora de coordinarlos.

Por otra parte, estos factores cambian constantemente. Todo esto añade dificultad potencial para comprender la situación.

Para realizar un desarrollo centrado en la arquitectura hay que prevenir estos fallos en la comprensión del sistema. Por consiguiente, el primer requisito que tiene lugar en una descripción de la arquitectura es que se debe capacitar a los desarrolladores, directivos, clientes y otros usuarios para comprender qué se está haciendo con suficiente detalle como para facilitar su propia participación.

Organización del desarrollo

Cuanto mayor sea la organización del proyecto software, mayor será la sobrecarga de comunicación entre los desarrolladores para intentar coordinar sus esfuerzos. Esta sobrecarga se incrementa cuando el proyecto está geográficamente disperso. Dividiendo el sistema en subsistemas, con las interfaces claramente definidas y con un responsable o un grupo de responsables establecido para cada subsistema, el arquitecto puede reducir la carga de comunicación entre los grupos de trabajo de los diferentes subsistemas, tanto si están en el mismo edificio como si están en diferentes continentes. Una "buena" arquitectura es la que define explícitamente estas interfaces, haciendo que sea posible la reducción en la comunicación. Una interfaz bien definida "comunica" eficientemente a los desarrolladores de ambas partes que necesitan saber sobre lo que los otros equipos están haciendo.

Fomento de la reutilización

Los desarrolladores capaces de reutilizar conocen el dominio del problema y qué componentes, especifica como adecuados la arquitectura. Los desarrolladores piensan en cómo conectar esos componentes para cumplir con los requisitos del sistema y realizar el modelo de casos de uso. Cuando tienen disponibles componentes reutilizables, los usan. Los componentes software reutilizables están diseñados y probados para encajar, y así el tiempo de construcción y el coste son menores. El resultado es predecible. En la estandarización del software se va avanzando con la experiencia pero esperamos que se incremente la "componentización".

La industria del software todavía tiene que alcanzar el nivel de estandarización que muchos dominios hardware han conseguido, pero las buenas arquitecturas y las interfaces bien definidas son pasos en esa dirección. Una buena arquitectura ofrece a los desarrolladores un andamio

estable sobre el que trabajar. El papel de los arquitectos es definir ese andamiaje y los subsistemas reutilizables que el desarrollador pueda utilizar. Se obtienen subsistemas reutilizables diseñándolos con cuidado para que puedan ser utilizados conjuntamente. Un buen arquitecto ayuda a los desarrolladores para que sepan dónde buscar elementos reutilizables de manera poco costosa, y para que puedan encontrar los componentes adecuados para ser reutilizados. El UML acelerará el proceso de "construcción de componentes", porque un lenguaje de modelado estándar es un prerequisite para construir componentes específicos del dominio que puedan estar disponibles para su reutilización.

Evolución del sistema

Si hay algo de lo que podemos estar seguros es de que cualquier sistema de un tamaño considerable evolucionará. Evolucionará incluso aunque aún esté en desarrollo. Más tarde, cuando esté en uso, el entorno cambiante provocará futuras evoluciones. Hasta que esto ocurra, el sistema debe ser fácil de modificar; esto quiere decir que los desarrolladores deberían ser capaces de modificar partes del diseño e implementación sin tener que preocuparse por los efectos inesperados que puedan tener repercusión en el sistema. En la mayoría de los casos, deberían ser capaces de implementar nuevas funcionalidades (es decir, casos de uso) en el sistema sin tener que pensar en un impacto dramático en el diseño e implementación existentes. En otras palabras, el sistema debe ser en sí mismo flexible a los cambios o tolerante a los cambios. Otra forma de enunciar este objetivo es afirmar que el sistema debe ser capaz de evolucionar sin problemas. Las arquitecturas del sistema pobres, por el contrario, suelen degradarse con el paso del tiempo y necesitan ser **parcheadas** hasta que al final no es posible actualizarlas con un coste razonable.

Casos de uso y la arquitectura Diapo 10

Ya hemos señalado que existe cierta interacción entre los casos de uso y la arquitectura. Si el sistema proporciona los casos de uso correctos - casos de uso de alto rendimiento, calidad y facilidad de utilización los usuarios pueden emplearlo para llevar a cabo sus objetivos. Pero, ¿cómo podemos conseguirlo? La respuesta, como ya hemos sugerido, es construir una arquitectura que nos permita implementar los casos de uso de una forma económica, ahora y en el futuro.

Vamos a clarificar cómo sucede esta interacción, observando primero qué influye en la arquitectura (véase la Figura 4.1) y después qué influye en los casos de uso.

