UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

DOCUMENTACIÓN DE ARQUITECTURAS DE SISTEMAS EN UN BANCO

TESIS PARA OPTAR AL GRADO DE
MAGISTER EN TECNOLOGÍAS DE LA INFORMACIÓN

FRANCISCO JAVIER AHUMADA AHUMADA

PROFESORA GUÍA:
CECILIA BASTARRICA PIÑEYRO

MIEMBROS DE LA COMISIÓN:
SERGIO OCHOA DELORENZI
JOSÉ BENGURIA DONOSO
MARCOS SEPÚLVEDA FERNÁNDEZ

SANTIAGO DE CHILE

ABRIL DE 2013

RESUMEN

Un banco posee cientos de sistemas para apoyar a la mayoría de las funciones del negocio y procesar millones de solicitudes de sus clientes internos y externos. Una falla en estos sistemas puede causar, desde un retraso en la entrega de informes de gestión, hasta pérdidas millonarias e incluso la cancelación de la licencia para poder operar.

La falta de documentación apropiada puede ser la causa de errores en el diseño de una modificación de la infraestructura existente. Cuando se produce una falla, el no tener la documentación adecuada puede aumentar los tiempos de respuesta de los equipos de soporte. En general, el contar con la documentación de la arquitectura de los sistemas permite a los equipos de trabajo tener una visión general y un punto de partida para tomar decisiones ante contingencias que puedan involucrarlos.

Cuando se habla de documentación formal de la arquitectura de software, existen varios modelos que se basan en la presentación de vistas que describen cómo se estructura. Algunos tienen un número fijo de vistas, y otros permiten acotar este número de acuerdo a un análisis de las necesidades existentes.

En este trabajo, se propone una metodología para aplicar y verificar la utilidad de uno de los modelos de documentación existentes: el modelo de vistas del Software Engineering Institute (SEI), que busca identificar y documentar las vistas de arquitectura de software más adecuadas, de acuerdo a los requerimientos de calidad de los stakeholders de un sistema en estudio.

Para realizar lo anterior, se seleccionó un sistema utilizado en un Banco, se entrevistó a sus stakeholders para conocer cuáles son sus intereses y con ello se generaron las vistas de arquitectura que propone el modelo. Finalmente, se presentan 3 casos donde se valida que la documentación de las vistas de arquitectura generadas en base al modelo mejoran la toma de decisiones de sus stakeholders, comparándola con el uso de la documentación existente antes de realizar este trabajo.

Este trabajo entrega una metodología práctica para aplicar el modelo de vistas del SEI, aprovechando principalmente la relación existente entre sus guías de estilos y los atributos de calidad que apoyan, orientando la documentación de la arquitectura de sistemas a las necesidades específicas de sus stakeholders. Para la comunidad académica y profesional sirve como caso de estudio de aplicación del modelo y, para el Banco involucrado, como el punto de partida para incorporarla en su ciclo de desarrollo de proyectos tecnológicos.

Dedicatoria

A mi familia, que ha sido un pilar y la inspiración para superarme día a día, a mis padres que han me dieron la base para tener mis primeros logros, y a todos aquellos que de alguna manera contribuyeron a que este trabajo fuera una realidad.

Tabla de Contenido

Índice de Figuras

Índice de Tablas

CAPÍTULO I.
INTRODUCCIÓN

1.1. El Banco y la Dificultad de Administrar la Complejidad de sus Sistemas

1.1.1. Servicios Financieros y su Administración

Existe un Banco, cuyo objetivo es proveer servicios financieros de excelencia, con soluciones creativas y efectivas para cada segmento de clientes, que aseguren la permanente creación de valor para sus accionistas. Entre los productos ofrecidos se consideran, entre otros, Cuentas Corrientes, Líneas de Crédito, Tarjetas de Débito, Tarjetas de Créditos, Créditos Hipotecarios, Créditos de Consumo, Cuentas de Ahorro, Depósitos a Plazo, Seguros, Transferencias Bancarias, Pago de Cuentas, Compra y Venta de Acciones, Fondos Mutuos, etc. La administración de estos productos, es apoyada por una serie de procesos operativos y sistemas informáticos que deben interactuar integradamente para cumplir con el de la organización.

1.1.2. El Ambiente Competitivo de los Servicios Financieros

Las fuerzas que actúan sobre el negocio bancario exigen que sus participantes desarrollen nuevos servicios y variaciones de los existentes para diferenciarse, atraer nuevos clientes y mantener los actuales. Para ello, las unidades de negocio generalmente necesitan modificar procesos operativos y sistemas o implementar nuevas tecnologías. Es importante que estas mejoras se realicen en el menor tiempo posible, ya que el ser los primeros en el mercado es una ventaja competitiva y es crítico para maximizar las utilidades que entrega el negocio.

1.1.3. Calidad de los Procesos y Sistemas

Por otra parte, también es importante la calidad de los procesos operativos y sistemas que soportan los procesos de negocio, ya que el descuido en algún atributo de calidad, puede producir pérdidas millonarias a la institución.
En base a los atributos de calidad, se puede realizar un análisis del impacto que tendría la degradación de alguno. Por ejemplo:

- Performance:
    - La degradación de los tiempos de respuesta de un sistema puede provocar que algún proceso no termine a tiempo, y como consecuencia, se apliquen multas por no cumplir con compromisos comerciales o normativos.
    - La degradación de los tiempos de respuesta de un sistema, puede generar una cancelación de una o más transacciones y sobrecargar a otros sistemas.
- Disponibilidad:

    - Una caída de un sistema, puede impactar la disponibilidad de múltiples servicios y generar pérdidas millonarias, al no concretarse una transacción.
    - Mientras más tiempo tarde la recuperación de un sistema, más transacciones se dejan de realizar.
    - Si la caída dura varios días, el Banco puede perder la licencia para operar.
    - Una caída de un sistema, puede impactar los servicios de otros Bancos y generar multas al Banco responsable. Por ejemplo, los sistemas de pagos automáticos de cuentas que se integran con la mayoría de los Bancos.
    - La disponibilidad de los servicios de un Banco, afecta su imagen y provoca pérdidas de Clientes.
- Seguridad:
    - Al tratarse de una institución financiera que resguarda los bienes de sus clientes, una falla de seguridad en sus sistemas, se traduce en desconfianza y pérdida de clientes.
- Modificabilidad:
    - Los avances tecnológicos han sido habilitadores de nuevos servicios. Por ejemplo: transferencias de dinero utilizando teléfonos celulares, compras con tarjeta de débito a través de internet, etc. El costo de implementación de estos nuevos servicios, está relacionado con la facilidad de modificación e integración con los sistemas existentes.
- Usabilidad:
    - Cada día el ser humano se ve enfrentado a nuevos productos tecnológicos que buscan hacerle la vida más fácil. El presentar productos fáciles, intuitivos y con la información adecuada permite un mayor número de interacciones con sus clientes y una impresión de buen servicio.

