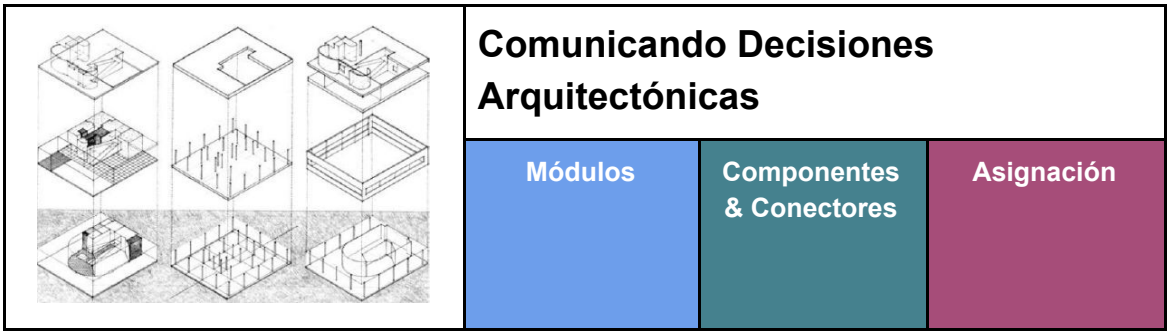
Versión: 0.91 (preliminar)
septiembre 2017



Basado en:

Documenting Software Architectures: Views and Beyond (2da Edición)

Software Engineering Institute - Addison-Wesley

Alcance:

El presente documento es una síntesis del trabajo publicado por el SEI. Su objetivo es introducir al lector en los tópicos presentados. Cada sección incluye una referencia al número de página de donde fue extraída de modo que el lector pueda profundizar en cada tema.

Contenidos:

¿Por qué Documentar una Arquitectura de Software?

Crear una arquitectura implica mucho más que dedicar un gran esfuerzo de análisis, diseño y trabajo en equipo si el proyecto no es comunicado de forma apropiada. Es tan importante crear un gran arquitectura como saber cómo comunicarla de manera tal que los involucrados en el proyecto puedan realizar correctamente su aporte. En muchos casos la documentación es tratada como trabajo complementario, contractual o por reglas que fijan estándares o procesos de la organización.

Entonces: *¿Cuáles son las razones para que el arquitecto deba crear documentación de calidad sobre el proyecto?*

Los mejores arquitectos producen la mejor documentación no porque "es requerida" sino porque saben que es esencial para crear un producto de alta calidad, predecible y que requiera el menor retrabajo posible.	

A su vez, también crean documentación de alta calidad para agregar valor a sí mismos. La documentación sirve para que el receptor pueda comprender los resultados de las decisiones de diseño se que tomaron y además asiste a que el proceso de diseño sea más gradual y sistemático.

Pag. 9

¿Qué es una Vista Arquitectónica?

Posiblemente el más importante concepto asociado a la arquitectura de software sea la vista. Una arquitectura de software es un entidad compleja que no puede ser descripta desde una sola dimensión.

Entonces: *¿Cuál vista es relevante?*

Depende de los objetivos. Diferentes vistas exponen diferentes atributos de calidad en diferentes grados. Por lo tanto, en función de los atributos de calidad que se deseen destacar determinarán la elección de la vista. Por ejemplo, una vista en capas explicará mejor acerca de la portabilidad de un sistema.

Se debe tener presente que diferentes vistas dan soporte a diferentes objetivos y usos. Cada vista enfatiza ciertos aspectos de un sistema mientras que puede estar minimizando o ignorando otros.

Pag. 22

¿Qué son los Estilos Arquitectónicos?

Los estilos permiten emplear conocimiento de diseño especializado a cierto sistema mediante estilos específicos para cada caso.

Ejemplo: Un estilo compuesto por *módulos* permite organizar un sistema según cierta funcionalidad específica. Otro ejemplo puede ser el estilo arquitectónico *cliente-servidor*, donde los elementos son: clientes, servidores y los protocolos de los conectores que representan su interacción.

Un sistema no está construido exclusivamente a partir de un solo estilo. Lo habitual es que un sistema sea una amalgama de diferentes estilos. A su vez, diferentes áreas de un sistema pueden exhibir diferentes estilos.

Por ejemplo, un sistema puede emplear un estilo de *tubería y filtro* (pipe-and-filter) para procesar datos de entrada y un estilo de *datos compartidos* (shared data) para persistir el resultado en una base de datos.

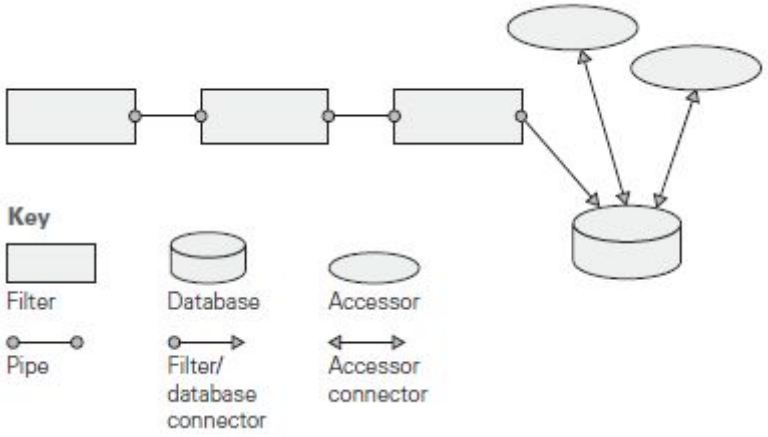


Figura 1: Un sistema que combina los estilos *tubería y filtro* con *datos compartidos*. El elemento *conector filtro/base de datos* es un elemento "puente".

Documentación de la Arquitectura y los Actores del Sistema

La documentación de arquitectura debe ser lo suficientemente abstracta para poder ser entendida rápidamente por nuevos actores. A su vez debe ser lo suficientemente concreta para servir de base para la construcción del sistema.

Diferentes roles involucrados al proyecto estarán interesados en la documentación arquitectónica del sistema. Ellos esperan que esa documentación los ayude en sus respectivos trabajos. Es esencial comprender el uso que le dará cada actor.

1. La arquitectura sirve como medio de educación: Nuevos miembros del equipo, analistas externos o eventualmente un nuevo arquitecto. En algunos casos el "nuevo integrante" puede ser el cliente que determine la aprobación del proyecto.

2. La arquitectura sirve como vehículo primario para la comunicación entre Interesados (stakeholders): Primero, la documentación provee de "compartimentos dedicados" para registrar decisiones de diseño tan pronto como se toman. Segundo, también puede servir como indicador de progreso o de trabajo pendiente. Y tercero, provee un marco de trabajo para abordar sistemáticamente el diseño de una arquitectura.

3. La arquitectura sirve de base para el análisis y la construcción de sistemas: Indica a los desarrolladores qué se debe implementar. Es una referencia para evaluar si el sistema cumple con los objetivos de calidad como por ejemplo seguridad, desempeño, usabilidad, disponibilidad y modificabilidad. Para crear código generado automáticamente.

Documentación de la Arquitectura y los Atributos de Calidad

Partiendo que la arquitectura se basa en lograr que el sistema posea ciertos atributos calidad y si uno de los principales usos de la documentación de arquitectura es servir de base para el análisis (para asegurar que la arquitectura alcanzará los atributos de calidad requeridos), entonces: *¿Dónde aparecen los atributos de calidad en la documentación?*

Existen cinco formas de ubicar atributos de calidad en la documentación de arquitectura:

1. La elección de un patrón o estilo arquitectónico tendrá asociado ciertos atributos de calidad, por ejemplo *cliente-servidor* es bueno para escalabilidad, *capas* es bueno

para portabilidad, *ocultamiento de información* es bueno para modificabilidad y *servicios* es bueno para interoperabilidad.

2. Elementos arquitectónicos individuales proveen cierto atributo de calidad asociado. Por ejemplo, los consumidores de servicios necesitan saber cuán rápido, seguro o confiable son. Esos atributos de calidad se encuentran en la documentación de la interfaz.

3. Los atributos de calidad habitualmente definen el "lenguaje" de objetivos que se desean alcanzar. Por ejemplo, *seguridad* involucra niveles de seguridad, autenticación, auditoría, firewalls. *Desempeño* posee asociadas palabras como capacidad de buffer, deadlines, períodos, tasa de eventos, temporizadores.

4. La documentación de arquitectura habitualmente posee un mapa a requerimientos que muestran cómo son abordados.

5. Cada atributo de calidad poseerá un conjunto de interesados o stakeholders que desearán saber si la arquitectura lo satisface. Se debe incluir alguna referencia en la introducción de modo tal que el interesado sepa dónde encontrará ese aspecto particular en la documentación.

Pag. 17

¿A qué nos referimos con Vistas?

Este enfoque no sólo apunta a documentar arquitectura, sino también a enfatizar el concepto de vista incluyendo información adicional para esa documentación.

Este "método" puede ser explicado a partir de las siguientes actividades:

1. Distinguir qué es lo que los involucrados (stakeholders) están buscando en la documentación.
2. Proveer información como registro de las decisiones de diseño en forma de vistas.
3. Verificar si la documentación resultante satisface las necesidades.
4. Distribuir la información en un medio que sea de utilidad para los involucrados.

Mientras que los ítems 3 y 4 son actividades centradas en la documentación, los ítems 1 y 2 son actividades que deberán ser realizadas en conjunto con el diseño de la arquitectura.

Pag. 19

La Documentación de la Arquitectura en un Entorno Ágil

Es un mito que el desarrollo ágil y la documentación son opuestos. Este enfoque provee una guía para documentar diferentes clases de información arquitectónica como: estructuras, elementos, relaciones, guías de estilos, contexto del sistema y otros.
Primero se debe decidir qué es de utilidad, para luego y en función de esa decisión, emplear alguna de las guías que se estudiarán a continuación.
En resumen, se trata de decidir qué vista es de utilidad y posee una equilibrada tasa de costo-efectividad según el objetivo.

Pag. 20

La Documentación de la Arquitectura y el Cambio

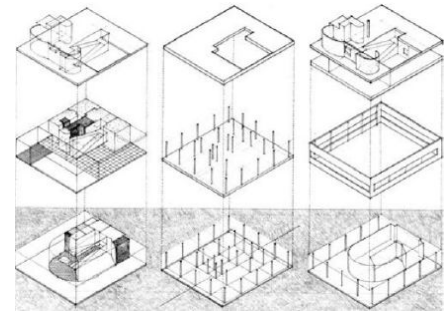
Un sistema puede cambiar en tiempo de ejecución (por ejemplo con la instalación de un plugin o la localización de un servicio) o como resultado de un ciclo de entrega-despliegue, ambos son casos que tienen algo en común respecto de la documentación: cambian más rápidamente que la documentación. Para estos casos se recomienda:

- Documentar qué es común para todas las versiones del sistema: Registrar las invariantes que serán persistentes en el tiempo respecto de la arquitectura del sistema.

- Documentar de qué manera la arquitectura puede cambiar: En el enfoque "Vistas y Más" se realiza en la "guía de invariantes".

- Hacer que el sistema capture su propia arquitectura-del-momento automáticamente: Si un sistema falla, el equipo de desarrollo deseará conocer exactamente qué configuración estaba en ejecución al momento que ocurrió el problema.

Pag. 20



Vistas de Arquitectura

Módulos:	Componentes & Conectores:	Asignación:
Descomposición	Flujo de datos	Despliegue
Uso	Call-Return	Instalación
Generalización	Basado en Eventos	Asignación
Capas	Repositorio	
Aspectos	Niveles	
Modelo de Datos		

A continuación en las siguientes se presentarán ejemplos de documentación de arquitectura tomados de sistemas reales.

Vista de Módulos

La vista de módulos se utiliza para:

- ➔ Definir los planos que darán lugar a la construcción del código.
- ➔ Simplificar el análisis.
- ➔ Planificar el desarrollo incremental.
- ➔ Dar soporte a la trazabilidad del análisis de los requerimientos.
- ➔ Explicar la funcionalidad del sistema y la estructura del código base.
- ➔ Dar soporte a la asignación de tareas, definición de cronograma y presupuesto.
- ➔ Mostrar la estructura de la información que será persistida.

Definiciones:

Elementos: Los módulos son unidades de implementación de software que proveen un conjunto coherente de responsabilidades.

Un módulo puede ser una estructura de software definida por un cierto lenguaje de programación o stored procedures de PL/SQL o simplemente agrupamientos en paquetes (Java) o namespaces (C#).

Los módulos pueden ser agregados y descompuestos. Cada uno de los estilos de módulos identifican un diferente conjunto de módulos y relaciones. Por ejemplo, el estilo de capas identifica módulos y los agrega en una relación autorizado-a-usar, mientras que la generalización los identifica y agrega basándose en atributos en común.

Responsabilidad: Es una declaración general acerca de un elemento de arquitectura y qué se espera con su contribución para con esa arquitectura. Incluye las acciones que ejecuta, el conocimiento que mantiene, las decisiones que toma o el rol que juega en conseguir que el sistema alcance a complir con la funcionalidad o los atributos de calidad esperados.

Relaciones:

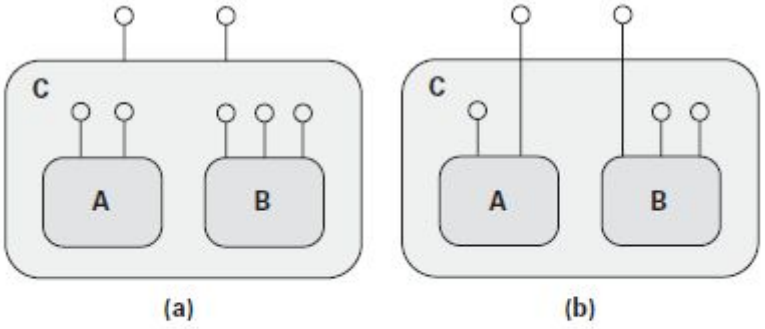
- **es-parte-de** define la relación parte de un todo entre un el submódulo (la parte) y el módulo que representa la agregación (el todo).
- **depende-de** define la relación de dependencia entre dos módulos.
- **es-un** define la relación de generalización/especialización entre un módulo más específico (el hijo) y uno más general (el padre).

Restricciones: Diferentes vistas de módulos pueden imponer restricciones topológicas específicas.