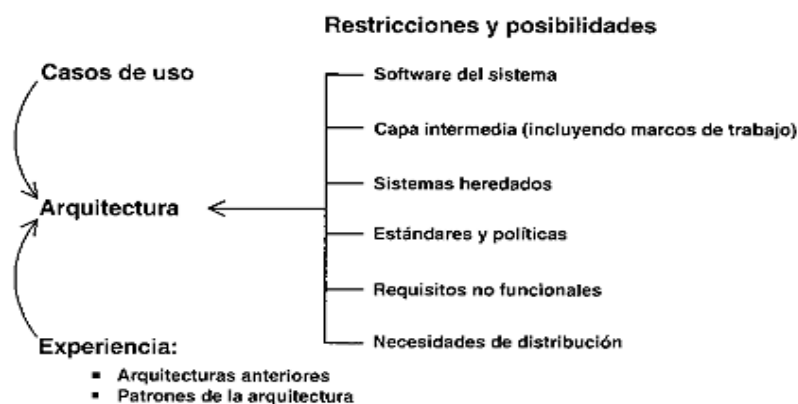


Figura 4.1. Existen diferentes tipos de requisitos y productos que influyen en la arquitectura, aparte de los casos de uso. También son de ayuda en el diseño de una arquitectura la experiencia de trabajos anteriores y las estructuras que podamos identificar como patrones de la arquitectura.

Como ya hemos dicho, la arquitectura está condicionada por los casos de uso que queremos que soporte el sistema; los casos de uso son directores de la arquitectura. Después de todo, queremos una arquitectura viable a la hora de implementar nuestros casos de uso. En las primeras iteraciones, elegimos unos pocos casos de uso, que pensamos que son los que nos ayudarán mejor en el diseño de la arquitectura. Estos casos de uso arquitectónicamente significativos incluyen los que son más necesarios para los clientes en la próxima versión y quizá para versiones futuras.

Sin embargo, la arquitectura no sólo se ve condicionada por los casos de uso arquitectónicamente significativos, sino también por los siguientes factores:

Sobre qué productos software del sistema queremos desarrollar, como sistemas operativos o sistemas de gestión de bases de datos concretos.

Qué productos queremos utilizar. Por ejemplo, tenemos que seleccionar un object request broker (ORB), que es un mecanismo para la conversión y envío de mensajes a objetos en entornos heterogéneos o un marco de trabajo independiente de la plataforma, es decir, un subsistema "prefabricado", para construir interfaces gráficas.

Qué sistemas heredados queremos utilizar en nuestro sistema. La utilización en nuestra arquitectura de un sistema heredado, como por ejemplo un sistema bancario existente, nos permite reutilizar gran parte de la funcionalidad existente, pero también tenemos que ajustar nuestra arquitectura para que encaje con el producto **antiguo**.

A qué estándares y políticas corporativas debemos adaptarnos. Por ejemplo, podemos elegir el Lenguaje de Definición de Interfaces para especificar todas las interfaces de las clases, o el estándar TMN de telecomunicaciones para especificar objetos en nuestro sistema.

Requisitos no funcionales generales (no específicos de los casos de uso), como los requisitos de disponibilidad, tiempo de recuperación, o uso de memoria.

Las necesidades de distribución especifican como distribuir el sistema, quizá a través de una arquitectura cliente/servidor.

¿Qué es primero la arquitectura o los casos de uso? (Diapo 11)

La arquitectura se desarrolla en iteraciones de la fase de elaboración. La siguiente podría ser una aproximación simplificada, de algún modo ingenua. Comenzamos determinando un diseño de alto nivel para la arquitectura, a modo de una arquitectura en capas. Después formamos la arquitectura en un par de construcciones dentro de la primera iteración.

En la primera construcción, trabajamos con las partes generales de la aplicación, que son generales en cuanto al dominio, y que no son específicas del sistema que pensamos desarrollar. Decidimos qué nodos contendrá nuestro modelo de desarrollo y cómo deben interactuar entre ellos. También decidimos cómo manejar los requisitos generales no funcionales, así como la disponibilidad de estos requisitos. Con la primera pasada es suficiente para tener una visión general del funcionamiento de la aplicación.

En la segunda construcción, trabajamos con los aspectos de la arquitectura específicos de la aplicación. Escogemos un conjunto de casos de uso relevantes en cuanto a la arquitectura,

capturamos los requisitos, los analizamos, los diseñamos, los implementamos y los probamos. El resultado serán nuevos subsistemas implementados como componentes del desarrollo que soportan los casos de uso seleccionados. Pueden existir también algunos cambios en los componentes significativos de la arquitectura que implementamos en la primera entrega (cuando no pensamos en términos de casos de uso). Los componentes nuevos o cambiados se desarrollan para realizar los casos de uso, y de esta forma la arquitectura se adapta para ajustarse mejor a los casos de uso.

Entonces elaboraremos otra construcción, y así sucesivamente hasta terminar con las iteraciones. Si este final de las iteraciones tiene lugar en el final de la fase de elaboración, habremos conseguido una arquitectura estable.

Cuando tenemos una arquitectura estable, podemos implementar la funcionalidad completamente realizando el resto de casos de uso durante la fase de construcción. Los casos de uso implementados durante la fase de construcción se desarrollan utilizando como entradas los requisitos de los clientes y de los usuarios, pero los casos de uso están también influenciados por la arquitectura elegida en la fase de elaboración.

Según vayamos capturando nuevos casos de uso, vamos utilizando el conocimiento que ya tenemos de la arquitectura existente para hacer mejor nuestro trabajo. Cuando calculamos el valor y el coste de los casos de uso que se sugieren, lo hacemos a la luz de la arquitectura que tenemos. Algunos casos de uso serán más fáciles de implementar, mientras que otros serán más difíciles.