El análisis de los atributos de calidad de los procesos y sistemas del Banco, permite dimensionar el impacto que puede generar la falta de alguno de ellos, pero ¿cómo tener una visión de alto nivel que permita adelantarse o enfrentar un incidente? Por ejemplo, ante un terremoto en el que más de un sistema puede dejar de funcionar ¿cómo identificar los servicios que fueron impactados? ¿Cómo identificar cuáles son los posibles puntos donde se cortó la comunicación entre los sistemas? Lo primero que podríamos desear, es tener documentación que indique cómo están estructurados los sistemas de tal manera que sea fácil comprender las relaciones entre sus elementos principales. Por ello, se hace necesario que cada sistema se acompañe de una documentación su arquitectura que sea útil para sus stakeholders.

1.2. Objetivos

1.2.1. Objetivo General

Realizar una especificación costo-efectiva que permita evaluar la utilidad de un modelo de documentación de arquitecturas de software, aplicado a un sistema existente en un Banco.

1.2.2. Objetivos Específicos

- Construcción de la Documentación de la Arquitectura de un Sistema utilizando un modelo de documentación de arquitecturas de software.
- Establecer la relación entre las necesidades de los stakeholders y las vistas de arquitectura del sistema en estudio.
- Comprobar que la documentación desarrollada permite realizar las modificaciones necesarias para los stakeholders de una forma más costo-efectiva que antes de contar con ella.

Para lograr estos objetivos, en este trabajo de tesis se comienza por realizar una definición de las características que debe tener la documentación de arquitecturas de software, y cómo estas pueden ser desarrolladas a través de la aplicación de algún modelo teórico que ayude a identificar vistas de arquitectura, como los que proponen Clements et al en [1], Kruchten en [2], Soni en [3] y otros. Basado en las características de los modelos propuestos por los autores antes mencionados, se selecciona el modelo del Software Engineering Institute (SEI) por ser un método práctico para documentar arquitecturas de software y, utilizando una metodología que hace uso de este modelo y fue propuesta en este proyecto, se documenta la arquitectura de un sistema existente en un banco. El sistema seleccionado corresponde a System Center Configuration Management 2007, que es el sistema que administra la configuración de las estaciones de trabajo o workstations pertenecientes al banco en estudio. Finalmente, se valida la utilidad del modelo seleccionado, proponiendo tres escenarios probables, sugeridos por los stakeholders del sistema:

- Aumento de la cantidad de workstations existentes en el banco
- Errores en la distribución e instalación de software
- Problemas en el control remoto de workstations

Estos escenarios permitirán comparar los beneficios de la documentación realizada, con las posibilidades existentes antes de construir la documentación de la arquitectura del sistema seleccionado.

1.3. Organización del Documento

Este documento de proyecto de tesis se compone de seis capítulos comenzando por su introducción:

- En el CAPÍTULO II, se explica qué características debe tener la documentación de arquitecturas de software, justificando la utilización de vistas junto con un catálogo de elementos, sus relaciones y propiedades, para cumplir con los atributos de calidad que deben priorizarse de acuerdo a las necesidades de los stakeholders del sistema.
- En el CAPÍTULO III, se realiza un breve análisis de algunos modelos de vistas de arquitectura y se propone utilizar el modelo de vistas del SEI [1], junto a una notación estándar como UML 2.0, para documentar la arquitectura de un sistema existente en un banco. Además, se realiza un análisis de la relación existente entre los estilos de cada tipo de vista propuesta y los atributos de calidad que soportan.

- En el CAPÍTULO IV, se aplica el modelo del SEI identificando el sistema, sus stakeholders y proponiendo una metodología para seleccionar y construir las vistas de arquitectura relacionadas a los atributos de calidad requeridos.
- En el CAPÍTULO V, se valida la utilidad del modelo del SEI comparando los beneficios entregados por la documentación de la arquitectura realizada versus la documentación existente antes de la realización de este proyecto.

Finalmente, se presentan las conclusiones del trabajo realizado y una serie de anexos, útiles para profundizar los contenidos presentados.

CAPÍTULO II.

DOCUMENTACIÓN DE ARQUITECTURAS DE SOFTWARE

Una Arquitectura de Software de un sistema, es la estructura o estructuras del sistema, que consiste en elementos, sus propiedades externas visibles, y las relaciones entre ellos [1]. Esta definición sugiere que todos los elementos, propiedades visibles y las relaciones, pueden ser representados para describir un sistema, pero si se intenta describir un sistema con muchos elementos, esta tarea resulta muy compleja y surge la necesidad de agruparlos en diferentes vistas que nos permitan reducir esta complejidad.

La documentación de una Arquitectura de Software se utiliza principalmente para la educación, la comunicación entre arquitectos de software, ingenieros, diseñadores, implementadores y, en general, por todos los participantes del ciclo de vida del software, y sirve como la base para su análisis y el diseño.

2.1. Vistas de Arquitectura

Una vista de arquitectura, es una representación de un conjunto de elementos del sistema y las relaciones asociadas a ellos. El documentar la arquitectura de un sistema, implica documentar las vistas relevantes desde las cuales el sistema puede ser observado. Por ejemplo: un desarrollador debería tener una vista que muestre los elementos de código, un vendedor de sistemas, debería tener vistas que destaquen las funcionalidades del sistema, etc. Estas vistas, pueden incluirse en paquetes que explican cómo se relacionan, donde cada una expresa atributos de calidad diferentes, para en su conjunto representar todo o una parte del sistema.

La documentación de una vista debe contener una representación, preferentemente gráfica, que muestra los elementos y relaciones del sistema principal, un catálogo que define los elementos y sus propiedades, una especificación de las interfaces de los elementos y su comportamiento, su justificación e información del diseño.

La documentación de un paquete, debe incluir una presentación del paquete, información de cómo se relacionan sus vistas y de todo el sistema en su conjunto, limitaciones y la justificación de la arquitectura general y, finalmente, la información de gestión necesaria para mantener el paquete completo.

2.2. Atributos de Calidad de Software

Según Barbacci et al. [9] la calidad de software se define como el grado en el cual el software posee una combinación deseada de atributos. Tales atributos son requerimientos adicionales del sistema, que hacen referencia a características que éste debe satisfacer y que raramente son documentados. Estas características o atributos se conocen con el nombre de atributos de calidad, los cuales se definen como las propiedades de un servicio que presta el sistema a sus usuarios.

Las decisiones de arquitectura, impactan directamente los atributos de calidad de un sistema. Por ello, los requisitos de calidad de un sistema sirven de guía para el diseño de su arquitectura (Bass et al. [10]).

El estándar ISO/IEC 9126, ha sido desarrollado en un intento de identificar los atributos clave de calidad para productos de software. Este estándar, identifica seis características básicas de calidad, que pueden estar presentes en cualquier producto de software. El estándar provee una descomposición de las características en subcaracterísticas, que se muestran en la Tabla II-1.

Sin embargo, no todos los atributos de calidad se relacionan directamente con la arquitectura de software. Según Stephen T. Albin [7] y Losavio et al. [13], los atributos de calidad que se relacionan directamente con la arquitectura son:

- Funcionalidad
- Confiabilidad
- Eficiencia
- Mantenibilidad
- Portabilidad

Losavio et al. [13] plantean que la característica de usabilidad propuesta por el modelo ISO/IEC 9126 puede ser refinada para obtener atributos que se relacionan con los componentes de la interfaz con el usuario. Dado que estos componentes son independientes de la arquitectura, no son considerados en su adaptación del modelo.