Propiedades: Son de utilidad para guiar la implementación o son la entrada para la documentación de soporte. El listado de propiedades puede variar, pero en términos generales debería estar integrado por:

- Nombre del módulo: Su nombre debería sugerir algo acerca de su rol en el sistema.

- Responsabilidad: Es una forma de identificar su rol en el sistema y establece una identidad más allá de su nombre.
- Visibilidad de la interfaz: Cuando un módulo posee submódulos, algunas interfaces pueden ser internas. Por ejemplo en la siguiente figura, ciertas interfaces provistas por los submódulos A y B no son visibles al exterior del módulo C.

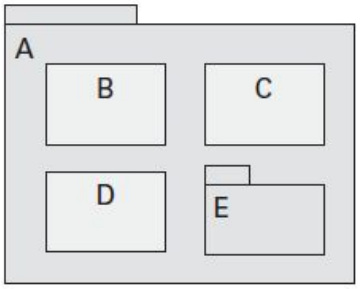


- Información para implementación: Es de utilidad incluir cierta información relacionada con su implementación y puede ser:
  - Mapeo con unidades de código fuente (un nombre de archivo).
  - Información para pruebas.
  - Información para gestión.
  - Restricciones de implementación.

Pag. 55

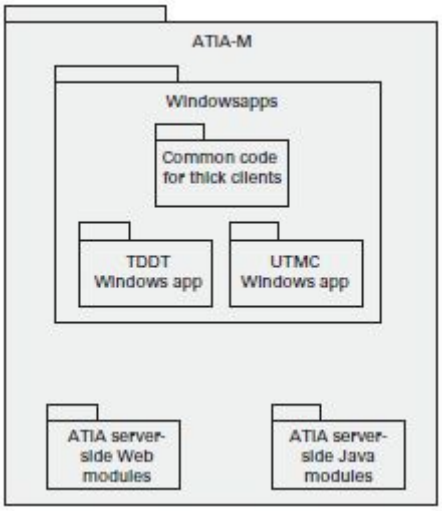
Vista de Módulos	➔ *Estilo de Descomposición*

Esta vista presenta las responsabilidades del sistema dividida en en partes intelectualmente manejables. Es apropiado para apoyar el proceso del aprendizaje del sistema y asignar tareas de desarrollo. También sirve para analizar el efecto de los cambios.

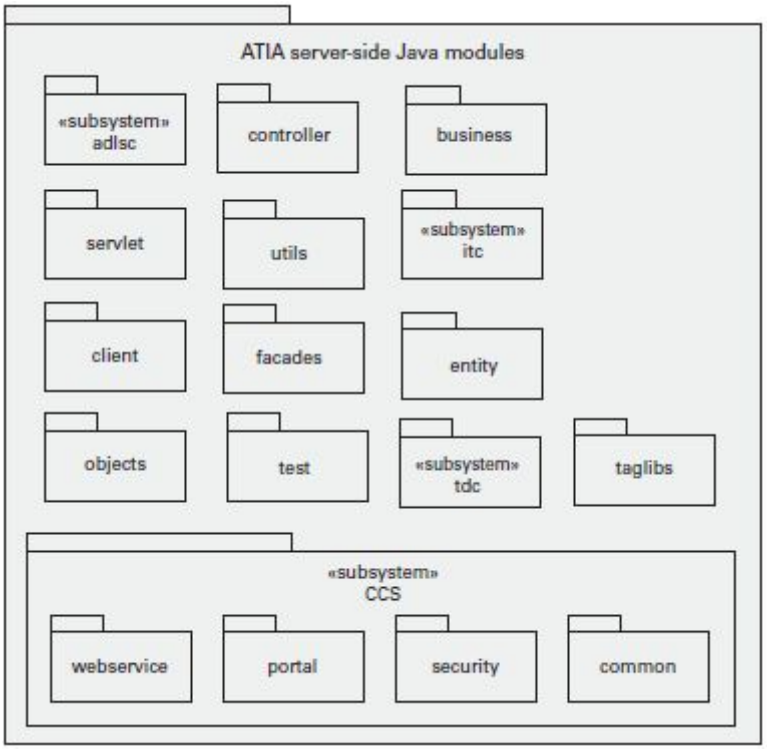
Notación	
	Se puede emplear la vista de paquetes de UML para representar módulos que contienen otros módulos. Un paquete puede contener clases y otros paquetes. La clase se representa como cajas (B, C y D) y los módulos como carpetas (A y E).

Ejemplo Sistema ATIA-M:

El sistema ATIA-M (Army Training Information Architecture-Migrated) es un sistema Java EE que da soporte al entrenamiento de la armada de EE.UU. Posee clientes livianos (thin clients) de escritorio desarrollados en C# que se comunican con el servidor (server-side) Java EE a través de Web services. El módulo Windowsapps se puede descomponer en 3 submódulos como se ve a continuación:



La descomposición del módulo ATIA server-side Java modules, muestra con mayor nivel de detalle su estructura interna:



Pag. 65

Vista de Módulos	➔ *Estilo de Uso*

El estilo “Usa” se emplea cuando la relación “depende-de” se especializa a “usa”. Un módulo usa otro módulo si su funcionamiento depende de otro. Por ejemplo:

El **Módulo A** *usa* al **Módulo B** si **A** *depende-de* la presencia y correcto funcionamiento de **B** para satisfacer sus propios requerimientos.

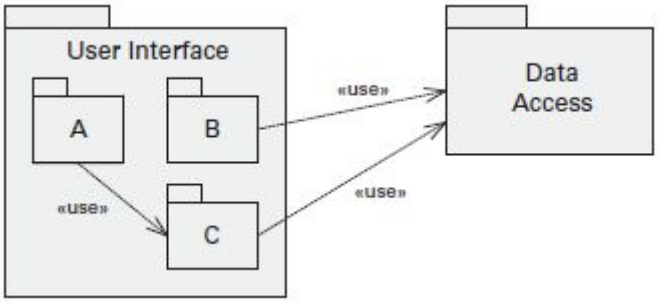
Este estilo indica al desarrollador qué otros módulos deben existir para que esa parte del sistema funcione correctamente. Sirve para planificar subconjuntos e incrementos del sistema en desarrollo.

Usos:

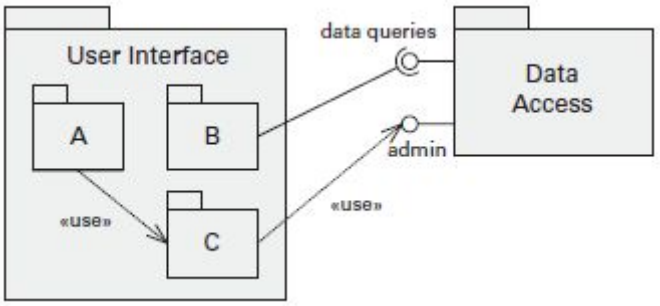
- Planificar desarrollo incremental y subconjuntos.
- Debugear y hacer testing
- Medir el efecto de cambios

Notación: Este estilo se puede documentar como una tabla de dos columnas, donde a la izquierda están los elementos que usan y a la derecha que son usados. También se puede emplear UML, donde los paquetes representan módulos y la relación “usa” se determina como una dependencia con el estereotipo <<use>>. En la siguiente figura el módulo User Interface posee una dependencia “usa” con el módulo Data Access:

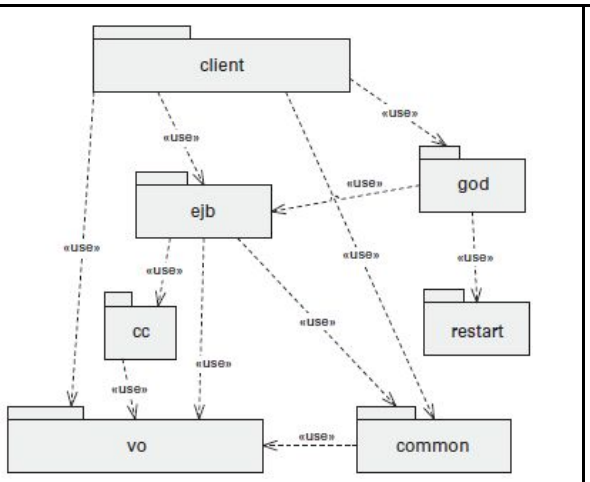
La descomposición de ambos módulos muestra con mayor nivel de detalle la relación “usa”:



Con UML se puede representar la relación “usa” y al mismo tiempo mostrar las interfaces de manera explícita. A continuación se muestra que el módulo Data Access posee dos interfaces, que son usadas por los módulos B y C respectivamente. Tanto el conector con el estereotipo <<use>> como el conector entre B y “data queries” determinan una relación de uso:



La relación “usa” se puede documentar como una matriz cuadrada, donde aparecen los módulos como filas y columnas:



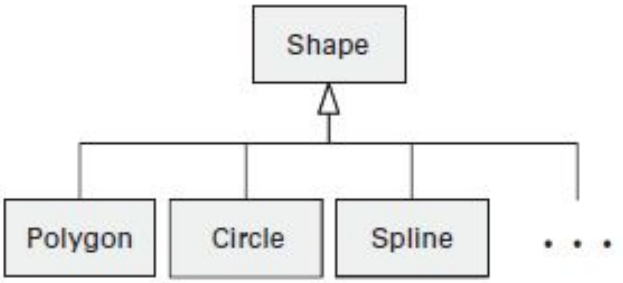
used module \ using module	client	ejb	cc	god	restart	common	vo
client	0	0	0	0	0	0	0
ejb	1	0	0	1	0	0	0
cc	0	1	0	0	0	0	0
god	1	0	0	0	0	0	0
restart	0	0	0	1	0	0	0
common	1	1	0	0	0	0	0
vo	1	1	1	0	0	1	0

Pag. 74

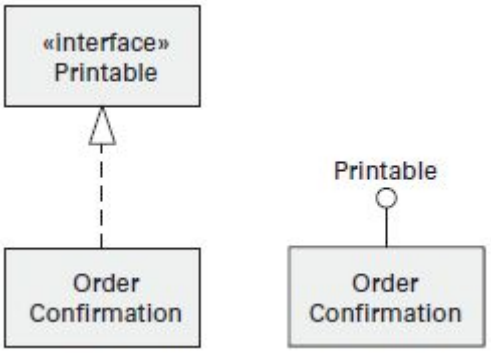
Vista de Módulos ➔ *Estilo de Generalización*

El estilo de generalización se emplea cuando existe una relación es-un. Es de utilidad cuando el arquitecto desea dar soporte a la extensión y evolución de arquitecturas o elementos individuales. Cuando los módulos poseen una relación de generalización, el módulo padre es la versión más general de los módulos hijos. Se pueden realizar extensiones a partir del agregado, quitado o modificación de módulos hijos. La generalización puede representar en la interfaz o la implementación. Los ciclos no están permitidos.

Notación: Con UML los módulos se muestran como clases o interfaces



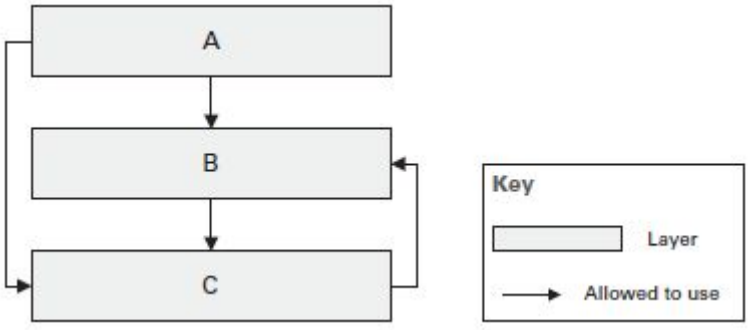
La realización (implementación) de una interfaz es un tipo de generalización. En UML puede ser representada como una línea punteada con una flecha desde el módulo hasta la interfaz o con el círculo que identifica a la interfaz conectado al módulo que la implementa:



Pag. 82

Vista de Módulos	➔ *Estilo de Capas*

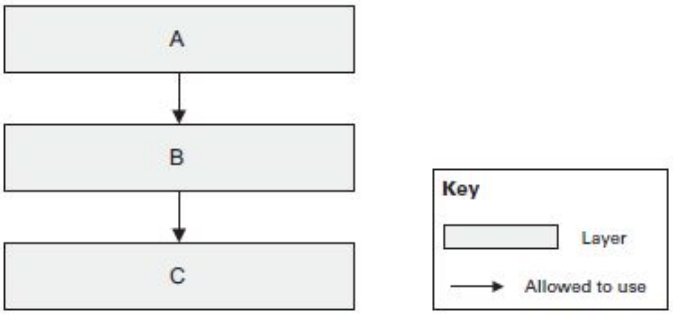
El estilo de capas al igual que todos los estilos de módulos, reflejan la división del software en unidades. Cada capa representa a un grupo de módulos que ofrecen un conjunto de servicios cohesivos. Las capas se crean con el propósito de interactuar de acuerdo a una estricta relación de orden y responden a la restricción *permitido-de-usar* (allowed-to-use) entre capas como se muestra en la figura a continuación:



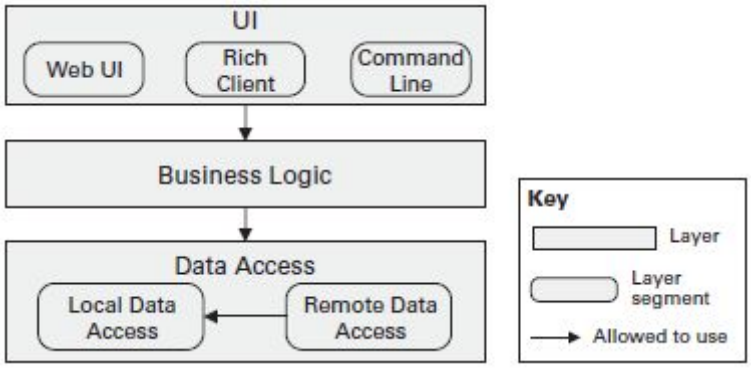
Las capas no pueden derivarse de la examinación del código fuente, son agrupamientos lógicos que se emplean para crear y comunicar una arquitectura pero que habitualmente no están explícitamente delimitados en el código fuente.
El Estilo de Capas se emplea para mostrar los atributos de modificabilidad y portabilidad a un sistema de software. La capa es la aplicabilidad del principio de ocultación de

información. En teoría, un cambio de una capa inferior no debería impactar en la capa superior.