Trabajadores que intervienen en la arquitectura (Diapo 12)

Flujo de trabajo Captura de requisitos

En este flujo de trabajo participa como trabajador el arquitecto para describir la vista de la arquitectura del modelo de casos de uso.

La vista de la arquitectura del modelo de casos de uso es una entrada importante para planificar las iteraciones.

Flujo de trabajo Análisis

El arquitecto es responsable de la integridad del modelo de análisis, garantizando que este sea correcto, consistente y legible como un todo. En sistemas grandes y complejos estas responsabilidades pueden requerir mas mantenimiento tras algunas iteraciones, y el trabajo que conlleva puede hacerse bastante rutinario. En estos casos, el arquitecto puede delegar ese trabajo a otro trabajador, posiblemente a un ingeniero de componentes de alto nivel. Sin embargo el arquitecto sigue siendo responsable de lo que es significativo para la arquitectura, la descripción de la arquitectura. El otro trabajador será responsable del paquete de nivel superior del modelo de análisis, que debe ser conforme con la descripción de la arquitectura.

El modelo de análisis es correcto cuando realiza la funcionalidad descrita en el modelo de casos de uso, y solo esa funcionalidad.

El arquitecto es también el responsable de la arquitectura del modelo de análisis, es decir de la existencia de sus partes significativas para la arquitectura tal y como se muestran en la vista de

la arquitectura del modelo. Recuérdese que esta vista es una parte de la descripción de la arquitectura del sistema.

Obsérvese que el arquitecto no es responsable del desarrollo y mantenimiento continuo de los diferentes artefactos del modelo de análisis. Estos son responsabilidad de los correspondientes ingenieros de casos de uso e ingeniero de componentes.

Flujo de trabajo Diseño

El arquitecto es responsable de la integridad de los modelos de diseño y de despliegue, garantizando que los modelos sean correctos, consistentes y legibles en su totalidad. Al igual que en el modelo de análisis, puede incluirse, para sistemas grandes y complejos, un trabajador aparte para asumir las responsabilidades del subsistema de mas alto nivel del modelo de diseño.

Los modelos son correctos cuando realizan la funcionalidad, y solo la funcionalidad, descrita en el modelo de casos de uso, en los requisitos adicionales, y en el modelo de análisis. El arquitecto también es responsable de la arquitectura de los modelos de diseño y despliegue, es decir, de la existencia de sus partes significativas para la arquitectura, como se muestran en las vistas arquitectónicas de esos modelos. Recuérdese que esas vistas son parte de la descripción de la arquitectura del sistema.

Obsérvese que el arquitecto no es responsable del desarrollo y mantenimiento continuos de los distintos artefactos del modelo de diseño. Estos se encuentran bajo la responsabilidad de los correspondientes ingenieros de casos de uso y de componentes.

Descripción de la arquitectura Diapo 13

La descripción de la arquitectura es un extracto o, en nuestros términos, un conjunto de vistas - quizá con una reescritura cuidada para hacerlas más legibles- de los modelos que están en la línea base de la arquitectura. Estas vistas incluyen los elementos arquitectónicamente significativos. Por supuesto, muchos de los elementos del modelo que son parte de la línea base de la arquitectura aparecerán también en la descripción de la arquitectura. Sin embargo, no lo harán todos ellos, debido a que para obtener una línea base operativa puede ser necesario el desarrollo de algunos elementos del modelo que no son arquitectónicamente interesantes, pero que se necesitan para generar código ejecutable. Debido a que la línea base de la arquitectura no sólo se usa para desarrollar una arquitectura, sino también para especificar los requisitos del sistema en un nivel que permita el desarrollo de un plan detallado, el modelo de casos de uso de esta línea base puede contener también más casos de uso aparte de los interesantes desde el punto de vista de la arquitectura.

La descripción de la arquitectura debe mantenerse actualizada a lo largo de la vida del sistema para reflejar los cambios y las adiciones que son relevantes para la arquitectura. Estos cambios son normalmente secundarios y pueden incluir:

- La identificación de nuevas clases abstractas e interfaces.
- La adición de nueva funcionalidad a los subsistemas existentes.
- La actualización a nuevas versiones de los componentes reutilizables.
- La reordenación de la estructura de procesos.

Puede que tengamos que modificar la propia descripción de la arquitectura. Pero su tamaño no debe crecer. Sólo se actualiza para ser relevante (véase la Figura 4.6).

Como dijimos anteriormente, la descripción de la arquitectura presenta vistas de los modelos. Esto incluye casos de uso, subsistemas, interfaces, algunas clases y componentes, nodos y colaboraciones. La descripción de la arquitectura también incluye requisitos significativos para la arquitectura que no están descritos por medio de los casos de uso. Estos otros requisitos son no Funcionales y se especifican como requisitos adicionales, como aquellos relativos a la seguridad, e importantes restricciones acerca de la distribución y la concurrencia (Apéndice C: véase también la Sección 9.3.2).