Atributo de Calidad	Descripción del Atributo	Característica	Descripción de la Característica
Funcionalidad	Grado en que las necesidades asumidas o descritas se satisfacen.	Idoneidad	Indica el grado en que las funciones que soportan las tareas especificadas están presentes.
		Precisión	Indica el grado de exactitud de los efectos del sistema (por ejemplo la de sus salidas).
		Interoperabilidad	Indica el grado en que el sistema puede interactuar con otros sistemas.
		Seguridad	Indica el grado en que un acceso no autorizado (accidental o deliberado) se prevenga y se permita un acceso autorizado.
Confiabilidad	Grado en que el sistema responde bajo las condiciones definidas durante un intervalo de tiempo dado.	Madurez	Indica la frecuencia con que ocurren los fallos.
		Tolerancia a Fallas	Indica el grado en que el sistema mantiene un nivel de respuesta ante fallos del sistema o interfaces.
		Recuperabilidad	Indica la capacidad del sistema para restablecer su nivel de respuesta después de un fallo crítico o error de hardware.
Usabilidad	Conjunto de características que influyen en el esfuerzo requerido para el uso y la evaluación individual de cada uso por parte de un conjunto de usuarios dados.	Comprensión	Indica las características del software que influyen en el esfuerzo del usuario para reconocer el concepto lógico y su aplicación.
		Capacidad de Aprendizaje	Indica las características del software que influyen en el esfuerzo del usuario para aprender su aplicación (por ejemplo: control, entrada, salida).
		Operabilidad	Indica las características del software que influyen en el esfuerzo del usuario para operar y su control operacional.
Eficiencia	Conjunto de características que determinan la relación entre el nivel de rendimiento del software y el número de recursos usados, bajo ciertas condiciones dadas.	Comportamiento en el Tiempo	Indica las características del software que influyen en el tiempo de respuesta y procesamiento, y productividad cuando se ejecuta su función.
		Comportamiento de Recursos	Indica las características del software que influyen en el número de recursos usados, y la duración de su uso, cuando se lleva a cabo su función.
Mantenibilidad	Esfuerzo requerido para implementar cambios.	Analizabilidad	Indica la cantidad de esfuerzo requerido para diagnosticar la causa de un fallo.
		Modificabilidad	Indica la cantidad de esfuerzo requerido para modificar o eliminar un defecto.
		Estabilidad	Indica volumen de riesgos de efectos inesperados tras una modificación.
		Capacidad de Prueba	Indica la capacidad del software para permitir que sea validado tras ser modificado.
Portabilidad	Conjunto de características que determinan la capacidad del software para ser transferido de un entorno de operación a otro.	Adaptabilidad	Indica las características del software que influyen en las posibilidades de adaptación a diferentes entornos especificados, sin realizar otras acciones que las Indicadas para este propósito.
		Instalabilidad	Indica las características del software que influyen en el esfuerzo requerido para instalar el software en un entorno especificado.
		Reemplazabilidad	Indica las características del software que influyen en la posibilidad y esfuerzo requerido para usarlo en lugar de otro software en el mismo entorno.

Tabla II-1 Atributos de Calidad según ISO/IEC 9126

Característica	Subcaracterística	Elementos de tipo arquitectónico
Funcionalidad	Adecuación	Refinamiento de los diagramas de secuencia
	Exactitud	Identificación de los componentes con las funciones responsables de los cálculos
	Interoperabilidad	Identificación de conectores de comunicación con sistemas externos
	Seguridad	Mecanismos o dispositivos que realizan explícitamente la tarea
Confiabilidad	Tolerancia a fallas	Existencia de mecanismos o dispositivos de software para manejar excepciones
	Recuperabilidad	Existencia de mecanismos o dispositivos de software para restablecer el nivel de desempeño y recuperar datos
Eficiencia	Desempeño	Componentes involucrados en un flujo de ejecución para una funcionalidad
	Utilización de recursos	Relación de los componentes en términos de espacio y tiempo
Mantenibilidad	Acoplamiento	Interacciones entre componentes
	Modularidad	Número de componentes que dependen de un componente
Portabilidad	Adaptabilidad	Presencia de mecanismos de adaptación
	Instalabilidad	Presencia de mecanismos de instalación
	Coexistencia	Presencia de mecanismos que faciliten la coexistencia
	Reemplazabilidad	Lista de componentes reemplazables para cada componente

Tabla II-2 Atributos de Calidad y subcaracterísticas asociadas con elementos de tipo arquitectónico

La Tabla II-2 presenta los atributos de calidad planteados por Losavio et al. [13], que poseen subcaracterísticas asociadas con elementos de tipo arquitectónico.

Para apoyar cada atributo de calidad existen tácticas, por ejemplo, si se requiere asegurar la disponibilidad de un sistema, se debe prever que los tipos de fallas estén cubiertos por un mecanismo que disminuya el impacto que les genera. Las tácticas comunes para fortalecer atributos específicos se describen en detalle en Bass et al. [10].

2.3. Consecuencias de la Priorización de Atributos de Calidad

Cuando Bass et al. [10] hablan de stakeholders, se refieren a cada participante en el ciclo de vida del software. Además de los requisitos funcionales, los stakeholders tienen intereses en los atributos de calidad de un sistema. Por ejemplo:

- Al administrador de la compañía que lo desarrolla, le interesa que el personal de desarrollo le entregue el mayor valor agregado al producto.
- Al encargado de marketing de la empresa de desarrollo, le interesa que el producto tenga características que lo destaquen de la competencia.
- Al usuario final, generalmente, le interesa que tenga todas las funcionalidades que le permitan realizar sus tareas, que la información esté disponible cuando es requerido, que sea confiable, fácil de usar, etc.
- Para el encargado de mantenimiento lo importante podría ser que el sistema fuera fácil de modificar.
- El cliente que paga los costos de implementar el sistema, busca que el desarrollo sea rápido y sin muchos cambios que encarezcan la implementación o mantenibilidad.

La tarea de plasmar todas estas características es responsabilidad del arquitecto de sistemas, comprendiendo las restricciones reales del entorno en que se implementará, administrando las expectativas de calidad de los interesados, negociando prioridades y generando decisiones.
Cuando se toma una decisión de arquitectura para cubrir una meta de negocio, se impacta implícitamente uno o más atributos de calidad. El resultado de una serie de decisiones, entregará una arquitectura de software que deberá cumplir con los atributos de calidad de la arquitectura, priorizados de acuerdo a los requerimientos de los participantes en el ciclo de vida del sistema (Tabla II-3).

	Funcionalidad	Confiabilidad	Eficiencia	Mantenibilidad	Portabilidad
Funcionalidad		-		-	
Confiabilidad			-		
Eficiencia				-	-
Mantenibilidad			-		
Portabilidad			-		

Tabla II-3 Matriz de Impacto de la mejora de un atributo de calidad sobre otro

En la Tabla II-3, se muestra un análisis del impacto que provoca una mejora aislada en un atributo de calidad (columna izquierda) sobre otros atributos de calidad (fila superior) dependientes de la arquitectura. En ella, se indica que una mejora o aumento de las funcionalidades que un sistema debe realizar, impacta negativamente en la confiabilidad del sistema, ya que al aumentar el número de procesos que realiza, aumenta la posibilidad de que existan puntos de falla no identificados durante las etapas de desarrollo y pruebas. Del mismo modo, un aumento en las funcionalidades de un sistema puede impactar negativamente su mantenibilidad, debido a las múltiples relaciones que pueden existir entre cada función que realice.
El mejorar la confiabilidad de un sistema implica incluir una serie de validaciones que impactan negativamente su eficiencia.