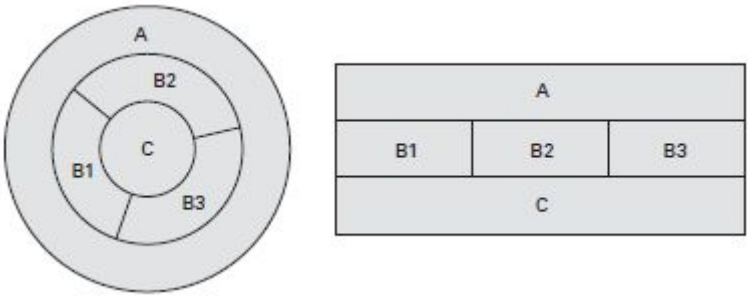
Notación: Las capas habitualmente se muestran como una pila de cajas. La relación *permitido-de-usar* se denota a través de la adyacencia geométrica y se lee de arriba hacia abajo:



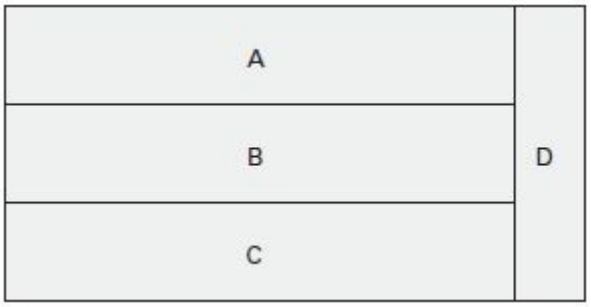
Capas segmentadas: Las capas pueden estar divididas en segmentos que denotan el refinamiento de una agregación de módulos como se muestra a continuación:



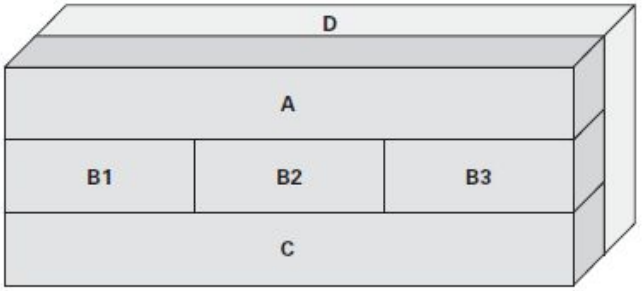
Una variación de capas es mostrando un conjunto de círculos o anillos concéntricos, donde el anillo interno corresponde a la capa más baja y el externo a la capa más alta:



Capas laterales (sidecar): Algunas arquitecturas necesitan mostrar que una capa es adyacente a todas, esto se muestra con una capa vertical y denota que esa capa puede ser usada por todas las demás o las otras pueden usar a esa capa:

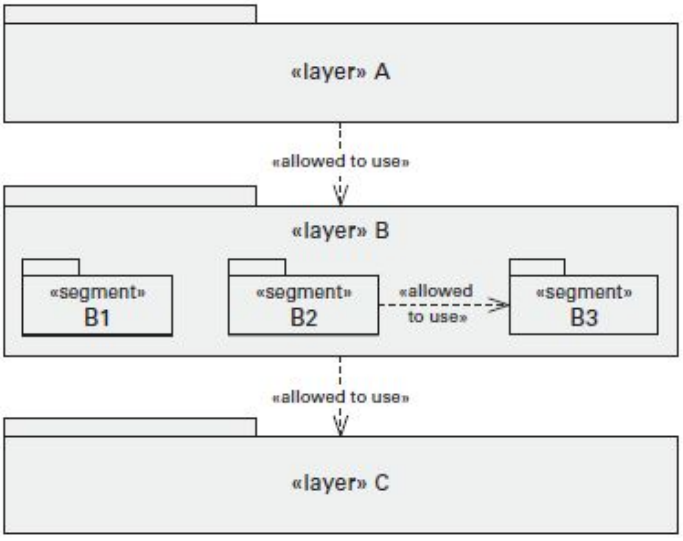


En algunos casos la arquitectura de capas se representa como una figura tridimensional, donde una capa es accesible a todas las otras:



También se pueden emplear diferentes tamaños y colores para distinguir si una capa corresponde a un equipo o denotar una característica en particular.

UML: Para representar capas con UML se puede hacer con el estereotipo “packages” tal como se muestra a continuación:



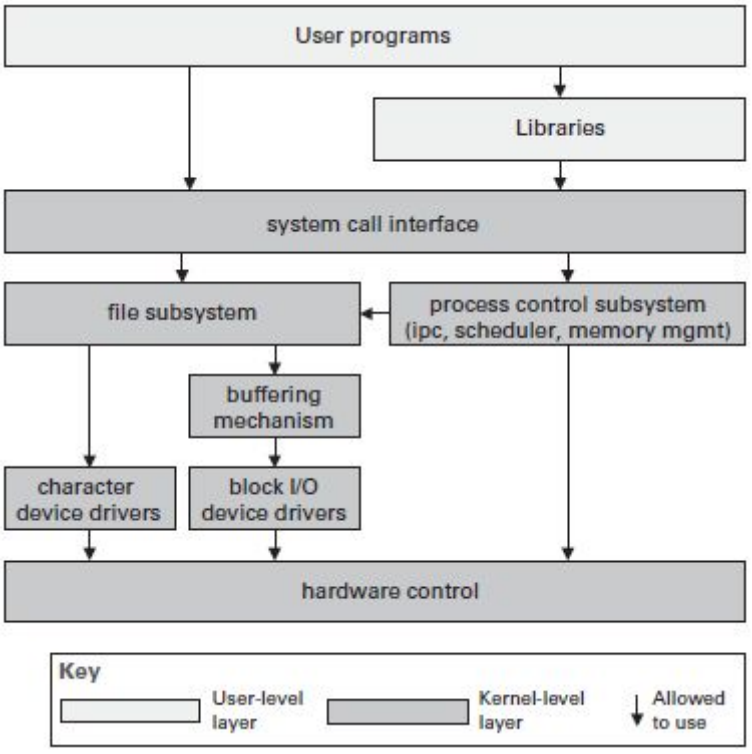
Cabe destacar que los accesos no representan transitividad, es decir, si el paquete 1 puede acceder al paquete 2 y el paquete 2 al 3, no implica que el paquete 1 pueda acceder al paquete 3.

Relación con otros estilos: Este diagrama de capas algunas veces es confundido con otros estilos, por ejemplo:

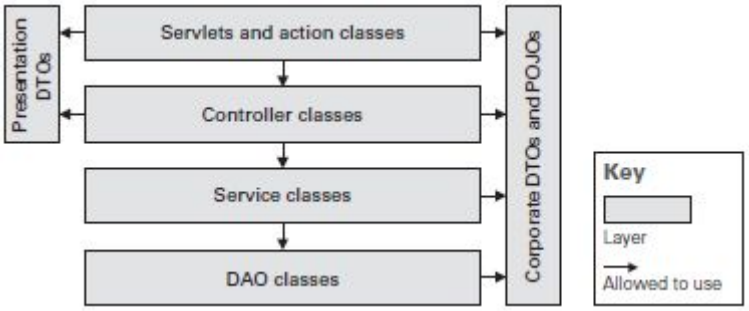
- Descomposición en módulos: Las capas y los módulos estan siempre relacionados pero no establecen una relación uno-a-uno. Una capa puede abarcar más de un módulo.
- Niveles (tiers): Habitualmente se confunden con niveles en una arquitectura de múltiples niveles (multi-tier). Las capas no son niveles. El estilo de capas muestra unidades de implementación agrupadas. En cambio, el estilo de múltiples niveles (multi-tier) es un estilo componente-y-conector porque los niveles congregan componentes de tiempo de ejecución.
- Estilo “Usa”: Como las capas expresan la relación permitido-de-usar existe una cercana correspondencia a ese estilo.

Ejemplos

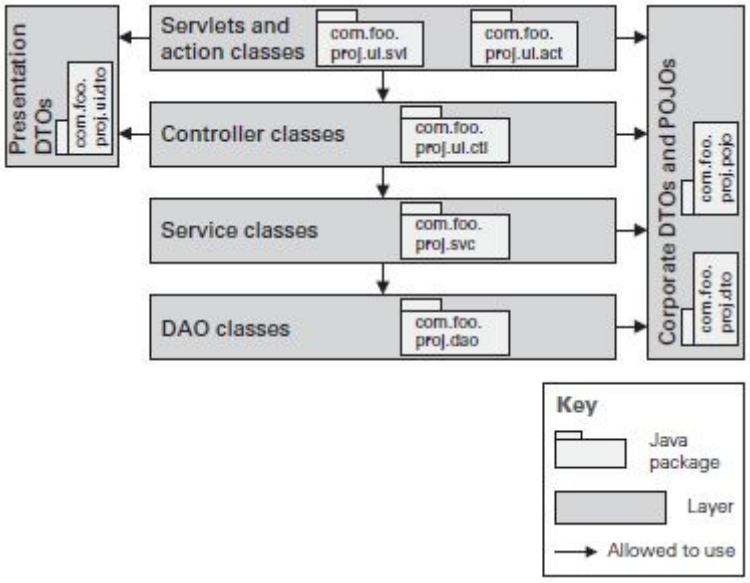
Sistema V de UNIX: Un ejemplo clásico de diseño por capas es el sistema operativo System V de UNIX, donde en la cima están los programas de usuario o bibliotecas que acceden al núcleo a través de llamadas al sistema:



Aplicación Java EE: En este ejemplo se muestra una vista en capas de aplicación multi niveles y basadas en Web que utilizan la plataforma Java EE. La capa de la cima muestra las clases de presentación que son servlets y clases JSF. La segunda capa son los controladores que implementan la secuencia de pasos para completar la funcionalidad de un caso de uso. La capa más baja representa acceso a datos que gestiona las interacciones con la base de datos relacional.



Además existen dos conjuntos adicionales de módulos auxiliares que están representados por capas laterales que en este caso son DTOs (Data Transfer Object) y POJOs (Plain Old Java Object). Siendo los DTO las entidades que transportan datos y los POJO las entidades de datos almacenadas en la base de datos.
Como se muestra a continuación, también se pueden mostrar los paquetes que conforman cada capa:



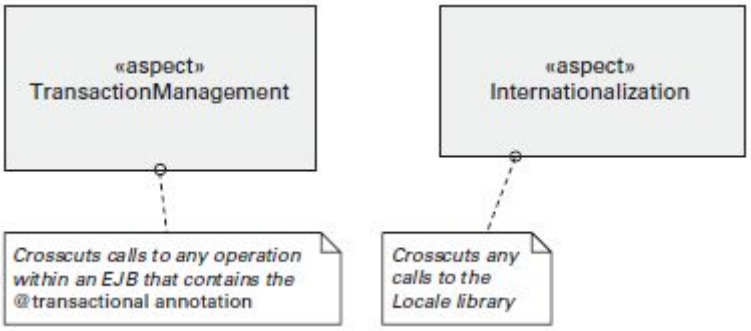
Pag. 87

Vista de Módulos	➔ *Estilo de Aspectos*

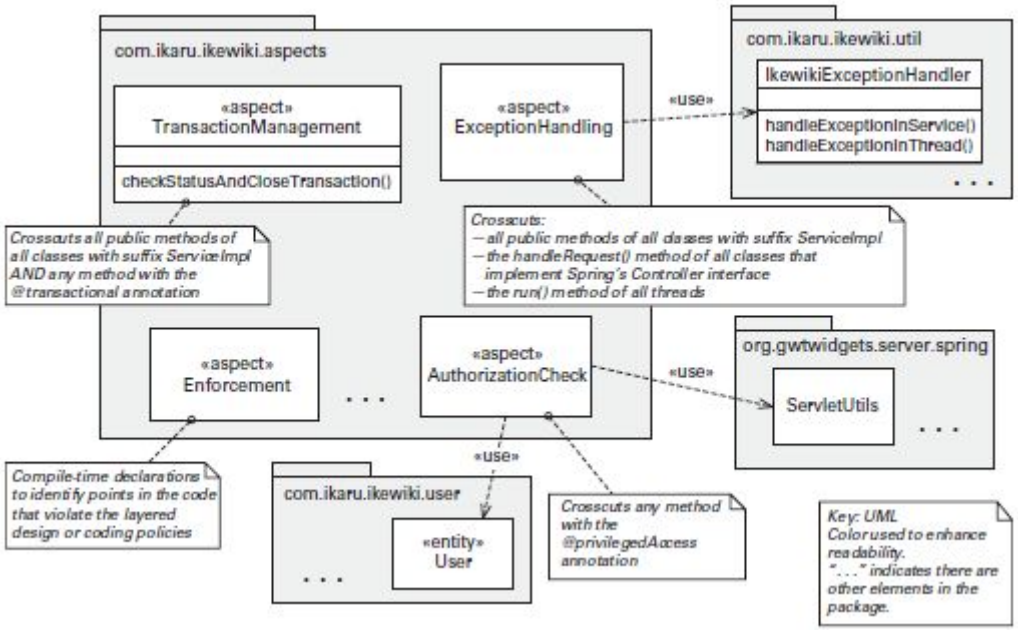
Es un aspecto que se emplea para aislar los módulos responsables de acciones transversales. Los módulos se denominan aspectos, basados en la terminología introducida por la Programación Orientada a Aspectos (AOP).
La vista de aspectos debería contener información que enlace cada módulo de aspectos a otros módulos que requieren cierta funcionalidad transversal. Este estilo es principalmente de utilidad cuando se planifica emplear AOP en la implementación, aunque también se puede emplear para funcionalidad transversal como por ejemplo herencia, inserción de marcos, inyección de dependencias, bibliotecas de utilidades u otros.

El objetivo de diseñar e implementar funcionalidad transversal de manera separada apunta a incrementar la modificabilidad de los módulos que gestionan funcionalidad específica de negocio.
Un aspecto es un módulo especializado que contiene la implementación de cierta funcionalidad transversal. En este estilo los aspectos atraviesan(crosscut) a otros módulos e influyen en la lógica del módulo atravesado.