La descripción de la arquitectura debería incluir también una breve descripción de la plataforma, los sistemas heredados, y el software comercial que se utilizará, como por ejemplo la invocación de métodos remotos de Java (Remoto Method Invocation. RMI) para la distribución de objetos. Es más, es importante describir los marcos de trabajo que implementan mecanismos (Apéndice C; véase también 9.5.1.4) genéricos, como el almacenamiento y recuperación de un objeto en una base de datos relacional.

Hemos estado hablando bastante tiempo sobre lo que es la arquitectura sin ofrecer un ejemplo significativo. Presentamos ahora un ejemplo concreto de la apariencia de una descripción de arquitectura. Sin embargo, antes tenemos que explicar por qué no es fácil hacerlo.

Recuérdese que la descripción de la arquitectura es sencillamente un extracto adecuado de los modelos del sistema (es decir, no añade nada nuevo). La primera versión de la descripción de la arquitectura es un extracto de la versión de los modelos que tenemos al término de la fase de elaboración en el primer ciclo de vida. Dado que no intentamos hacer una reescritura más legible de esos extractos, la descripción de la arquitectura se parece mucho a los modelos normales del sistema. Esta apariencia significa que la vista de la arquitectura del modelo de casos de uso es muy parecida a un modelo de casos de uso normal. La única diferencia reside en que la vista de la arquitectura sólo contiene los casos de uso significativos para la arquitectura (véase la Sección 12.6.2), mientras que el modelo de casos de uso final contiene todos los casos de uso. Lo mismo ocurre con la vista de la arquitectura del modelo de diseño. Es igual que un modelo de diseño, pero sólo representa los casos de uso que son interesantes para la arquitectura.

Otra razón por la cual es difícil ofrecer un ejemplo es que sólo es interesante hablar de la arquitectura en sistemas reales, y cuando queremos hablar aquí sobre un sistema en detalle, debe ser por necesidad un sistema pequeño. Sin embargo, vamos a emplear el ejemplo del CA del Capítulo 3 para ilustrar lo que podrían llevar las vistas de la arquitectura. Lo haremos comparando lo que debería estar en las vistas y lo que debería estar en los modelos completos del sistema.

Vistas 4+1 de la arquitectura

- Vista de casos de uso
- Vista lógica
- Vista de procesos
- Vista de despliegue
- Vista de implementación

Vista de casos de uso

Esta vista representa un subconjunto del artefacto Modelo de casos de uso y lista los casos de usos o escenarios del modelo de casos de uso más significativos, con las funcionalidades centrales del sistema. Si el sistema se hace extenso entonces se debería organizarse en paquetes, lo cual facilitaría la comprensión de la vista de casos de uso. Aquí se representa el diagrama de casos de uso de los mismos con una breve descripción de cada uno.

Vista lógica

Esta vista representa un subconjunto del artefacto Modelo de diseño, la cual representas los elementos de diseño más importantes para la arquitectura del sistema. Este describe las clases más importantes, su organización en paquetes y subsistemas, y estos a su vez en capas. También describe las realizaciones de casos de uso más importantes como por ejemplo las que describen aspectos dinámicos del sistema.

Vista de procesos

Esta vista suministra una base para la comprensión de la organización de los procesos de un sistema, ilustrados en el mapeo de las clases y subsistemas en procesos e hilos. Solo suele usarse cuando el sistema presenta procesos concurrentes o hilos.

Vista de despliegue

Esta vista suministra una base para la comprensión de la distribución física de un sistema a través de nodos. Suele utilizarse cuando el sistema está distribuido. Esto incluye la asignación de tareas provenientes de la vista de procesos en los nodos.

Vista de implementación

Esta vista describe la descomposición del software en capas y subsistemas de implementación. También provee una vista de la trazabilidad de los elementos de diseño de la vista lógica ahora para la implementación.

Esta contiene:

- Una enumeración de los subsistemas.
- Diagramas de componentes que ilustran la organización en capas y jerarquías de los subsistemas.
- Dependencia entre subsistemas.

Patrones

Un patrón es una solución a un problema de diseño que aparece con frecuencia, estos incluyen reglas y líneas a seguir para la organización de un sistema.

Existen patrones aplicables a cada una de las diferentes vistas de la arquitectura.

Utilización de patrones de la arquitectura

Las ideas del arquitecto Christopher Alexander sobre cómo los "lenguajes de patrones" se utilizan para sistematizar principios y practicas importantes en el diseño de edificios y comunidades, han inspirado a muchos miembros de la comunidad de la orientación a objetos a definir, coleccionar y probar una gran variedad de patrones software. La "comunidad de patrones" define un patrón como "una solución a un problema de diseño que aparece con frecuencia". Muchos de

los patrones de diseño están documentados en libros, que presentan los patrones utilizando plantillas estándar. Estas plantillas asignan un nombre a un patrón y presentan un resumen de los problemas y las fuerzas que lo hacen surgir, una solución en términos de colaboración de clases participantes e interacción entre objetos de esas clases. Las plantillas también proporcionan ejemplos de cómo se utiliza el patrón en algunos lenguajes de programación, junto con variantes del patrón, un resumen con las ventajas y las consecuencias de la utilización de patrones y referencias a estos. Según Alexander, sería bueno que los ingenieros de software aprendiesen los nombres y el objetivo de muchos patrones estándar, y que los aplicasen para hacer diseños mejores y más comprensibles. Existen patrones de diseño como Facade, Decorator, Proxy Observer, Strategy y Visitor ampliamente citados y utilizados.