La mantenibilidad de un sistema puede verse afectada negativamente al mejorar la eficiencia, cuando esta mejora es producto de optimizaciones de código que dificultan el entendimiento humano. Asimismo, si se aumenta la eficiencia a través de la eliminación de elementos que validan la compatibilidad entre plataformas, se genera un impacto negativo en la portabilidad del sistema.

El incrementar la mantenibilidad de un sistema para que sea fácilmente modificado por otros, implica generar componentes cuya lógica sea lo más humanamente legible. Esta facilidad de lectura, puede impactar negativamente la eficiencia de las funciones de un componente.

El incrementar la portabilidad de un sistema, implica generar una serie de validaciones que afectan negativamente la eficiencia de sus componentes.

En el ciclo de desarrollo de software, una matriz de impacto como la descrita en la Tabla II-3 ayudará a priorizar los requerimientos de calidad de los stakeholders relacionados con su arquitectura. Para el caso de estudio de esta tesis, esta matriz ayudará a definir los elementos que deben incorporarse en la documentación de la arquitectura del sistema. Por ejemplo, si un stakeholder requiere que el sistema privilegie la eficiencia de un sistema, la documentación de la arquitectura deberá contemplar elementos como tasas de transferencia de datos entre componentes y los tiempos de procesamiento de cada uno de ellos.

CAPÍTULO III.
MODELO DE VISTAS DE ARQUITECTURA

Un Modelo de Vistas de Arquitectura o Framework de Arquitectura de Software [7], es un conjunto de especificaciones de puntos de vista y sus relaciones. Estos modelos son utilizados para describir una clase de sistemas, como por ejemplo sistemas de procesamiento distribuidos o sistemas orientados a objetos.

	Foco Principal	Características	Puntos de Vista
Modelo de Vistas del SEI [1]	Comunicación	Puntos de Vista Independientes	3 puntos de vista
Modelo de Vistas "4+1" [2]	Diseño	Proceso de Diseño Interactivo	5 Vistas
Modelo de Cuatro Vistas de Siemens [3]	Diseño	Flujo de Información a través de puntos de vista	4 Vistas
RM-ODP [4]	Reuso	Define un vocabulario común	5 puntos de vista

Tabla III-1 Algunos Modelos de Vistas de Arquitectura [5]

Existen varios modelos de vistas de arquitectura; algunos de ellos son los descritos en la Tabla III-1. Los detalles de cada uno de los modelos pueden encontrarse en la bibliografía o en los anexos de este documento y servirán como material de apoyo para la documentación del sistema en estudio. Sin embargo, este trabajo de tesis se basará principalmente en el Modelo de Vistas del SEI por ser un método práctico de documentación de arquitecturas basado en el estándar IEEE 1471 [6] y se centra en los siguientes aspectos:

- Los arquitectos deben documentar las vistas que sean de mayor utilidad y no ajustarse a un número fijo de vistas, como es el caso de otros como el modelo de vistas 4+1.
- Se debe documentar la arquitectura tomando en cuenta los intereses y necesidades de las personas involucradas en el proyecto, estos intereses corresponden principalmente a atributos de calidad que debe poseer el sistema.

3.1. Modelo de Vistas del SEI

Para el SEI, el documentar una arquitectura de software es una cuestión de seleccionar un conjunto relevante de vistas, documentar cada una de ellas, y luego, documentar aquella información que

aplica a más de una vista. En su propuesta en 2002 [1], existen tipos de vistas que representan las perspectivas que un arquitecto debe considerar cuando se diseña un sistema:

- **Tipo de Vista de Módulos:** muestra la estructura estática del sistema.
- **Tipo de Vista de Componentes y Conectores (C&C):** muestra la estructura dinámica del sistema.
- **Tipo de Vista de Asignación:** muestra las relaciones con los ambientes externos al sistema.

Cada tipo de vistas tiene asociado un conjunto predefinido de estilos de arquitectura [1] mostrados en la Figura III-1, en los cuales los arquitectos pueden basarse para representar una o varias vistas que permitan documentar la arquitectura de algún sistema. Un estilo es una especialización de un tipo de vista y refleja patrones recurrentes de interacción, independientes de cualquier sistema.

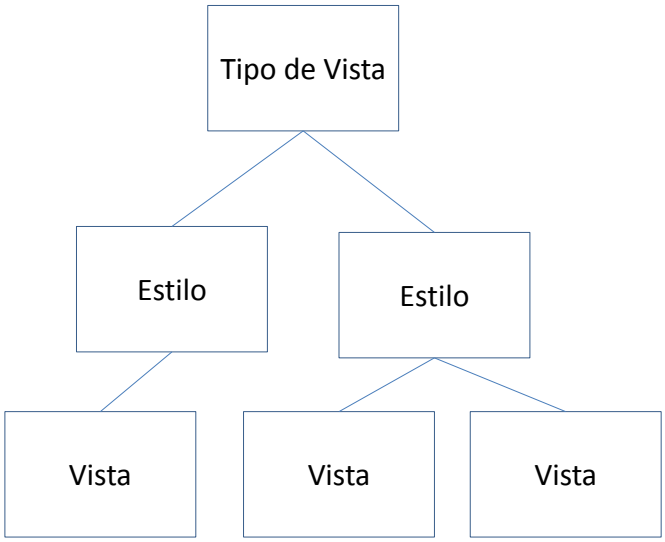


Figura III-1 Relación entre Tipos de Vistas, Estilos y Vistas

Dentro de los límites de un estilo, se describe cómo los elementos de un estilo están vinculados a los elementos de un sistema, a estas descripciones se les llama vistas. Con ello, la documentación de una vista de arquitectura de un sistema corresponderá a la combinación de un conjunto de estilos, algunos de esos estilos se indican como referencia en la Tabla III-2.

Tipo de Vista	Estilo	Vistas
Módulos	Descomposición	La arquitectura de cada sistema posee múltiples vistas que utilizan estilos para representar cómo se resuelven las necesidades de calidad de los stakeholders.
	Generalización	
	Usos	
	Capas	
Componentes y Conectores	Tuberías y Filtros	
	Datos Compartidos	
	Comunicación de Procesos	
	De Pares	
	Cliente Servidor	
Asignación	Despliegue	
	Aplicación	
	Asignación de Trabajo	

Tabla III-2 Resumen de Tipos de Vistas y sus Estilos

En el modelo del SEI, la documentación de un estilo se realiza a través de una estructura estándar llamada guía de estilos, que organiza los tipos de elementos, de relaciones y sus propiedades. La definición para las secciones más importantes de una guía de estilos es la siguiente:

- Resumen: explica por qué el estilo es útil y qué tipos de sistemas se clasifican en él.
- Elementos: define los bloques de construcción naturales para el estilo.
- Relaciones: determinan cómo los elementos interactúan para hacer que un sistema funcione. La descripción de una relación nombra las relaciones entre elementos y entrega reglas de cómo los elementos pueden y no pueden estar relacionados.
- Propiedades: entregan información adicional de los elementos y relaciones. En la guía de estilos se incluye el nombre y la descripción de las propiedades, las que al momento de documentar una arquitectura en base a una vista del estilo deben ser completadas para permitir, por ejemplo, el análisis de atributos de calidad de una solución.
- Restricciones: lista las reglas para ubicar elementos y relaciones en una instancia del estilo.
- Para qué sirve: describe las clases de inferencias que pueden obtenerse de una instancia del estilo.
- Notaciones: es una guía que entrega una descripción gráfica o textual de las representaciones que comúnmente se realizan para documentar vistas del estilo.
- Relación con otros estilos: describe cómo las vistas del estilo pueden ser relacionadas a vistas de otros estilos.