Notación: En UML se emplea el estereotipo “aspect” para indicar que el módulo es un aspecto. La relación que atraviesa a un módulo va desde el aspecto hacia el módulo objetivo. Los aspectos que atraviesan más de un módulo, no deben poseer la línea hacia todos los objetivos, se puede indicar con una anotación los módulos que el aspecto influye.



A continuación un diagrama que representa la aplicación Java EE IkeWiki implementada con Spring Framework y Google Web Toolkit que emplea aspectos en ciertos casos como por ejemplo la gestión de transacciones, gestión de excepciones, autorizaciones y otros:

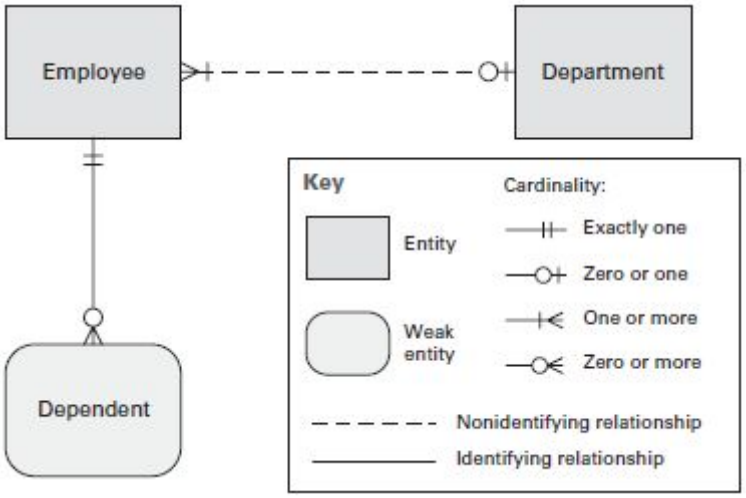


Vista de Módulos	➔ *Estilo de Modelo de Datos*

El modelo de datos describe la estructura de las entidades de datos y sus relaciones. Sus elementos y características son:

- **Entidades de datos:** Es un objeto que contiene la información que necesita ser almacenada o representada por el sistema. Sus propiedades son: nombre, atributos de datos, clave primaria, clave foránea y reglas.
- **Relaciones:** *Uno-a-uno*, *uno-a-muchos* y *muchos-a-muchos*. Generalización/especialización, indica la relación *es-un* entre entidades. Agregación, transforma la relación en una entidad de agregación. Por ejemplo: la relación entre un paciente, un médico y una fecha se puede abstraer en la entidad de agregación "Turno".
- **Uso:** Se emplea para describir la estructura de los datos que gestiona el sistema. Realizar análisis de impacto de cambios en el modelo de datos. Incrementar la calidad de los datos evitando redundancias e inconsistencias. Guiar en la implementación de los módulos que acceden a los datos.

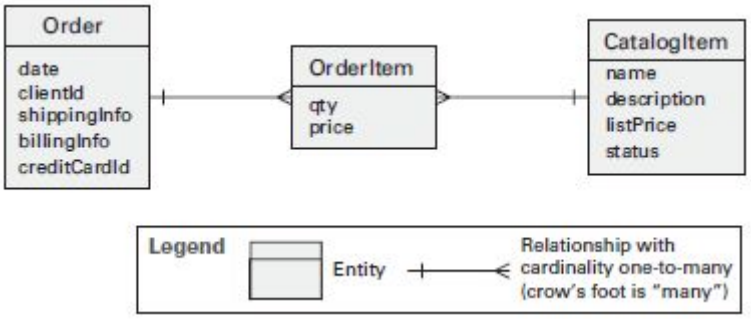
Notación: La notación más empleada actualmente es una combinación de una propuesta por Baker durante los años '80 y Martín-Finkelstein en el año 1981, se denomina DER Crow's Foot y su simbología se puede resumir con siguiente diagrama:



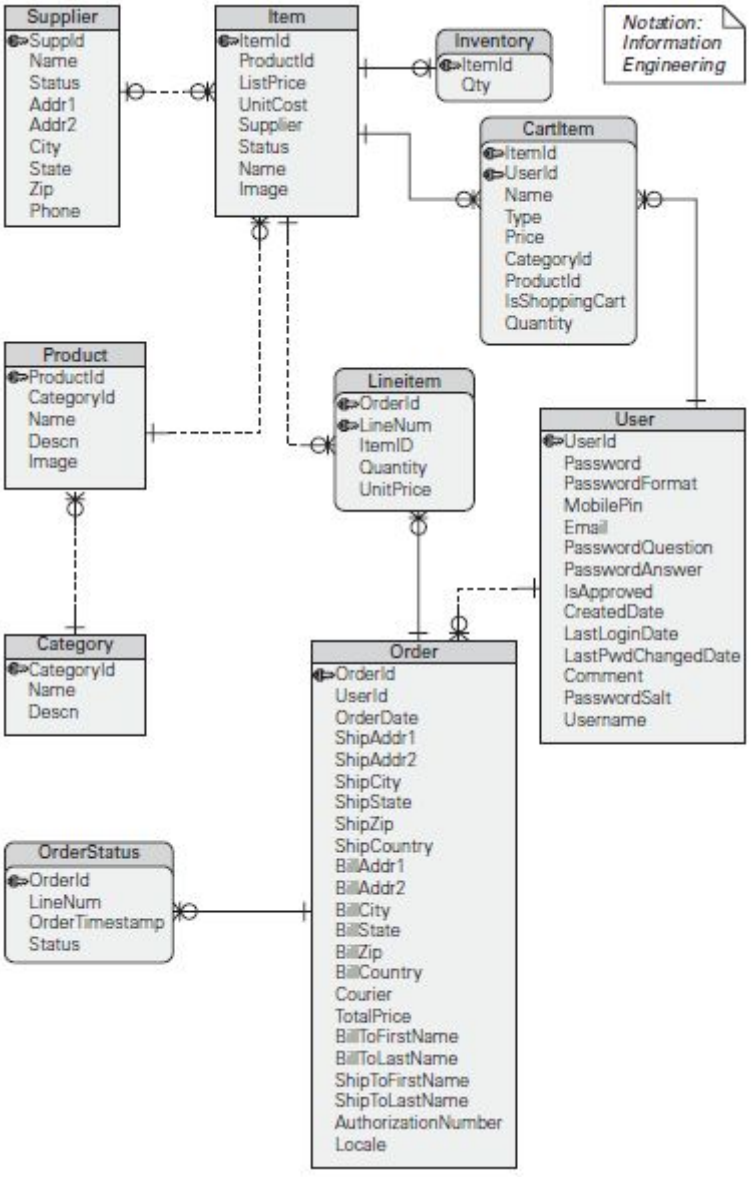
Una entidad es débil (o dependiente) cuando su existencia depende de la existencia de otra. Por ejemplo, Ítem depende de la existencia de Factura.

Ejemplos:

El modelo de datos lógico es una evolución del modelo de datos conceptual que omite detalles sobre la tecnología de gestión de datos que lo soporta. El siguiente diagrama representa un modelo de datos donde un Pedido (Order) posee muchos Items que su vez cada Item corresponde a sólo un ítem del catálogo (CatalogItem):



A continuación un ejemplo que emplea la notación “Information Engineering” que es una variación del DER Crow’s Foot (pata de cuervo):

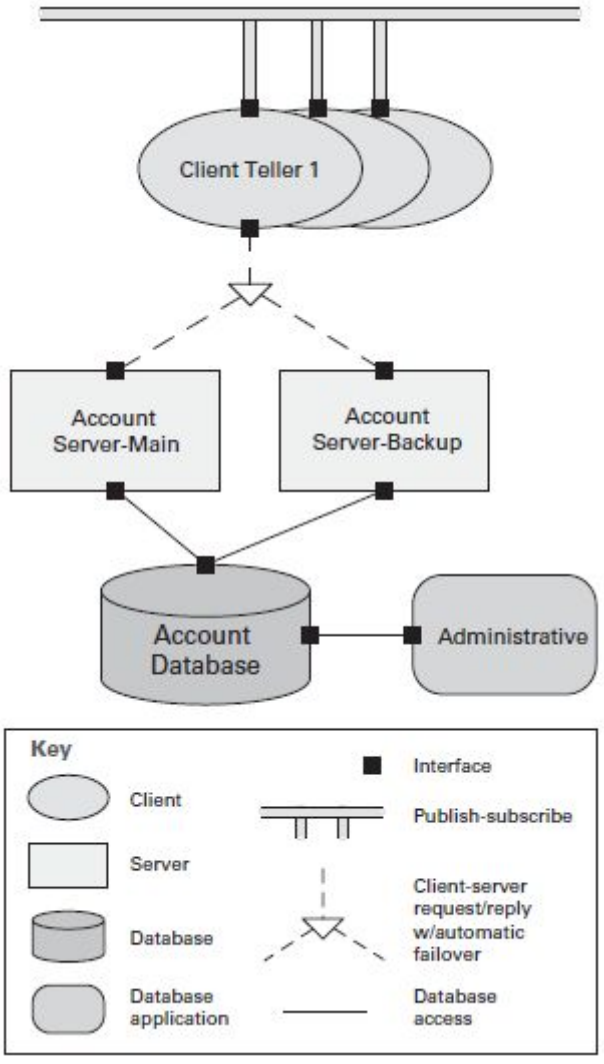


Este modelo de datos corresponde a la aplicación de ejemplo Pet Shop (Microsoft) que es una tienda virtual que gestiona un catálogo de mascotas y toma pedidos (orders) de usuarios registrados.

Pag. 109

Vista de Componentes & Conectores

La vista de Componentes y Conectores muestra elementos que tienen presencia durante la ejecución, tales como procesos, objetos, clientes, servidores y repositorios de datos. Esos elementos se denominan *componentes*. A su vez esta vista incluye elementos que son la senda de interacción, tales como enlaces de comunicación, protocolos, flujos de información y acceso a almacenamiento compartido. Esas interacciones son representadas como *conectores* en la vista C&C.
La siguiente figura ilustra con una vista C&C la arquitectura general de un sistema.



El sistema está integrado por un repositorio compartido (Account Database) que es accedido por dos servidores y un componente administrativo. Un conjunto de clientes

pueden interactuar con el repositorio mediante un estilo *cliente-servidor*. Estos 3 tipos de conectores representan diferentes formas de interacción entre partes:

- Conectores cliente-servidor: Permiten que clientes concurrentes recuperen datos sincrónicamente a través de solicitudes de servicio.
- Conector de acceso a base de datos: Soporta acceso autorizado y transaccional para leer, escribir y monitorear la base de datos.
- Conector publicante-suscriptor: Soporta anuncio de eventos asincrónicos y notificaciones.

La combinación de los diagramas C&C con su documentación de soporte provee un medio esencial para comunicar el diseño arquitectónico de un sistema, explicando su funcionamiento durante la ejecución y justificando decisiones de diseño en términos de su impacto en atributos de calidad relevantes.

Sus principales características son:

Elementos

- Componentes: Los componentes son las unidades de procesamiento y persistencia de datos. Un componente posee un conjunto de puertos a través de los cuales interactúa con otros componentes.
- Conectores: Son las vías de interacción entre componentes. Los conectores poseen un conjunto de roles que indican cómo los componentes deben utilizarlos durantes las interacciones.

Relaciones

- Adjuntos: Los puertos de los componentes están asociados a roles de conectores y juntos conforman un grafo de componentes y conectores.
- Delegación de interfaz: En ciertas situaciones los puertos de los componentes están asociados con uno o más puertos dentro de una subarquitectura interna.

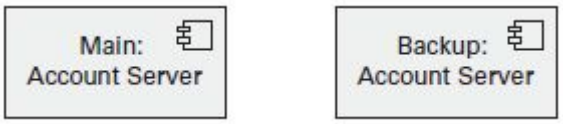
Restricciones

- Los componentes sólo pueden ser enlazados a través de conectores.
- Los conectores sólo pueden ser enlazados a componentes, no a otros conectores.
- La delegación de interfaz sólo puede ser definida entre dos puertos compatibles.
- Los conectores no pueden aparecer en forma aislada, un conector debe estar adjunto a un componente.

Uso

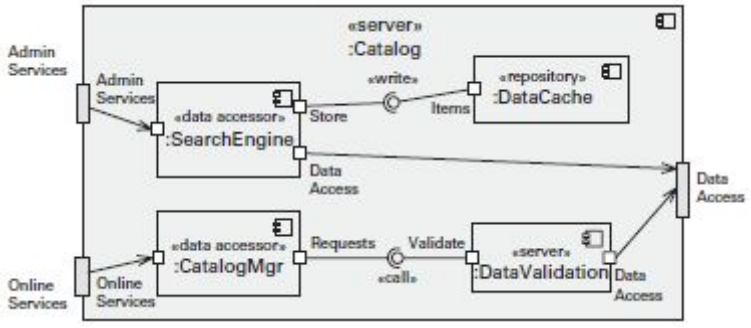
- Se utilizan para mostrar cómo funciona el sistema.
- Guían el desarrollo especificando la estructura y comportamiento de elementos en tiempo de ejecución.
- Son una herramienta de soporte para estudiar el comportamiento del sistema en tiempo de ejecución de atributos de calidad tales como desempeño, confiabilidad y disponibilidad.