Los patrones de las arquitecturas se utilizan de una forma parecida pero se centran en estructuras e interacciones de grano más grueso, entre subsistemas e incluso entre sistemas.

Existen muchos patrones de arquitectura, pero aquí sólo trataremos brevemente algunos de los más interesantes.

Patrones arquitectónicos generales.

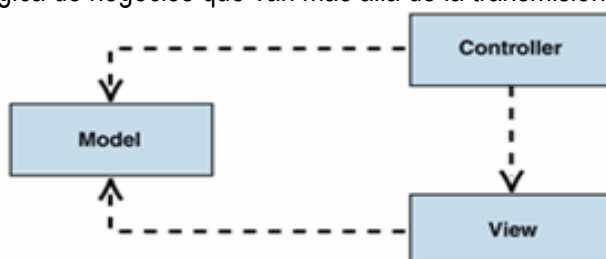
Un estilo arquitectónico o variante arquitectónica define a una familia de sistemas informáticos en términos de su organización estructural. Un estilo arquitectónico describe componentes y las relaciones entre ellos con las restricciones de su aplicación, la composición asociada y el diseño para su construcción.

Los sistemas empresariales distribuidos pueden agrupar los siguientes estilos arquitectónicos:

- Modelo-Vista-Controlador (MVC)
- Arquitecturas en Capas
- Arquitecturas Orientadas a Objetos
- Arquitecturas Basadas en Componentes
- Arquitecturas Orientadas a Servicios

Modelo-Vista-Controlador (MVC)

Un propósito común en numerosos sistemas es el de tomar datos de un almacenamiento y mostrarlos al usuario. Luego que el usuario introduce modificaciones, las mismas se reflejan en el almacenamiento. Dado que el flujo de información ocurre entre el almacenamiento y la interfaz, una tentación común, un impulso espontáneo (hoy se llamaría un anti-patrón) es unir ambas piezas para reducir la cantidad de código y optimizar el rendimiento. Sin embargo, esta idea es antagónica al hecho de que la interfaz suele cambiar, o acostumbra depender de distintas clases de dispositivos (aplicaciones de ventana, navegadores, dispositivos inalámbricos); la programación de interfaces de HTML requiere habilidades muy distintas de la programación de lógica de negocios. Otro problema es que las aplicaciones tienden a incorporar lógica de negocios que van más allá de la transmisión de datos.



El patrón conocido como Modelo-Vista-Controlador (MVC) separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres clases diferentes:

Modelo: Administra el comportamiento y los datos del dominio de aplicación, responde a requerimientos de información sobre su estado (usualmente formulados desde la vista) y responde a instrucciones de cambiar el estado (habitualmente desde el controlador).

Vista: Maneja la visualización de la información.

Controlador: Controla el flujo entre la vista y el modelo (los datos).

Tanto la vista como el controlador dependen del modelo, el cual no depende de las otras clases. Esta separación permite construir y probar el modelo, independientemente de la representación visual

Entre las ventajas del estilo Modelo-Vista-Controlador están las siguientes:

Soporte de múltiples vistas: Dado que la vista se halla separada del modelo y no hay dependencia directa del modelo con respecto a la vista, la interfaz de usuario puede mostrar múltiples vistas de los mismos datos simultáneamente. Por ejemplo, múltiples páginas de una aplicación Web pueden utilizar el mismo modelo de objetos mostrado de maneras diferentes.

Adaptación al cambio: Los requerimientos de interfaz de usuario tienden a cambiar con mayor rapidez que las reglas de negocios. Los usuarios pueden preferir distintas opciones de representación, o requerir soporte para nuevos dispositivos como teléfonos celulares o PDAs. Dado que el modelo no depende de las vistas, agregar nuevas opciones de presentación generalmente no afecta al modelo.

Una desventaja que tiene este modelo es el costo de actualizaciones frecuentes: Si el modelo experimenta cambios frecuentes, por ejemplo, podría desbordar las vistas con una lluvia de requerimientos de actualización.

Arquitecturas en Capas

Este patrón define cómo organizar el modelo de diseño en capas, que pueden estar físicamente distribuidas, lo cual quiere decir que los componentes de una capa sólo pueden hacer referencia a componentes en capas inmediatamente inferiores. Este patrón es importante porque simplifica la comprensión y la organización del desarrollo de sistemas complejos, reduciendo las dependencias de forma que las capas más bajas no son conscientes de ningún detalle o interfaz de las superiores. Además, nos ayuda a identificar qué puede reutilizarse, y proporciona una estructura que nos ayuda a tomar decisiones sobre qué partes comprar y qué partes construir.