Cada organización debe documentar sus propias guías de estilos especializadas, que deriven de los estilos propuestos por el SEI, agregando propiedades a los elementos de los estilos utilizados de acuerdo a sus necesidades. Por ejemplo, se puede agregar la propiedad de unidades de software de una capa, que es un elemento del estilo de capas perteneciente al tipo de vistas de módulo.
Antes de comenzar a utilizar estilos para representar una Arquitectura de Software, es necesario comprender el contexto del problema a través de los requisitos funcionales y de calidad.

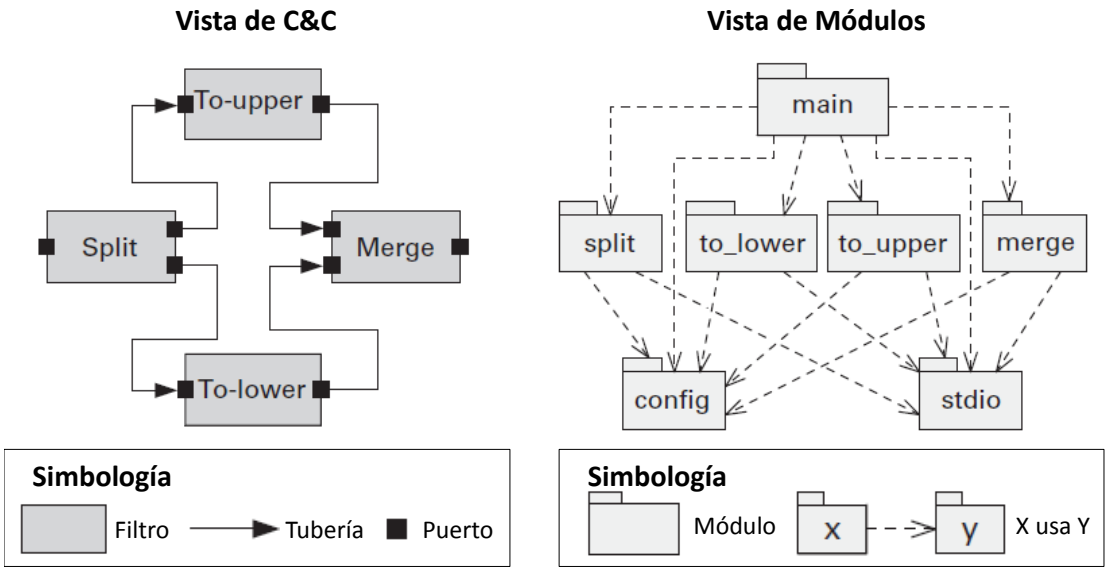


Figura III-2 Vistas de C&C y de Módulos de un sistema representado en Clements et. al. [1]

Como un ejemplo de este modelo de vistas de arquitectura, en la Figura III-2 se muestra la vista de C&C y la vista de módulos de un sistema simple que acepta una secuencia de caracteres como entrada, y entrega una nueva secuencia pero con sus caracteres alternando mayúsculas y minúsculas. En la figura se puede observar que el tipo de vista de C&C muestra las interacciones en tiempo de ejecución de los componentes, mientras que la vista de módulos muestra el diseño de los módulos y sus relaciones.

Vistas de Asignación

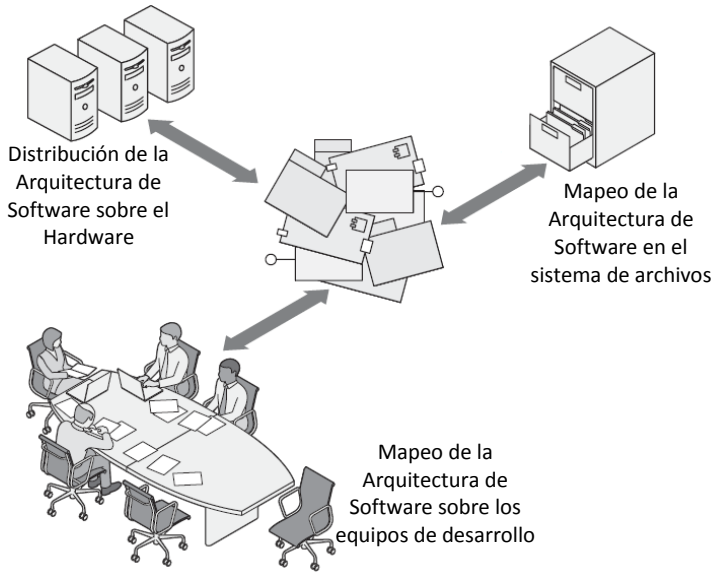


Figura III-3 Objetivos de las vistas de asignación

No sólo es necesario saber cómo están distribuidos los componentes del sistema en tiempo de diseño y ejecución, también se debe conocer los elementos que rodean al sistema. En el modelo del

SEI, estos elementos se muestran en las vistas de asignación representadas en la Figura III-3 donde se debe responder a:

- ¿Cómo se distribuye la arquitectura en la infraestructura de hardware?
- ¿Cómo se distribuye en el sistema de archivos del ambiente productivo?
- ¿Cómo se distribuye en los equipos de trabajo?

3.1.1. Atributos de Calidad del Tipo de Vista de Módulos

En general, los estilos de tipo de vista de módulos entregan un modelo para el código fuente, y ayudan a la trazabilidad de requerimientos para poder determinar cómo los requerimientos funcionales son enlazados con las responsabilidades de cada módulo. También, se utilizan para realizar análisis de impacto que buscan predecir el efecto de modificaciones al sistema y para la comunicación de las funcionalidades que tiene un sistema a los integrantes de un equipo de proyecto.
En los tipos de vista de módulos, el SEI propone los siguientes estilos:

- El **estilo de descomposición** busca una representación top-down de los sistemas, dividiéndolos en partes comprensibles que ayudan a: realizar análisis de impacto de las modificaciones, la distribución de trabajo y el aprendizaje y exploración de las funcionalidades del sistema.
- El **estilo de usos** busca identificar las dependencias entre módulos, lo cual permite planear modificaciones en desarrollos incrementales. Además, la identificación de las dependencias entre los módulos sirve de guía para los procesos de migración e instalación de módulos.
- El **estilo de generalización** busca apoyar la reutilización de los módulos apoyando la realización de cambios locales.
- El **estilo de capas** busca representar las dependencias entre las capas de abstracción que componen los sistemas.
- El **estilo de aspectos** busca simplificar la representación de funcionalidades transversales al sistema.
- El **estilo de modelo de datos** busca identificar cómo se estructuran los datos entregando elementos que ayudan a reorganizarlos en busca de mayor eficiencia.