Notación: Los componentes UML concuerdan semánticamente con los componentes C&C porque comunican intuitivamente aspectos importantes como interfaces, propiedades y comportamiento. Los componentes UML también distinguen entre tipos de componentes (Account Server) e instancias (Main y Backup):

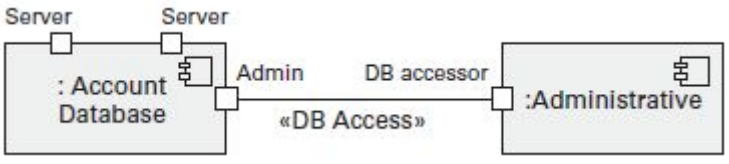


Los componentes pueden crear y administrar dinámicamente diferentes tipos y cantidades de puertos, por lo tanto también deben especificar la multiplicidad para cada instancia:

UML proporciona una notación de lollipop/socket para mostrar las interfaces necesarias conectadas a los puertos. Cada puerto puede tener un número arbitrario de interfaces proporcionadas (lollipop o círculo) y requeridas (socket o zócalo):

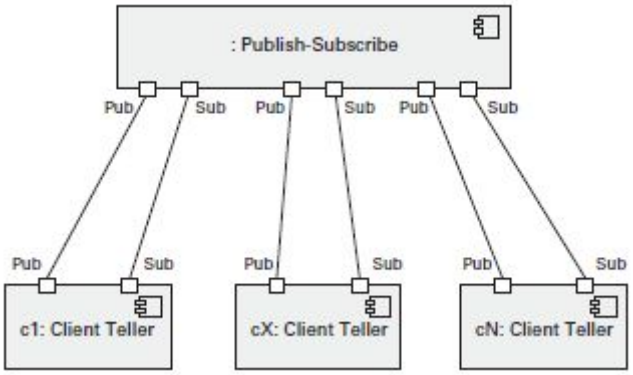


Habitualmente en las vistas C & C se omiten los zócalos y círculos y se emplean sólo las definiciones de tipo de componente y conectores:



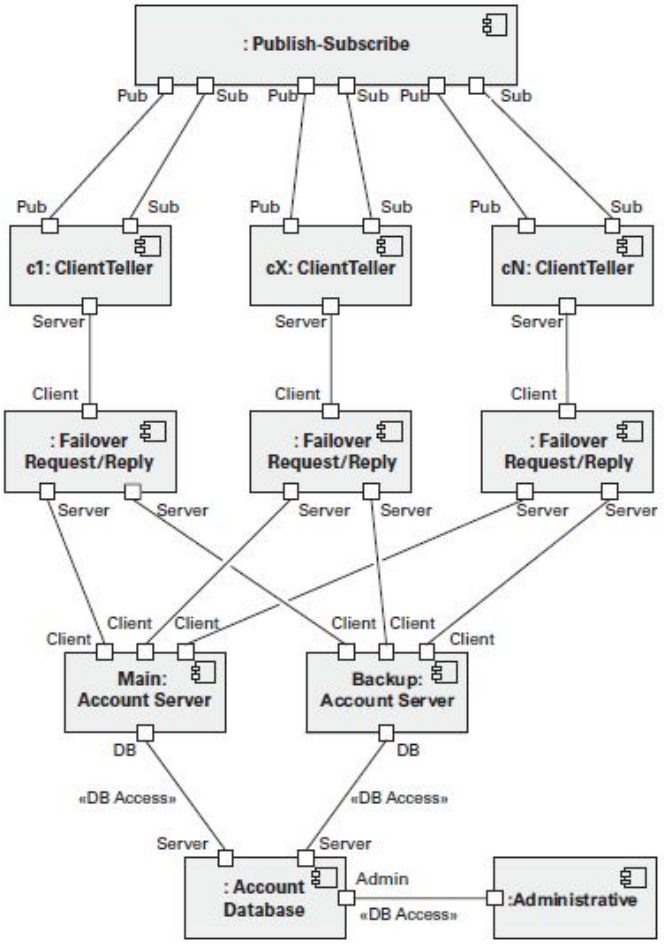
*Los **componentes** son los principales elementos computacionales y almacenes de datos que se ejecutan en un sistema. Un **puerto** es una interfaz de un componente. Un puerto define un punto de interacción de un componente con su entorno.*

En el siguiente diagrama se representa la combinación de vistas Publish-Subscribe y Cliente-Servidor a través del diagrama de Componentes y Conectores:



Dado que las vistas C & C captan los aspectos de tiempo de ejecución de un sistema, es un modelo que describe cómo los datos y el control fluyen a través de los sistemas.

El siguiente diagrama que un número variable de componentes ClientTeller pueden estar conectados a uno o ambos componentes Account Server:



Pag. 123

Categorías de Estilos C & C

Una forma de establecer orden conceptual en el espacio de C&C estilos es a través de 4 categorías principales:

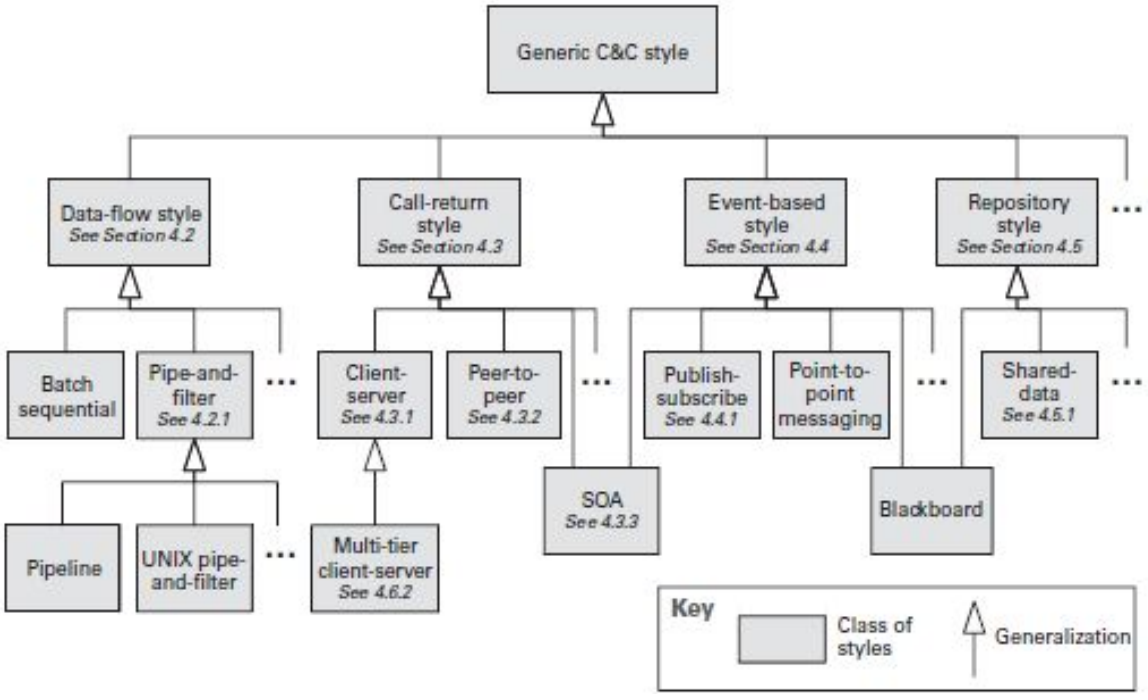
Estilo de Flujo de Datos: En este estilo el procesamiento de datos se lleva a cabo durante el flujo de datos en el sistema.

Estilo Call-Return: Donde los componentes interactúan a través de la invocación sincrónica de sus capacidades de comunicación.

Estilo basado en Eventos: Los componentes interactúan a través de la invocación asincrónica de eventos o mensajes.

Estilo de Repositorio: En este estilo los componentes interactúan a través de grandes colecciones de datos persistentes y compartidos.

La siguiente figura muestra las categorías y sus especializaciones:

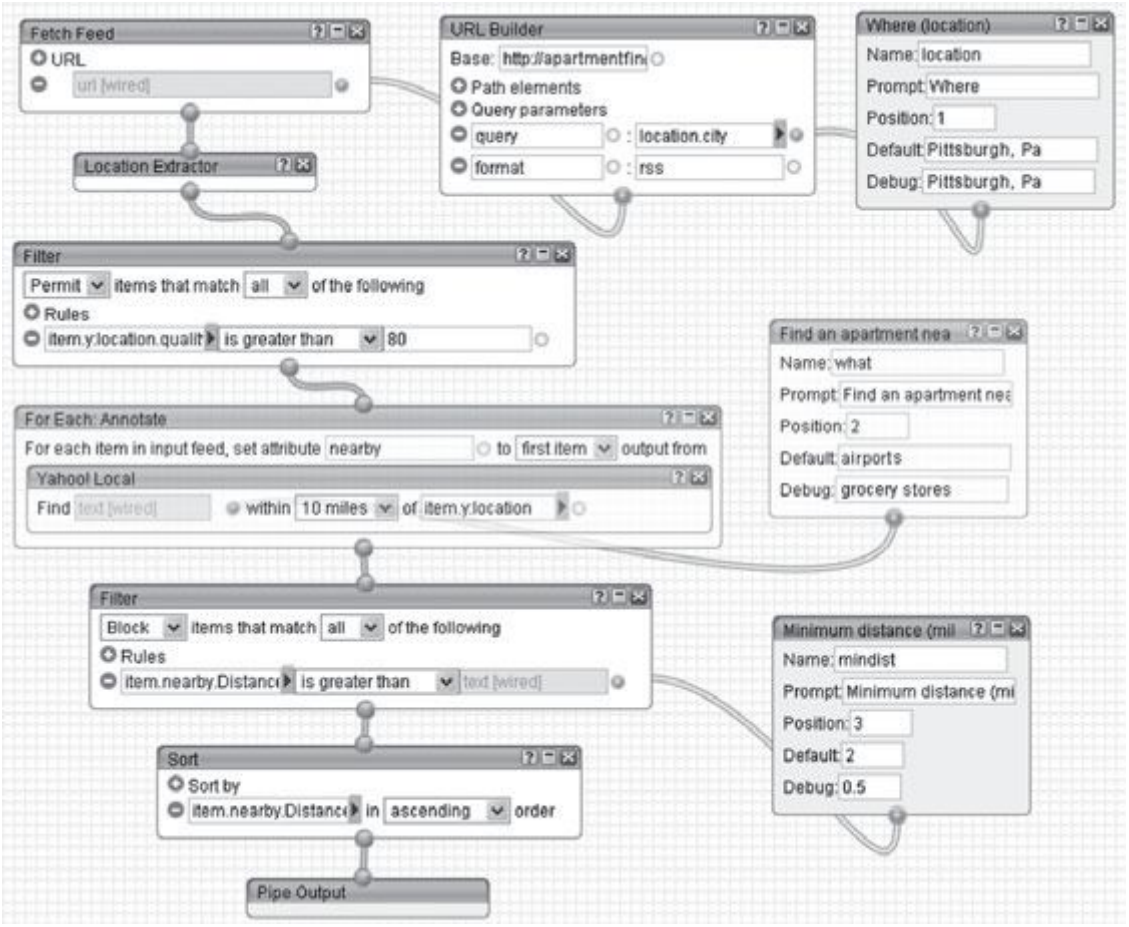


Vista de C & C	➔ *Estilo Tubería y Filtro*

Flujo de Datos: Este estilo de patrón de interacción se caracteriza por la sucesiva transformación de flujos de datos. Sus principales características son:

Uso

Típicamente se emplean en sistemas de transformación de datos, donde el procesamiento puede desglosarse en un conjunto de pasos independientes y responsables de la transformación incremental de los datos. Esto favorece la reutilización, paralelización del procesamiento y simplificación del sistema en general. Dado que las tuberías almacenan datos durante la comunicación, los filtros pueden actuar asincrónicamente y de forma simultánea. Un ejemplo es la aplicación Yahoo! Pipes para buscar departamentos para alquilar cerca de un lugar determinado:



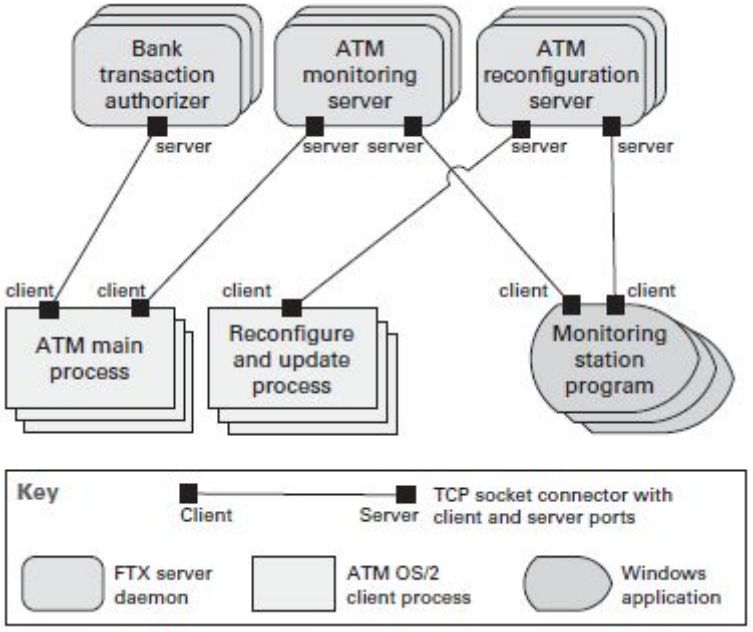
Vista de C & C	➔ *Estilo Call-Return*

Call-Return: En este estilo los componentes proporcionan un conjunto de servicios que pueden ser invocados por otros componentes. Un componente que invoca un servicio pausa (o se bloquea) hasta que el servicio haya finalizado. Por lo tanto, este estilo es análogo a Procedure Call de los lenguajes de programación. Los conectores son responsables de transportar la solicitud de servicio y retornar los resultados. Como se verá a continuación, algunos ejemplos de este estilo son: Cliente-Servidor, Peer-to-Peer y REST.