Principales estilos de arquitecturas estratificadas de las aplicaciones distribuidas contemporáneas:

- Arquitecturas de dos niveles
- Arquitecturas de tres niveles
- Arquitecturas de n niveles
- Arquitectura de dos niveles

En una aplicación tradicional de 2 niveles, la carga de procesamiento recae en la PC cliente mientras que el servidor actúa simplemente como controlador del tráfico entre la aplicación y los datos. Como resultado, el rendimiento de la aplicación no solo sufre debido a los recursos limitados del PC sino que el tráfico de la red también tiende a aumentar. Cuando la aplicación completa es procesada en una PC, la aplicación es forzada a realizar múltiples peticiones de datos antes incluso de presentar algo al usuario. Estas múltiples peticiones de bases de datos pueden sobrecargar la red.

Otro problema típico relacionado con el enfoque de dos niveles es el del mantenimiento; incluso el menor cambio realizado a una aplicación puede conllevar a una completa alteración en la base de usuario. Aunque sea posible automatizar el proceso hay que enfrentarse a la actualización de cada instalación de cliente, es más, algunos usuarios pueden que no estén preparados para una alteración total y posiblemente ignoren los cambios mientras que otro grupo puede que insista en realizar los cambios de inmediato. Esto puede provocar que diferentes instalaciones de cliente utilicen diferentes versiones de la aplicación.

Arquitectura de tres niveles

Para enfrentarse a estos temas, la comunidad de software desarrolló la noción de una arquitectura de tres niveles. La aplicación se divide en tres capas lógicas distintas, cada una de ellas con un grupo de interfaces perfectamente definido. La primera capa se denomina capa de presentación y normalmente consiste en una interfaz gráfica de usuario de algún tipo. La capa intermedia, o capa de empresa, consiste en la aplicación o lógica de empresa, y la tercera capa, la capa de datos, contiene los datos necesarios para la aplicación.

La capa intermedia (lógica de aplicación) es básicamente el código al que recurre la capa de presentación para recuperar los datos deseados. La capa de presentación recibe entonces los datos y los formatea para su presentación. Esta separación entre la lógica de aplicación de la interfaz de usuario añade una enorme flexibilidad al diseño de la aplicación. Pueden construirse y desplegarse múltiples interfaces de usuario sin cambiar en absoluto la lógica de aplicación siempre que esta presente una interfaz claramente definida a la capa de presentación.

La tercera capa contiene los datos necesarios para la aplicación. Estos datos consisten en cualquier fuente de información, incluido una base de datos de empresa como Oracle o Sybase, un conjunto de documentos XML o incluso un servicio de directorio como el servidor LDAP. Además del tradicional mecanismo de almacenamiento relacional de bases de datos, existen muchas fuentes diferentes de datos de empresa a las que pueden acceder las aplicaciones.

Sin embargo, todavía no se ha finalizado en la subdivisión de la aplicación. Se puede dar un paso más para crear una arquitectura de n niveles.

Arquitectura de N niveles

Una arquitectura de n niveles se descompone en las siguientes partes:

Una interfaz de usuario que maneja la interacción del usuario con la aplicación, esta puede ser un navegador Web que ha ejecutado un pedido a través de un cortafuegos, una aplicación de escritorio o incluso un dispositivo inalámbrico.

Una lógica de presentación que define lo que muestra la interfaz de usuario y cómo son gestionadas las demandas del usuario. Dependiendo de las interfaces de usuario que se mantengan, puede que sea necesario contar con versiones de la lógica de presentación ligeramente diferentes para satisfacer al cliente adecuadamente.

Una lógica de negocio que modela las reglas de negocio de la aplicación, a menudo a través de la interacción con los datos de la aplicación.

Servicios de infraestructura que proporcionan la funcionalidad adicional requerida por los componentes de la aplicación, tales como mensajería y apoyo transaccional.

La capa de datos donde residen los datos de la empresa.

Un sistema con una arquitectura en capas pone a los subsistemas de aplicación individuales en lo más alto. Estos se construyen a partir de subsistemas en las capas más bajas, como son los marcos de trabajo y las bibliotecas de clases. Observemos la Figura 4.5. La capa general de aplicación contiene los subsistemas que no son específicos de una sola aplicación, sino que pueden ser reutilizados por muchas aplicaciones diferentes dentro del mismo dominio o negocio. La arquitectura de las dos capas inferiores puede establecerse sin considerar los casos de uso debido a que no son dependientes del negocio. La arquitectura de las dos capas superiores se crea a partir de los casos de uso significativos para la arquitectura (estas capas son dependientes del negocio).

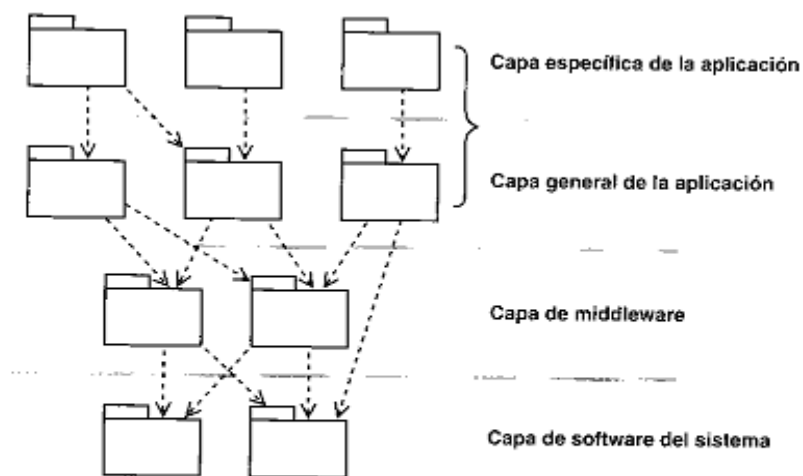


Figura 4.5. La arquitectura en capas organiza los sistemas en capas de subsistemas.