La Tabla III-3 ayuda a identificar si el tipo de vista de módulo es adecuado para documentar la arquitectura de un sistema en base a los atributos de calidad que buscan sus stakeholders. Las tres últimas columnas (eficiencia, mantenibilidad y portabilidad) corresponden a los atributos de calidad que apoya el tipo de vista de módulo. Cuando se conocen los atributos de calidad y las características que buscan los stakeholders, esta tabla indica los estilos de vista que deberían utilizarse cuando se define y/o documenta la arquitectura de un sistema. Por ejemplo: si los stakeholders buscan que la documentación de la arquitectura de un sistema apoye la mantenibilidad facilitando la identificación del impacto de una modificación, se encontraría que el estilo de descomposición ayuda a realizar análisis de impacto de las modificaciones a los sistemas, por lo que generar las vistas de descomposición sería la elección.

Estilo	Eficiencia	Mantenibilidad	Portabilidad
Estilo de Descomposición		• **Analizabilidad**: Ayuda a realizar Análisis de Impacto de modificaciones. • **Modificabilidad**: Ayuda a distribuir los módulos entre equipos de desarrollo, y al aprendizaje y exploración de los componentes de sistemas.	
Estilo de Usos		• **Analizabilidad**: Ayuda a identificar dependencias de módulos • **Modificabilidad**: Ayuda a la planeación de desarrollos incrementales y extensiones de sistemas.	• **Instalabilidad**: Ayuda en procesos de migración e instalación de módulos.
Estilo de Generalización		• **Modificabilidad**: Ayuda al reuso de módulos, extensiones de sistemas y cambios locales en los elementos hijos	
Estilo de Capas		• **Modificabilidad**: Ayuda a reducir el impacto de modificaciones y a la asignación de trabajo.	• **Adaptabilidad**: Ayuda a aislar las modificaciones que deben realizarse para portar sistemas a otras plataformas y facilita el desarrollo de interfaces de usuario para múltiples plataformas.
Estilo de Aspectos		• **Modificabilidad**: Ayuda a evitar la mezcla de funcionalidades transversales, reduciendo el impacto de las modificaciones	
Estilo de Modelo de Datos	• **Comportamiento de Recursos:** Ayuda a mejorar el modelo de datos a través de desnormalizaciones, optimizaciones, y otras decisiones de diseño.	• **Modificabilidad:** Ayuda a realizar análisis de modificabilidad y a escribir código a los desarrolladores.	

Tabla III-3 Relación entre Atributos de Calidad y los Estilos del Tipo de Vista de Módulos

3.1.2. Atributos de Calidad del Tipo de Vista de Componentes y Conectores

Las vistas de componentes y conectores son usadas principalmente para analizar los atributos de calidad del sistema, que tienen presencia principalmente durante el tiempo de ejecución, como la **eficiencia**, **confiabilidad**, y **mantenibilidad**.
En los tipos de vista de componentes y conectores, el SEI propone los siguientes estilos:

- El **estilo de flujo de datos** busca identificar la secuencia de procesos que realizan transformaciones sucesivas sobre los datos, apoyando la mantenibilidad de sus componentes, su reutilización y posibilitando su distribución en diferentes unidades de procesamiento.
- El **estilo de llamada y retorno** representa un modelo de procesamiento en el cual los componentes entregan un conjunto de servicios u operaciones, que pueden ser invocados por otros componentes, y donde se garantiza que cada llamada entrega una respuesta. Este modelo apoya la eficiencia y la mantenibilidad.
- El **estilo basado en eventos** permite comunicar componentes a través de mensajes asíncronos. Los sistemas generalmente se organizan como una distribución pobremente acoplada de componentes que generan comportamientos en otros componentes a través de eventos.
- El **estilo de repositorio o centrado en datos** enfatiza la integrabilidad de los datos apoyando su confiabilidad y disponibilidad, permitiendo que los productores estén desacoplados de los consumidores.
- El **estilo de comunicación de procesos** se distingue por la interacción de componentes ejecutándose concurrentemente a través de varios mecanismos de conexión, permitiendo la implementación de mecanismos de concurrencia.

La Tabla III-4, al igual que la Tabla III-5, ayuda a identificar si el tipo de vista de componentes y conectores es adecuado para documentar la arquitectura de un sistema, en base a los atributos de calidad que buscan sus stakeholders. Sus cuatro últimas columnas (Funcionalidad, Confiabilidad, Eficiencia y Mantenibilidad) corresponden a los atributos de calidad que apoyan el tipo de vista de componentes y conectores. Cuando se conoce el atributo de calidad, y las características que buscan los stakeholders, estas tablas ayudan a elegir los estilos que deberían utilizarse cuando se define y/o documenta la arquitectura de un sistema. Por ejemplo: si los stakeholders buscan que un sistema sea mantenible, permitiendo reconfiguraciones dinámicas que redirijan el flujo de consultas a componentes alternativos cuando un componente falla, sería útil documentar la arquitectura del sistema en base al estilo de arquitectura orientada a servicios.

Estilo	Especializaciones del Estilo	Funcionalidad	Confiabilidad	Eficiencia	Mantenibilidad
Estilos de Flujo de Datos	Estilo de Tuberías y Filtros		• **Tolerancia a Fallas:** Facilita la implementación de transacciones atómicas de grandes volúmenes de datos.	• **Comportamiento de Recursos:** Ayuda a la paralelización y distribución de filtros	• **Analizabilidad**: Ayuda a la simplificación del Análisis del comportamiento total de sistemas. • **Estabilidad**: Ayuda a la reutilización de componentes • **Modificabilidad**: Ayuda independizar los cambios en componentes individuales.
	Estilo Secuencial por Lotes				
Estilos de Llamada y Retorno	Estilo Cliente Servidor	• **Seguridad**: Facilita la implementación de mecanismos de autenticación a través del sistema.	• **Madurez**: Sus tipos de conectores ayudan al control de errores • **Tolerancia a Fallas:** Permite la replicación de servidores apoyando disponibilidad del sistema	• **Comportamiento de Recursos:** Promueve la escalabilidad agregando recursos para soportar un mayor número de clientes.	• **Modificabilidad**: Ayuda independizar los cambios en componentes individuales. • **Analizabilidad**: Ayuda al entendimiento del sistema • **Estabilidad**: Ayuda a la reutilización de componentes debido a la factorización de servicios comunes. • **Capacidad de Prueba:** Ayuda al análisis de dependencias ante fallas de algún componente
	Estilo de Pares		**Tolerancia a Fallas:** Facilita la implementación de conexiones redundantes en cualquiera de sus nodos.	• **Comportamiento de Recursos:** Ayuda a promover la escalabilidad permitiendo agregar pares sin impactos significativos y, con un despliegue adecuado de las aplicaciones, permite un uso más eficiente de CPU y Disco.	
	Estilo de Arquitectura Orientada a Servicios	• **Interoperabilidad:** Facilita la interoperabilidad, debido a que los proveedores y los consumidores de servicios pueden ser ejecutados en plataformas diferentes.			• **Modificabilidad**: Ayuda a realizar reconfiguraciones dinámicas cuando existe la necesidad de remplazar o agregar versiones de componentes sin interrumpir el uso de los sistemas.