Pag. 161

Vista de C & C	➔ *Estilo Cliente-Servidor*

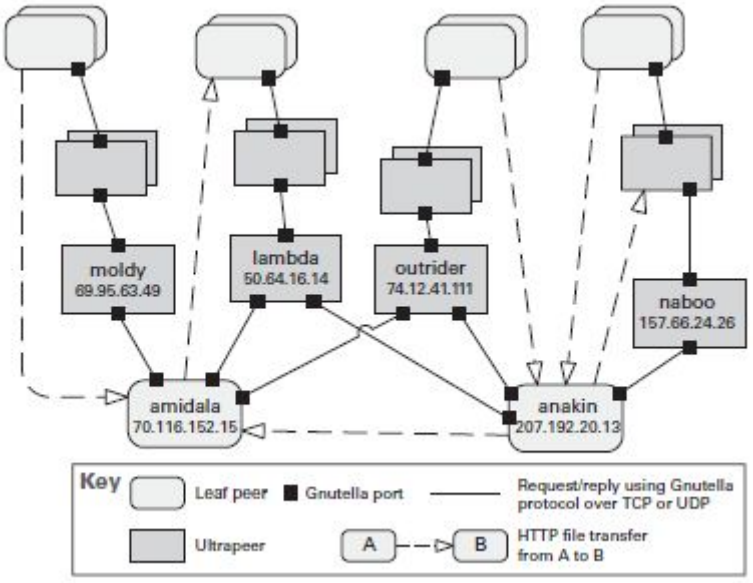
Dentro del estilo Call-Return está el estilo Cliente-Servidor donde los componentes interactúan con otros componentes solicitando servicios entre sí. Un ejemplo de arquitectura Cliente-Servidor es la red de cajeros automáticos (ATM):



Pag. 162

Vista de C & C	➔ *Estilo Peer-to-Peer*

En el estilo peer-to-peer, los componentes interactúan directamente como pares mediante el intercambio de servicios. La comunicación es punto a punto y mediante solicitudes request-response pero sin la asimetría del estilo cliente-servidor. El siguiente diagrama muestra una parte de una vista de la red Gnutella utilizando notación informal C&C:



Pag. 166

Vista de C & C	➔ *Estilo SOA*

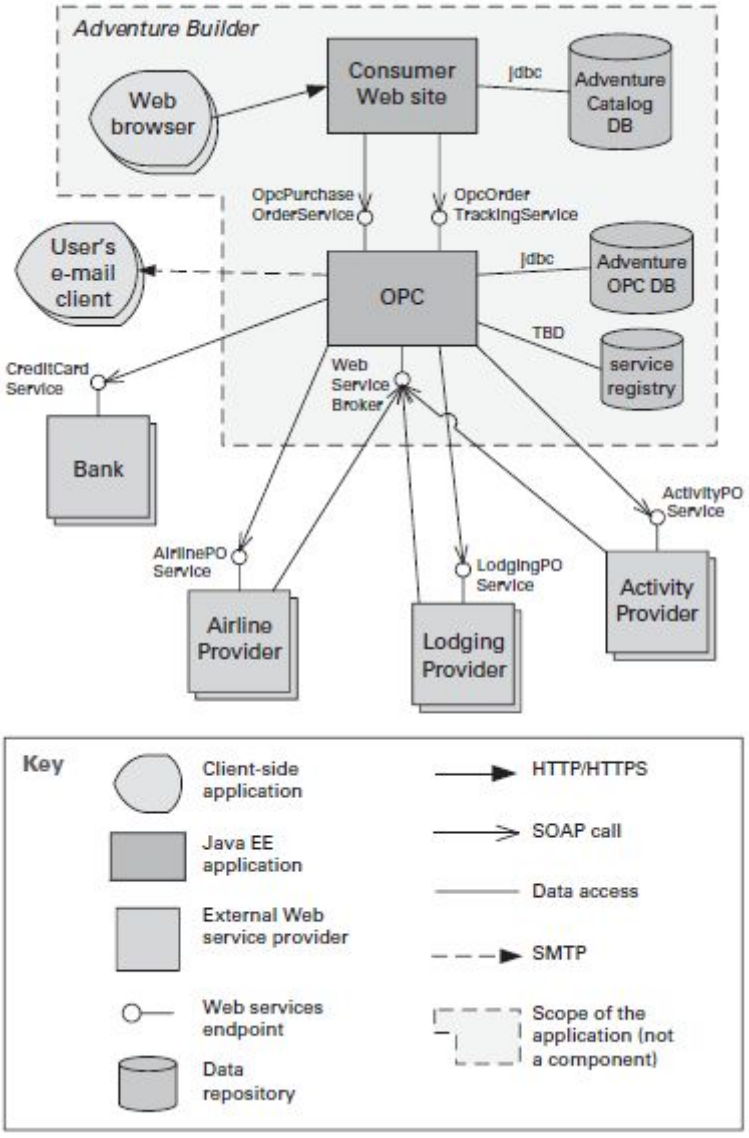
Las arquitecturas orientadas a servicios son una colección de componentes que proporcionan y/o consumen servicios. Habitualmente esta arquitectura también puede estar integrada por:

- **ESB:** La invocación de servicios puede estar mediada por un Enterprise Service Bus (ESB) que enruta los mensajes entre servicios consumidores y proveedores. También pueden realizar conversiones de protocolos.
- **Registro:** Para facilitar la ubicación de estos servicios, existen registros de servicios donde los servicios proveedores se registran y los consumidores los ubican.
- **Orquestación:** Un servidor de orquestación o motor de orquestación es un componente especial que ejecuta scripts ante la ocurrencia de un evento específico.

Los tipos básicos de conectores pueden ser:

- **Call-return:** Los dos más comunes son SOAP y REST.
- **Mensajería Asincrónica:** Los componentes intercambian mensajes asíncrona. Estos pueden ser punto-a-punto o publish-subscribe. La comunicación por mensajería ofrece excelentes capacidades confiabilidad y escalabilidad.

El siguiente diagrama un ejemplo de SOA donde un sistema interactúa a través de servicios web SOAP y otros proveedores externos:



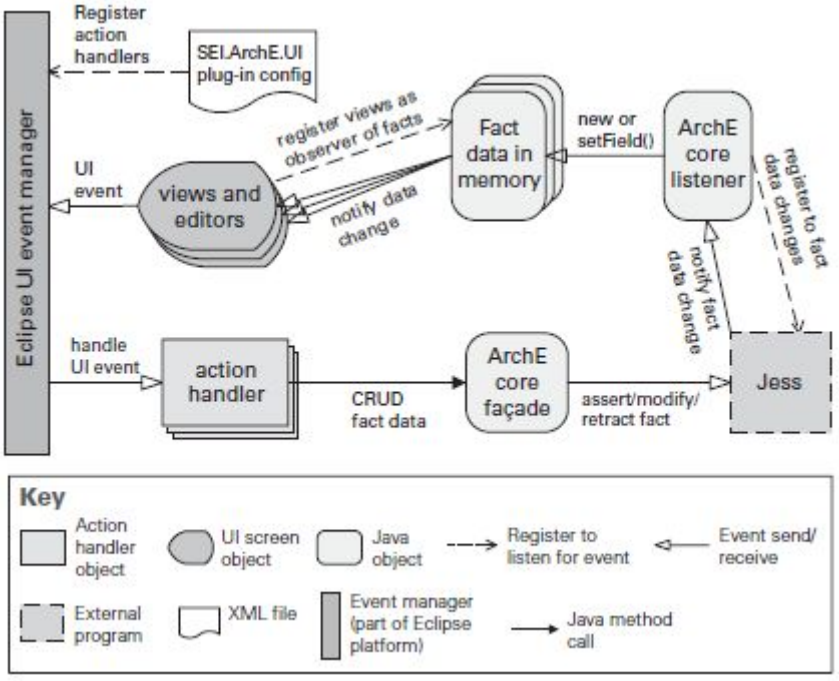
Pag. 169

Vista de C & C	➔ *Estilo Publish-Subscribe*

Eventos: Estos estilos permiten que los componentes se comuniquen a través de mensajes asincrónicos. Suelen organizarse en una federación débilmente acoplada de componentes que disparan comportamiento en otros componentes a través de eventos. En este estilo los componentes interactúan a través de la publicación de eventos. Los componentes se pueden suscribir a un conjunto de eventos. Es responsabilidad de la infraestructura publish-subscribe asegurar que cada evento que es publicado sea entregado a todos los suscriptos.

El siguiente diagrama es un ejemplo del estilo publish-subscribe, donde:

1. Eclipse UI event manager es un bus de eventos de usuario (por ejemplo clicks en botones).
2. Cuando un usuario modifica elementos de datos (facts), genera un evento que es enviado a Jess (el motor de reglas), de forma tal que pueda activar otras reglas.

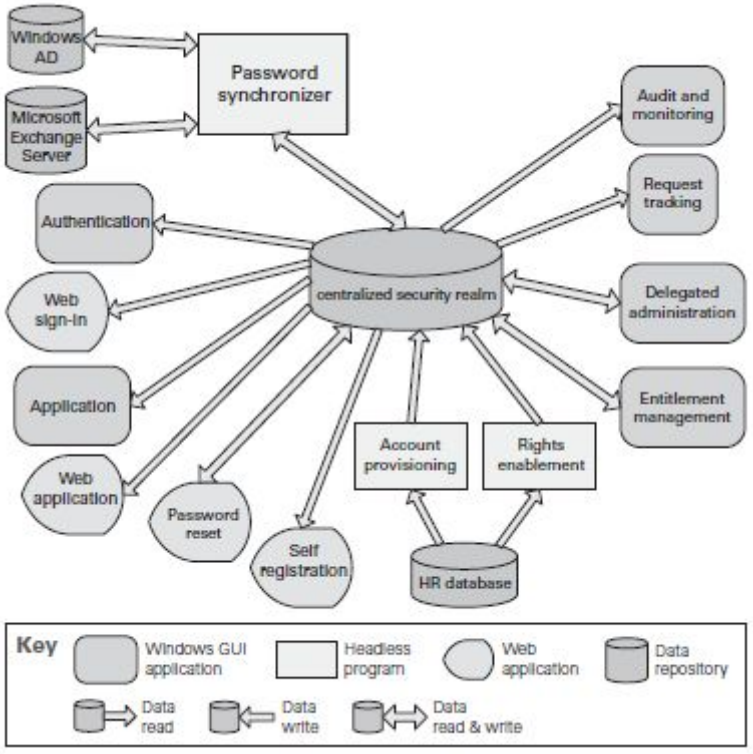


Pag. 174

Vista de C & C	➔ *Estilo de Datos Compartidos*

Repositorio: En este estilo el patrón de interacción es el intercambio de datos persistentes. Los sistemas de gestión de bases de datos (DBMS) y los sistemas basados en conocimiento son ejemplos de este estilo.

En el siguiente diagrama se muestra un ejemplo de este estilo, donde uno o más almacenes de datos, comparten datos a la que otros componentes pueden leer o escribir:

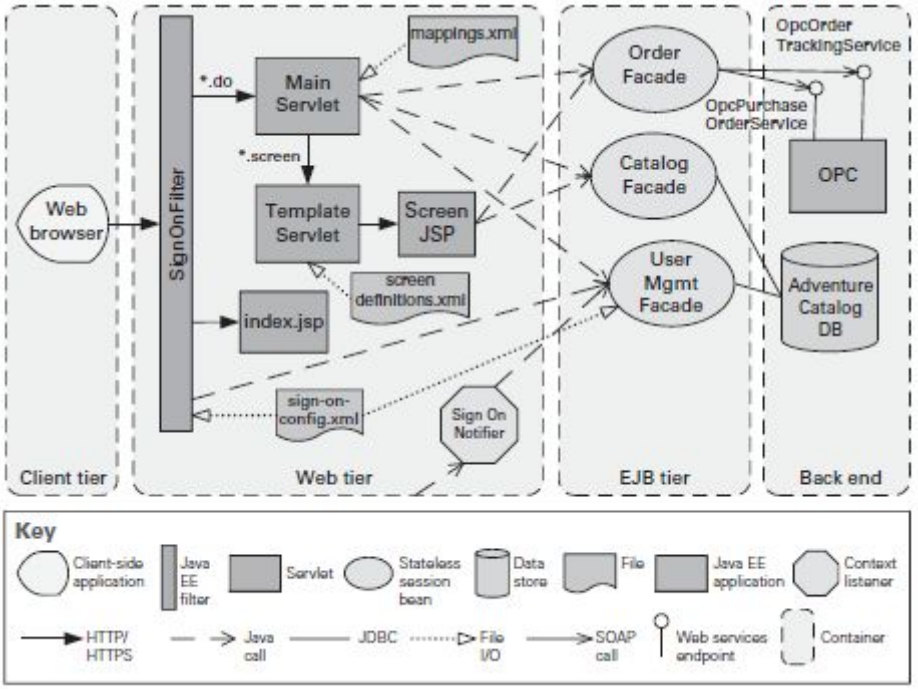


Pag. 178

Vista de C & C	➔ *Estilo de Niveles*

Las estructuras de ejecución de muchos sistemas pueden estar organizadas como un conjunto de niveles lógicos de componentes de acuerdo a un criterio específico como compartir el mismo ambiente de ejecución o tener un propósito en común. En la práctica es habitual verlo para el estilo Cliente-Servidor, no obstante, puede ser utilizado para cualquier estilo C&C. Puede resumirse como un mecanismo utilizado para particionar un sistema.

El siguiente diagrama emplea una notación informal para describir una arquitectura de múltiples niveles:



Pag. 183

Vista de Asignación

Estas vistas presentan un mapeo entre los elementos software y los elementos no-software de su ambiente.

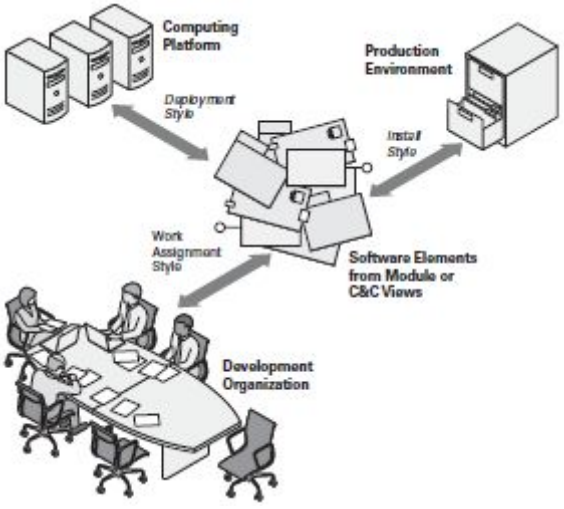
Se identifican 3 estilos de asignación:

Estilo de despliegue: Describe el mapeo entre los componentes de software y conectores y el hardware de la plataforma de computación donde éste se ejecuta.

Estilo de instalación: Describe el mapeo entre los componentes de software y las estructuras en el sistema de archivos del ambiente de producción.

Estilo de asignación de trabajo: Describe el mapeo entre los módulos de software y las personas, equipos o unidades de trabajo asignadas al desarrollo de esos módulos.