Una capa es un conjunto de subsistemas que comparten el mismo grado de generalidad y de volatilidad en las interfaces: las capas inferiores son de aplicación general a varias aplicaciones y deben poseer interfaces más estables, mientras que las capas más altas son más dependientes de la aplicación y pueden tener interfaces menos estables. Debido a que las capas inferiores cambian con menor frecuencia, los desarrolladores que trabajan en las capas superiores pueden construir sobre capas inferiores estables. Subsistemas en diferentes capas pueden reutilizar casos de uso, otros subsistemas de más bajo nivel, clases, interfaces, colaboraciones, y componentes de las capas inferiores. Podemos aplicar sobre un mismo sistema muchos patrones de arquitectura. Los patrones que estructuran el modelo de despliegue (es decir, Client/Server, Three-Tier, o Peer-to-Peer) pueden combinarse con el patrón Layers, lo cual nos ayuda a estructurar el modelo de diseño. Los patrones que tratan estructuras en diferentes

modelos son a menudo independientes unos de otros. Incluso los patrones que tratan el mismo modelo suelen poder combinarse bien mutuamente. Por ejemplo, el patrón Broker se combina correctamente con el patrón Layers, y ambos se utilizan en el modelo de diseño. El patrón Broker se encarga de cómo tratar con la distribución transparente de objetos, mientras que el patrón Layers nos indica cómo organizar el diseño entero. De hecho, el Patrón Broker puede interpretarse como un subsistema en la capa intermedia.

Arquitecturas Orientadas a Objetos

Características de las arquitecturas Orientadas a Objetos (OO):

Los componentes de este estilo se basan en principios OO: encapsulamiento, herencia y polimorfismo. Son así mismo las unidades de modelado, diseño e implementación, y los objetos y sus interacciones el centro de las incumbencias en el diseño de la arquitectura y en la estructura de la aplicación.

Las interfaces están separadas de las implementaciones. En general la distribución de objetos es transparente, y en el estado de arte de la tecnología apenas importa si los objetos son locales o remotos. El mejor ejemplo de OO para sistemas distribuidos es Common Object Request Broker

Architecture (CORBA), en la cual las interfaces se definen mediante Interface Description Language (IDL); un Object Request Broker media las interacciones entre objetos clientes y objetos servidores en ambientes distribuidos.

Entre las cualidades de esta arquitectura, la más básica concierne a que se puede modificar la implementación de un objeto sin afectar a sus clientes. Así mismo es posible descomponer problemas en colecciones de agentes en interacción. Además, por supuesto (y esa es la idea clave), un objeto es ante todo una entidad reutilizable en el entorno de desarrollo.

Entre las limitaciones, el principal problema de este patrón se manifiesta en el hecho de que para poder interactuar con otro objeto a través de una invocación de procedimiento, se debe conocer su identidad. La consecuencia inmediata de esta característica es que cuando se modifica un objeto (por ejemplo, se cambia el nombre de un método, o el tipo de dato de algún argumento de invocación) se deben modificar también todos los objetos que lo invocan.

Arquitecturas Basadas en Componentes.

Un componente de software, es una unidad de composición con interfaces especificadas contractualmente y dependencias del contexto explícitas. Que sea una unidad de composición y no de construcción quiere decir que no es preciso confeccionarla: se puede comprar hecha o se puede producir para que otras aplicaciones de la empresa la utilicen en sus propias composiciones.

Pragmáticamente se puede también definir un componente como un artefacto diseñado y desarrollado de acuerdo ya sea con CORBA Component Model (CCM), JavaBeans y EJB (Enterprise JavaBeans) en J2EE y lo que alternativamente se llamó en Microsoft: DCOM (Modelo de Objeto Componente Distribuido), ActiveX, COM+ (Sucesor de DCOM) y luego .NET.

En un estilo de este tipo:

Las interfaces están separadas de las implementaciones, y las interfaces y sus interacciones son el centro de incumbencias en el diseño arquitectónico.

En cuanto a las restricciones, puede admitirse que una interfaz sea implementada por múltiples componentes. Usualmente, los estados de un componente no son accesibles desde el exterior. La evaluación dominante del estilo de componentes subraya su mayor versatilidad respecto del modelo de objetos, pero también su menor adaptabilidad comparado con el estilo orientado a servicios.

En J2EE, el estilo de componentes, en el contexto de las arquitecturas en capas, ha sido uno de los vectores tecnológicos más importantes en los últimos años. Las especificaciones de J2EE permiten construir componentes avanzados e ínter operar componentes y servicios a nivel de las tecnologías EJB y Servlet en el contexto mixto de Servicios de Componentes.