Tabla III-4 Cómo apoyan los Estilos de Tipos de Vistas de C&C a los atributos de calidad

		Funcionalidad	Confiabilidad	Eficiencia	Mantenibilidad
Estilos Basados en Eventos	Estilo Publicación y Suscripción	• **Idoneidad:** Ayuda a entregar las funcionalidades principales para frameworks de interfaz de usuario.		• **Comportamiento de Recursos:** Permite una mayor concurrencia que el estilo de llamada y retorno debido a que los emisores de eventos no necesitan bloquearse mientras múltiples receptores pueden procesarlos.	• **Modificabilidad:** Permite la modificación de componentes ya que los componentes productores de eventos están desacoplados de los consumidores y ayuda a agregar nuevos destinatarios sin modificar a productores de eventos.
Estilos de Repositorios o Centrados en Datos	Estilo de Datos Compartidos		• **Tolerancia a Fallas:** Apoya la disponibilidad a través de la redundancia de datos.	• **Comportamiento en el Tiempo:** Apoya el rendimiento a través de la redundancia de datos.	• **Modificabilidad**: Permite desacoplar productores de datos de sus consumidores disminuyendo el impacto de las modificaciones de sus componentes.
Estilos Heterogéneos de C&C	Estilos de Comunicación de Procesos			• **Comportamiento de Recursos:** Permite la implementación de mecanismos de concurrencia	

Tabla III-5 Cómo apoyan los Estilos de Tipos de Vistas de C&C a los atributos de calidad (Continuación)

3.1.3. Atributos de Calidad del Tipo de Vista de Asignación

Los principales atributos de calidad que apoya el tipo de vista de asignación son la funcionalidad, confiabilidad, eficiencia y mantenibilidad del sistema.
En los tipos de vista de asignación, el SEI propone los siguientes estilos:

- El **estilo de despliegue** busca identificar donde se ejecutan los elementos de software que componen el sistema, apoyando al análisis de confiabilidad y eficiencia de uso de recursos.
- El **estilo de aplicación** busca identificar los elementos de configuración, que permiten organizar la infraestructura de desarrollo del sistema.
- El **estilo de asignación de trabajo** busca identificar cómo los equipos de trabajo se pueden organizar y distribuir los componentes del sistema.

La Tabla III-6 ayuda a identificar cuáles son los estilos del tipo de vista de asignación que deberían utilizarse al documentar arquitecturas de sistemas, en base a los atributos de calidad que buscan sus stakeholders. Las cuatro últimas columnas (funcionalidad, confiabilidad, eficiencia y mantenibilidad) corresponden a los atributos de calidad que apoya el tipo de vista de asignación. Cuando se conocen los atributos de calidad, y las características que buscan los stakeholders, esta tabla indica los estilos de vista que deberían utilizarse cuando se define y/o documenta la arquitectura de un sistema. Por ejemplo: si los stakeholders buscan que la documentación de la arquitectura de un sistema apoye la confiabilidad, facilitando la realización de análisis de impacto de los cambios que pueden hacerse al sistema, sería útil utilizar el estilo de despliegue.

Estilo	Funcionalidad	Confiabilidad	Eficiencia	Mantenibilidad
Estilo de Despliegue	• **Seguridad**: Ayuda a realizar análisis de seguridad del sistema	• **Tolerancia a Fallas:** Ayuda a realizar análisis de confiabilidad del sistema	• **Comportamiento en el Tiempo:** Ayuda a realizar análisis de performance del sistema	
Estilo de Aplicación				• **Capacidad de Prueba:** Ayuda a identificar las entidades de configuración de cada módulo del sistema, ya que relaciona los módulos con las entidades de configuración
Estilo de Asignación de Trabajo				• **Modificabilidad**: Ayuda a la selección y asignación de equipos de trabajo más idóneos para modificar el sistema, ya que relaciona los módulos con los equipos de trabajo.

Tabla III-6 Cómo apoyan los Estilos de Tipos de Vistas de Asignación a los atributos de calidad

3.2. Organización de la Documentación

Al hablar de documentación se piensa en uno o más documentos que describen algo. Estos documentos tienen una estructura pensada para que sea comprendida por un público determinado. En el caso de la documentación de una arquitectura, el SEI propone:

- **Diagrama de Contexto:** Muestra la relación entre el sistema o una porción de éste y su entorno.
- **Presentación Primaria:** Muestra los elementos y sus relaciones, generalmente de manera gráfica.
- **Catálogo de Elementos:** Contiene los detalles de los elementos, sus propiedades e interfaces.
- **Guía de Variabilidad:** Muestra los posibles puntos de variación en caso de que las vistas sean modificadas.
- **Antecedentes de la Arquitectura:** Explica la justificación de la arquitectura, sus supuestos, y los resultados de los análisis realizados.
- **Información Adicional:** En esta sección se incluyen las prácticas y políticas de la organización.
- **Paquetes de Vistas Relacionados:** En esta sección se explican las relaciones entre los distintos paquetes de vistas.

Por su parte, el IEEE Software propone el estándar IEEE 1471 en [6] que define un conjunto de recomendaciones centradas básicamente en dos ideas, un marco conceptual para describir arquitecturas, y un conjunto de prácticas a seguir. Si algún arquitecto le interesa que la documentación de sus arquitecturas cumpla con el estándar IEEE 1471, éste debe seguir las prácticas que el estándar recomienda.
La descripción de la arquitectura se organiza en un conjunto de vistas. Cada vista modela una parte del sistema y satisface uno o más intereses de las personas involucradas. Los distintos intereses se deben considerar durante la construcción del sistema. Si alguno de éstos no es considerado en al menos una de las vistas, se dice que la descripción de la arquitectura está incompleta.
Cada vista se documenta de acuerdo a un punto de vista determinado. En éste, se definen las notaciones, técnicas y reglas para construir e interpretar una vista. A su vez, éste determina cómo el contenido de una vista satisface uno o más intereses de las personas involucradas. Un punto de vista es un artefacto que puede ser reutilizado. En este marco conceptual, una vista es la instancia de un punto de vista dado. Un conjunto (biblioteca) de puntos de vista en el estándar IEEE 1471 es análogo a la guía de estilos que propone el SEI.

CAPÍTULO IV.
DOCUMENTACIÓN DE LA ARQUITECTURA DE UN SISTEMA

4.1. Introducción

Cuando ocurre un incidente que afecta a los usuarios de un sistema existente en el Banco, muchas veces es necesario tener una visión de alto nivel que permita tomar las mejores soluciones en el menor tiempo posible. Algunas de las capacidades que se requieren en esos casos son el identificar los componentes principales y la secuencia con que se ejecuta cada funcionalidad. Esta visión de alto nivel se describe en la arquitectura del sistema, y para representarla existen modelos y notaciones como los que se proponen en Clements, et al en [1], Kruchten en [2] y otros. Para validar la utilidad de uno de estos modelos, se eligió el Modelo de Vistas del SEI, propuesto por Clements, et al en [1], por ser un método práctico de documentación de arquitecturas basado en el estándar IEEE 1471 [5] y, como Lenguaje de Descripción de Arquitectura, se eligió UML 2.0 por ser el estándar de facto de modelamiento. El sistema seleccionado para aplicar el modelo, es el sistema de administración de plataforma que permite administrar las workstations de los empleados de un banco. Los usuarios finales de este sistema son todos los trabajadores del banco que tienen un PC a su cargo.