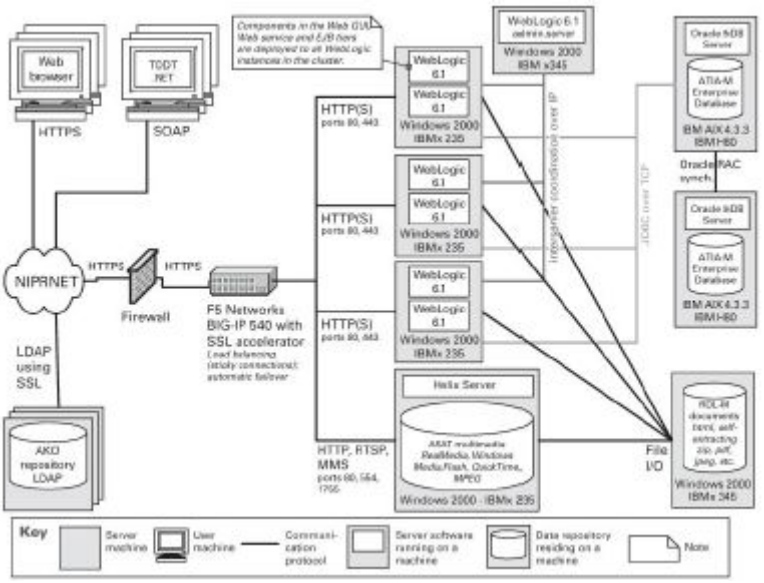
La siguiente figura ejemplifica visualmente cada uno de los estilos (despliegue, instalación y asignación de trabajo):



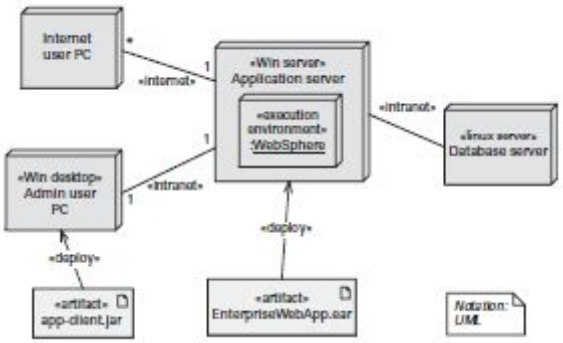
Pag. 189

Vista de Asignación	➔ *Estilo de Despliegue*

A continuación 2 ejemplos del estilo de despliegue:



(Notación informal)

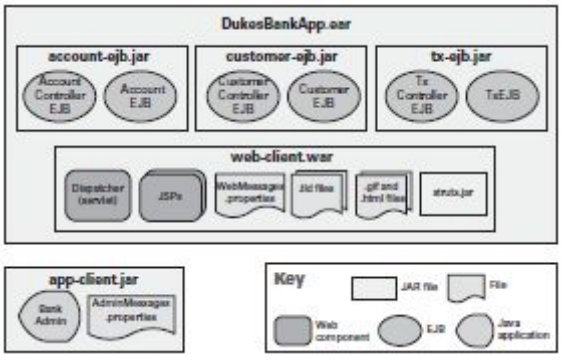


(UML)

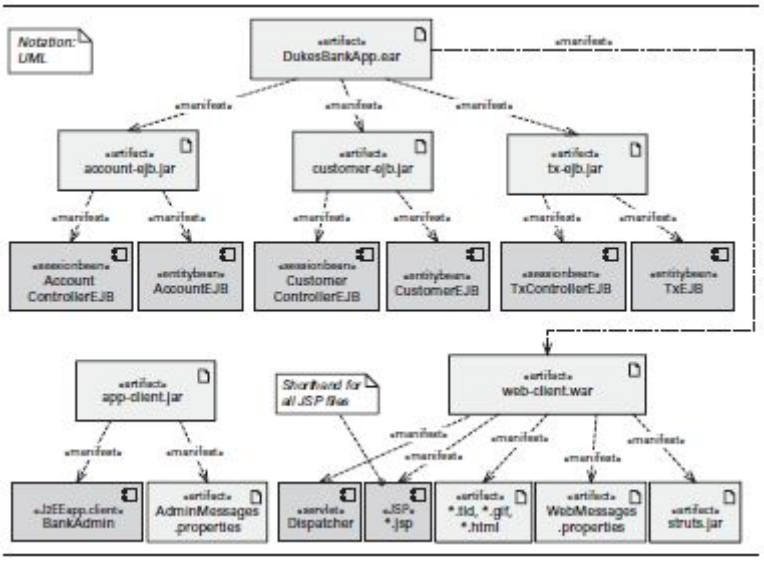
Vista de Asignación	➔ *Estilo de Instalación*

Este estilo es apropiado para entender la organización de los archivos y carpetas del software instalado. Es de utilidad para:

- Crear procedimientos de construcción y despliegue.
- Navegar por un gran número de archivos y carpetas que constituyen el sistema instalado, para localizar archivos específicos que requieren atención (como un archivo de registro o un archivo de configuración).
- Seleccionar y configurar los archivos para empaquetar una versión línea de productos de software.
- Actualizar y configurar archivos de varias versiones instaladas de el mismo sistema
- Identificar el propósito o el contenido de un objeto perdido o dañado que esté causando un problema en producción.
- Diseñar e implementar "actualizaciones automáticas".



(notación informal)



(UML)

Vista de Asignación	➔ *Estilo de Asignación*

El estilo suele usarse para vincular las actividades a los recursos para garantizar que los módulos se asignen un individuo o un equipo.

El WBS es una herramienta que define al proyecto como elementos discretos de trabajo, de modo tal que ayuda a organizar y definir todo el alcance del trabajo requerido por el proyecto:

WBS (Work Breakdown Structure)

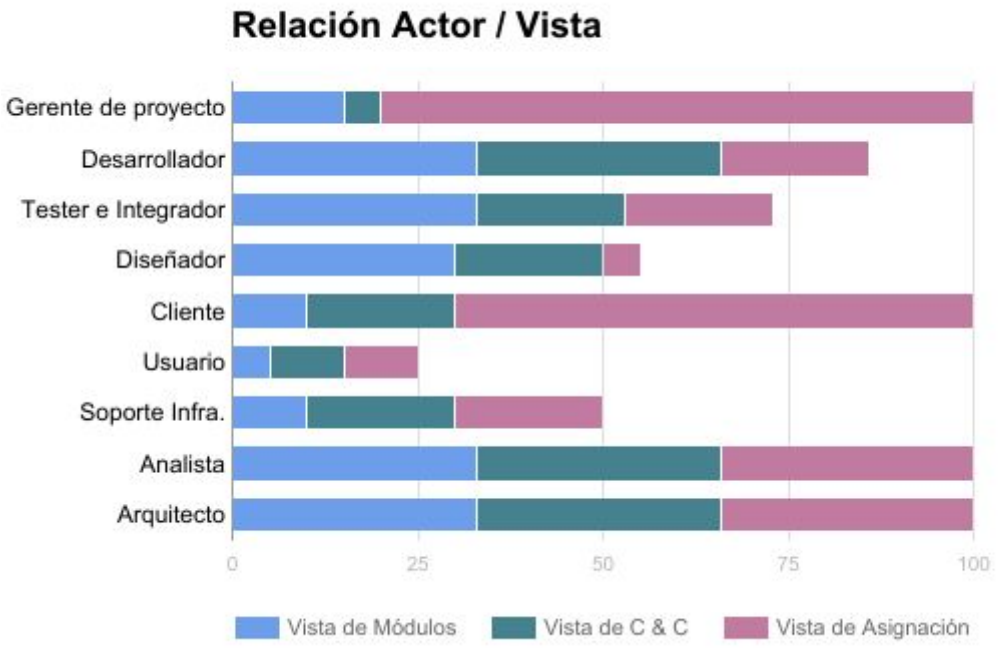
No existen notaciones especiales para mostrar las vistas de asignación de trabajo. Entre las notaciones informales, una tabla que muestra los elementos de software y los equipos responsables a menudo es suficiente. La siguiente figura muestra una vista de asignación de trabajo para un sistema de la NASA llamado ECS. Los módulos de nivel más alto se llaman segmentos y se descomponen en unidades llamadas subsistemas.

ECS Element (Module)		
Segment	**Subsystem**	**Organizational Unit**
Science Data Processing Segment (SDPS)	Client	Science team
	Interoperability	Prime contractor team 1
	Ingest	Prime contractor team 2
	Data Management	Data team
	Data Processing	Data team
	Data Server	Data team
	Planning	Orbital vehicle team
Flight Operations Segment (FOS)	Planning and Scheduling	Orbital vehicle team
	Data Management	Database team
	User Interface	User interface team
. . .	. . .	. . .

(asignación de trabajo para sistema ECS)

Seleccionando las Vistas

Es clave saber determinar qué vista es apropiada para cada actor del sistema y en qué momento liberarla. El conjunto de actores dependerá de la organización y el proyecto. A continuación una lista de actores y una relación "estándar" con sus requerimientos de documentación.



Gerente de proyecto: Se enfoca principalmente en el cronograma, asignación de recursos y planes de contingencia. No se interesa en cuestiones específicas del diseño o las características detalladas de las interfaces, no obstante si precisa conocer el propósito general del sistema, sus restricciones e interacción con otros sistemas.

Desarrollador: La documentación proporciona una referencia sobre avance de su tarea, modo de desarrollar esa tarea, responsabilidad sobre algún elemento o componente del sistema. A su vez la documentación puede proporcionar información acerca de un producto COTS y saber cómo funciona o debe adaptarlo.

Tester e Integrador: Son actores para quienes la arquitectura especifica el comportamiento "caja negra" correcto de las piezas que la integran. Un tester unitario deseará conocer la misma información que el desarrollador de ese elemento. Un tester de "caja negra" precisará conocer la documentación acerca de la interfaz del elemento.

Diseñador: Los diseñadores de otros sistemas con los que el sistema en desarrollo deberá interoperar, también son actores clave. Para ellos la documentación de arquitectura define el conjunto de operaciones provistas y requeridas, como así también protocolos y su modo de operación.

Cliente: Son los actores que financian el desarrollo del sistema. Se interesan en costos y avances, por lo tanto desearán contar con documentación que convincente que verifique que la arquitectura es apropiada para cumplir con los requerimientos funcionales y de calidad esperados. También les interesa saber sobre el ambiente donde el sistema se ejecutará y si
interoperará con otros sistemas en ese ambiente.

Usuario: No precisan conocer la arquitectura en profundidad, pero puede ser favorable que incorporen nociones para saber qué hace el sistema y cómo pueden utilizarlo.

Soporte de Infraestructura: Preparan y mantienen la infraestructura que soporta el desarrollo, construcción y puesta en producción del sistema. Se debe proveer documentación acerca de las "partes" disponibles en la infraestructura.

Analista: Están interesados en saber si el diseño cumple con los objetivos de calidad del sistema. La arquitectura sirve como referencia de evaluación y deben contener la suficiente información para evaluar atributos de calidad como seguridad, desempeño, usabilidad, disponibilidad y modificabilidad.

Arquitecto: Los futuros arquitectos son los más ávidos lectores de la documentación de arquitectura del sistema.

Pag. 319

Documentando Decisiones Arquitectónicas

El proceso de desarrollar la arquitectura de un sistema complejo, implica tomar cientos de grandes o pequeñas decisiones. El resultado de esas decisiones son reflejadas por vistas que documentan la arquitectura. La mayoría de esas decisiones se toman en ambientes complejos donde en muchos casos implica negociar cronograma o costos. Si esas decisiones no son documentadas correctamente se perderá la información del contexto y que las justificaron.

Documentar decisiones arquitectónicas a medida que se obtienen resultados, demuestran cuán alineada con el negocio y los objetivos técnicos del sistema está.

Documentar no es una actividad que se debe realizar una vez que el proyecto ha finalizado. Documentar ayuda a diseñar la arquitectura.

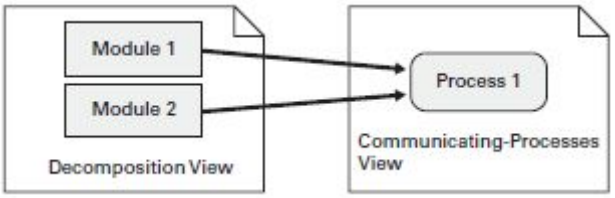
Pag. 239

Combinando Vistas Arquitectónicas

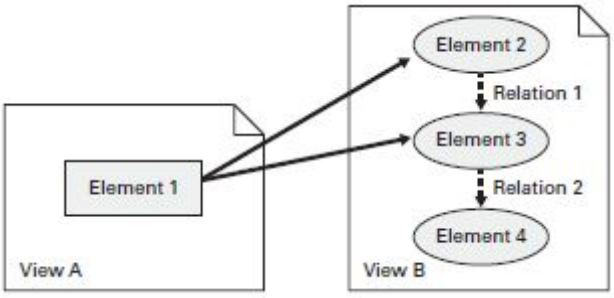
Como todas las vistas de una arquitectura poseen un propósito común, muchas poseen fuertes asociaciones entre sí. Gestionar cómo se asocian es un parte importante del trabajo de un arquitecto, al igual que definir y documentar esas asociaciones.