Arquitectura Orientada a Servicio (SOA)

Una arquitectura orientada a servicios (SOA) es un patrón en el que los recursos que están interconectados en una red se conciben como servicios accesibles por terceros a través de una interfaz estándar. En este modelo existen tres actores principales: el proveedor del servicio, el registro del servicio y el solicitante del servicio. Un componente en esta arquitectura podría considerarse como un servicio que puede ser publicado, descubierto e invocado de forma dinámica. La característica más importante de este modelo es el bajo grado de acoplamiento entre componentes junto a una mayor flexibilidad ante cambios futuros.

Las aplicaciones empresariales deben facilitar su integración. Deben motivar más la construcción de servicios que de aplicaciones. Estos servicios se encargarían de exponer una funcionalidad bien definida a la aplicación que la requiera. De esta manera, una aplicación final simplemente utiliza un conjunto de estos servicios que le añaden valor a su lógica particular y le presenta una interfaz al usuario final.

La Arquitectura Orientada a Servicios (SOA) define los servicios de los cuales estará compuesto el sistema, sus interacciones, y con qué tecnologías serán implementados. Las interfaces que utiliza cada servicio para exponer su funcionalidad son gobernadas por contratos que definen claramente el conjunto de mensajes soportados, su contenido y las políticas aplicables.

Un servicio debe ser una aplicación completamente autónoma e independiente. A pesar de esto, no es una isla, porque expone una interfaz de llamado basada en mensajes, capaz de ser accedida a través de la red. Generalmente, los servicios incluyen tanto lógica de negocio como manejo de estado (datos) relevantes a la solución del problema para el cual fueron diseñados. La manipulación del estado es gobernada por las reglas de negocio.

La comunicación hacia y desde el servicio, es realizada utilizando mensajes y no llamadas a métodos. Estos mensajes deben contener o referenciar toda la información necesaria para entenderlo. La idea es que haya el mínimo posible de llamadas entre el cliente y el servicio.

Ahora bien, esta arquitectura no se funda en la idea de cualquier servicio en general, comunicado de cualquier manera, sino que más específicamente va de la mano de la expansión de los Web Services.

(Un Web Service es un sistema de software diseñado para soportar interacción máquina-máquina sobre una red. Posee una interfaz descrita en un formato procesable por máquina (específicamente WSDL). Otros sistemas interactúan con el Web Service de una manera prescrita por su descripción utilizando mensajes SOAP, típicamente transportados usando

HTTP con una serialización en XML en conjunción con otros estándares relacionados con la Web.)

Vale la pena destacar la forma en la cual este estilo de arquitectura orientada a servicios redefine los modelos de ORPC (Llamadas a Procedimientos de Objetos Remotos) propios de las arquitecturas orientadas a objetos y componentes, y al hacerlo establece un modelo en el que es casi razonable pensar que cualquier entidad computacional podría llegar a conversar o a integrarse con cualquier otra.

Lo que hace diferentes a los Web services de otros mecanismos de RPC como Java RMI, CORBA o DCOM es que utiliza estándares de la Web para los formatos de datos y los protocolos de aplicación. Esto permite que las aplicaciones inter operen con mayor libertad, dado que las organizaciones ya seguramente cuentan con una infraestructura activa de HTTP y pueden implementar tratamiento de XML y SOAP en casi cualquier lenguaje y plataforma.

Otros Patrones:

Patrón Broker

El patrón Broker es un mecanismo genérico para la gestión de objetos distribuidos. Permite que los objetos hagan llamadas a otros objetos remotos a través de un gestor que redirige la llamada al nodo y al proceso que guardan al objeto deseado. Esta redirección se hace de manera transparente, lo cual quiere decir que el llamante no necesita saber si el objeto llamado es remoto. El patrón Broker suele utilizar el patrón de diseño Proxy. Que proporciona un objeto sustituto local con la misma interfaz que el objeto remoto para hacer transparentes el estilo y los detalles de la comunicación distribuida.

Arquitectura cliente/servidor:

Esta consiste en varios clientes distribuidos en diferentes nodos, conectados en red a uno o varios nodos servidores, donde el servidor puede servir a varios clientes a la vez. En los nodos clientes generalmente encontramos presentación de usuario y en los nodos servidores la lógica del negocio.

Arquitectura cliente grueso:

Es cuando toda la lógica del negocio y de la aplicación en general recae sobre la aplicación en el nodo cliente, excepto en la arquitectura en 2 capas, en la cual, los servicios de datos son separados en otro nodo.

Patrón (Blackboard)

Este patrón es utilizado, en varias aplicaciones que necesitan trabajar en conjunto para obtener una solución única, por ejemplo, una aplicación obtiene parte de una solución, esta antes de terminar su ejecución, la guarda en el blackboard, luego la otra aplicación extrae del blackboard esta solución a medias y continua a partir de ahí.

Bibliografía

- JACOBSON, Ivar; BOOCH, Grady, RUMBAUGH, James, "El Proceso Unificado de Desarrollo de Software".2000. Addison Wesley. CAPITULO 6.
- Rational Unified Process, Rational Suite 2003.