Tipos de asociaciones entre vistas:

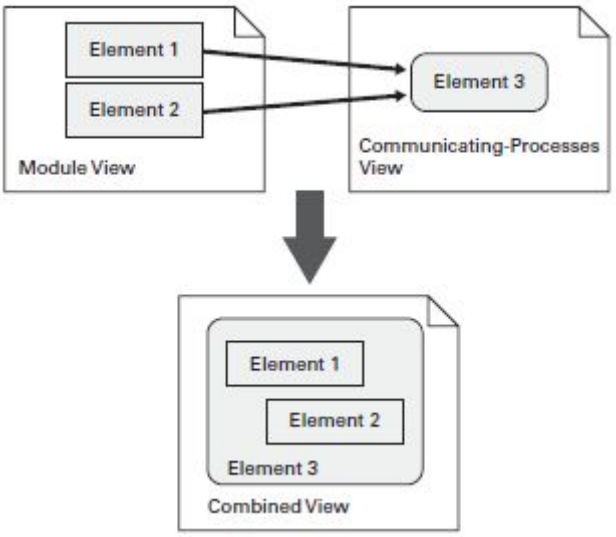
En asociaciones **muchos-a-uno**, múltiples elementos de una vista son asociados en un solo elemento de otra vista. Esta asociación debe mostrar claramente el mapeo entre módulos y componentes. La siguiente figura muestra cómo múltiples módulos deben mapear a un solo proceso:



En asociaciones **uno-a-muchos**, un solo elemento de una vista es asociado a múltiples elementos en otra vista. Por ejemplo, un módulo de carrito de compras mapea a múltiples componentes en una vista de niveles de una aplicación de tienda web:



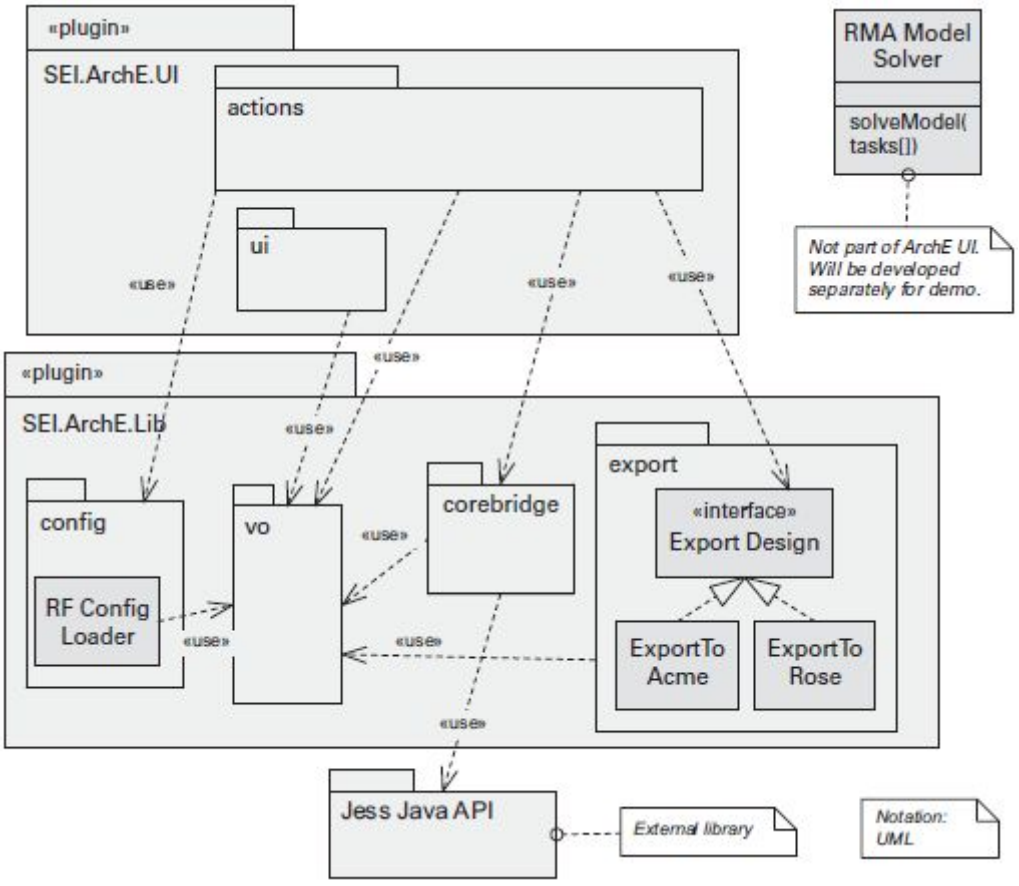
Por último, una asociación **muchos-a-muchos** asocia un conjunto de elementos en una vista a un conjunto de elementos en otra. Este tipo de asociación refleja la complejidad inherente al relacionar dos vistas entre sí, cada una de las cuales fue diseñada para mostrar su aspectos importantes que en muchos aspectos pueden ser ortogonales aquellos en la otra vista:



Existen 2 formas de crear una vista combinada:

- Creando una **superposición** que combine la información de dos vistas separadas.

Por ejemplo la siguiente figura es una vista combinada de descomposición de módulos y generalización:



Creando un estilo **híbrido** que combine dos estilos existentes y creando una guía de estilo que indique cuáles estilos fueron combinados y describir cualquier elemento o relación nuevo o híbrido, especificando sus tipos, propiedades y restricciones.

Estas vistas habitualmente se combinan de forma “natural”:

Vistas C&C: Dado que las vistas C&C muestran las relaciones en tiempo de ejecución de varios tipos de componentes y conectores, tienden a combinar bien.

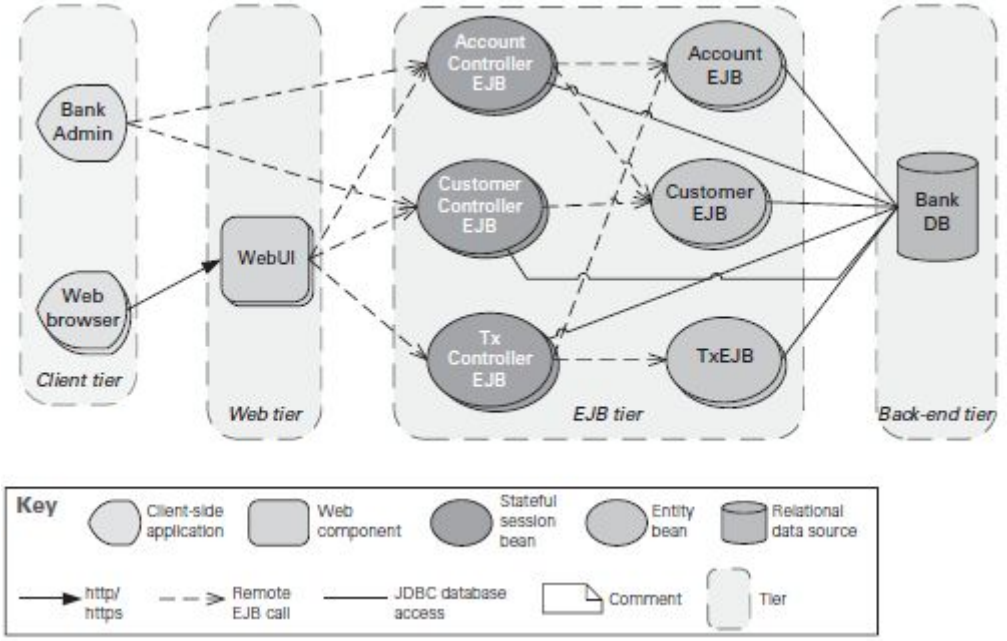
Vistas de despliegue con vistas orientadas a servicios o procesos de comunicación: Ambas vistas muestran procesos de comunicación y muestran componentes desplegados en procesadores. Existe una fuerte asociación entre los elementos de estas vistas.

Vista de despliegue con vista de instalación: Su combinación muestra los archivos de instalación y el hardware donde deberán ser desplegados.

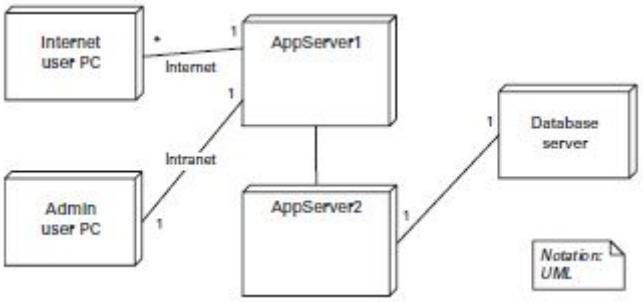
Vista de descomposición y asignación de trabajo, implementación o capas: La descomposición de módulos muestran las unidades de trabajo, desarrollo y uso.

Generalización y aspectos: Ambas vistas tratan clases, objetos y sus relaciones, por lo tanto, estas vistas poseen una fuerte asociación entre sí.

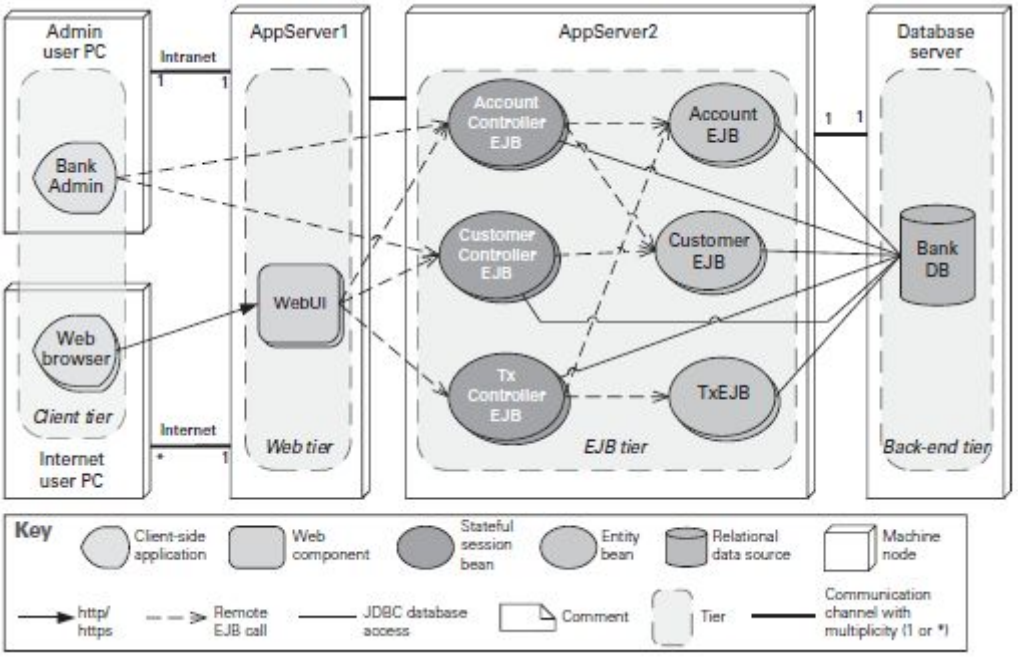
El siguiente ejemplo muestra una combinación de una vista multi-capa cliente-servidor combinadas con una vista de despliegue:



Cliente-servidor multicapa



Vista de despliegue



Vista combinada

Pag. 250

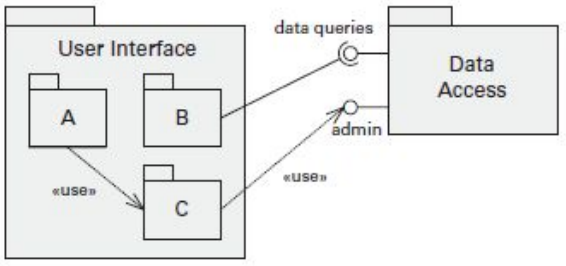
Referencias

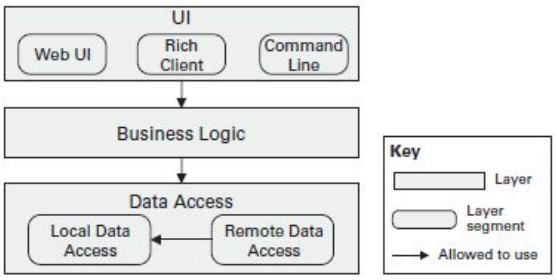
Clemens, P., Documenting Software Architectures Views and Beyond.
Addison-Wesley. 2011.

Anexo A

Guía de Estilos

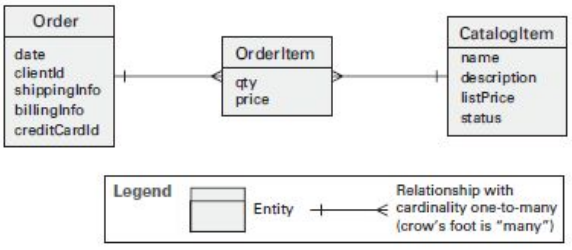
Módulos:
Descomposición

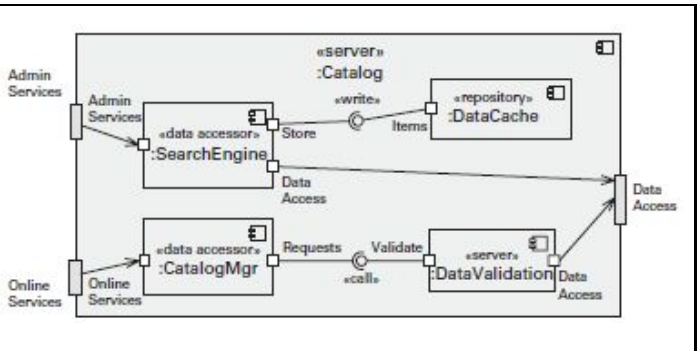




Módulos:
Capas

Módulos:
Modelo de Datos



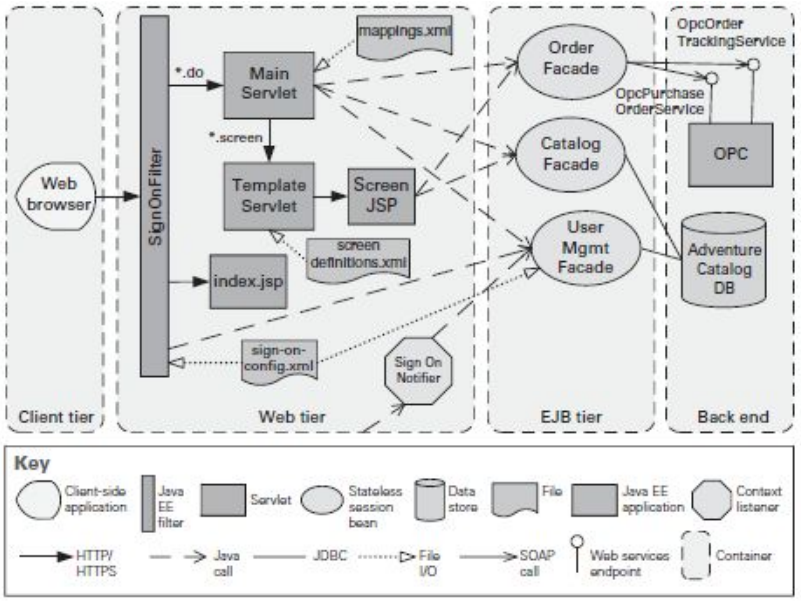


Componentes &
Conectores

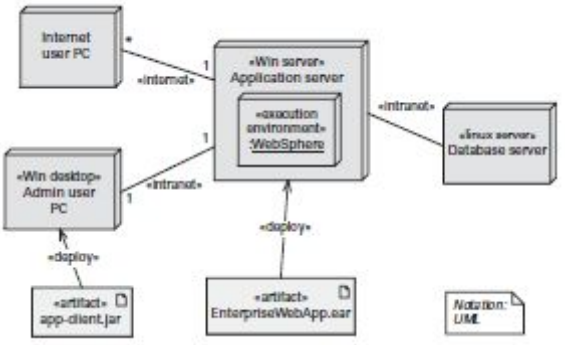
Anexo A

Guía de Estilos

Componentes & Conectores: Niveles



Asignación: Despliegue



Combinada: Niveles - Despliegue