4.1.1. Metodología Propuesta

Para definir la metodología que permitirá identificar las vistas de arquitectura más adecuadas de acuerdo a los requerimientos de los stakeholders, se aprovechará la relación entre estilos de cada tipo de vista y los atributos de calidad que apoyan. Estas relaciones, que están explicadas en el capítulo II, ayudan a identificar los estilos más convenientes para representar las vistas de arquitectura de un sistema, basándonos en los atributos de calidad que buscan los stakeholders. Como resultado del análisis de esta relación, este proyecto de tesis permitió definir la metodología que muestra la Figura IV-1.

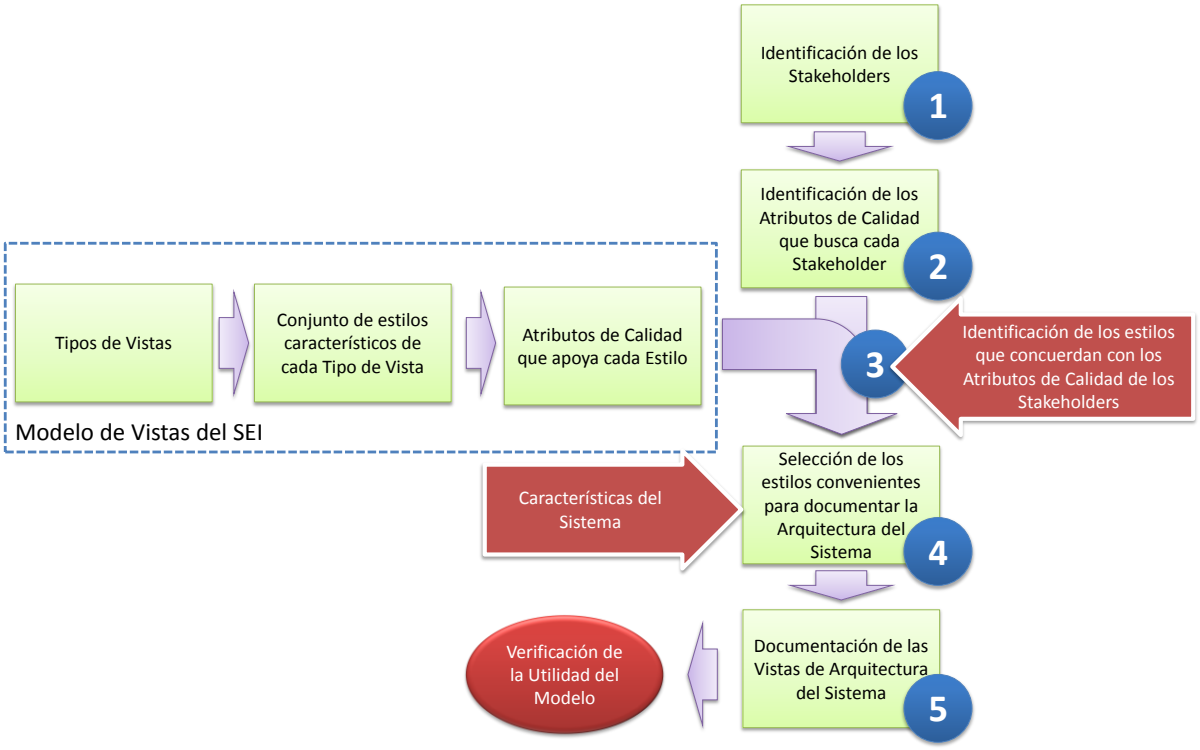


Figura IV-1 Metodología propuesta para documentar las vistas de arquitectura

La metodología propuesta en la Figura IV-1 sigue los siguientes pasos:

1. **Identificación de los stakeholders:** se identifican los roles y el grado de influencia que pueden ejercer cada uno de los stakeholders sobre el éxito del proyecto de documentación.
2. **Identificación de los atributos de calidad que busca cada stakeholder**: se busca conocer cuáles son los intereses de los stakeholders.
3. **Identificación de los estilos que concuerdan con los atributos de calidad de los stakeholders:** se relaciona la guía de estilos del modelo de vistas del SEI con los atributos de calidad requeridos por los stakeholders. Esto se desarrolla en dos etapas:

    3.1. **Identificación de los tipos de vistas candidatas:** se consideran los atributos de calidad apoya cada tipo de vista (ver CAPÍTULO III). Por ejemplo: si los stakeholders buscan eficiencia durante el tiempo de ejecución de los procesos, se debe elegir el Tipo de Vista de Componentes y Conectores para apoyar los análisis sobre el comportamiento en el uso de recursos que utiliza el sistema.

    3.2. **Identificación de los estilos de cada tipo de vista:** una vez que se ha seleccionado el o los tipos de vistas, se identifican los estilos particulares que apoyan los atributos de calidad requeridos por los stakeholders. Para ello se utilizan las tablas que incluye el CAPITULO II de este proyecto de tesis:

    - La Tabla III-3 para el caso de los Tipos de Vista de Módulos.

- La Tabla III-4 y la Tabla III-5 para el caso de los Tipos de Vista de Componentes y Conectores.
- La Tabla III-6 para el caso de los Tipos de Vista de Asignación.

El resultado de esta identificación genera una lista de estilos candidatos que podrían ser incluidos en las vistas de arquitectura.

4. **Selección de los estilos convenientes para documentar la arquitectura del sistema:** de los estilos candidatos identificados en el punto anterior, se seleccionan aquellos que concuerdan con las características del sistema a documentar. Por ejemplo, si los stakeholders buscan eficiencia en tiempo de ejecución, los estilos candidatos seleccionados en el punto anterior deberían ser 7 (Estilo de Tuberías y Filtros, Estilo Secuencial por Lotes, Estilo Cliente Servidor, Estilo de Pares, Estilo Publicación y Suscripción, Estilo de Datos Compartidos, y Estilos de Comunicación de Procesos), pero las características del sistema del ejemplo indican que se trata de un sistema que sigue el modelo Cliente-Servidor, por lo cual, la elección será utilizar el Estilo Cliente Servidor para representar sus vistas de arquitectura de C&C.
5. **Documentación de las Vistas de Arquitectura del Sistema:** seleccionado el o los estilos más adecuados para representar las vistas, se identifican las funcionalidades del sistema, de ser necesario se agrupan por similitud, y se diagraman de acuerdo a las características de los estilos.

Finalmente, se verificará la utilidad de la representación de cada vista, proponiendo modificaciones para evaluar el impacto considerando los escenarios antes y después de contar con la documentación. El resumen de este proceso se ilustra en la Figura IV-1.

4.1.2. Criterios de Éxito

Los criterios de éxito al representar las vistas de arquitectura del sistema seleccionado serán:

1. Poder responder a los requerimientos de los stakeholders en relación a cómo está estructurado el sistema.
2. Poder identificar los elementos del sistema que deberían modificarse ante un cambio en el sistema. La utilización de la documentación obtenida deberá mejorar la efectividad y/o disminuir los costos que existirían sin contar con ella. Este punto considerará sólo los cambios más probables en el contexto del sistema y las sugerencias de sus stakeholders, por ejemplo:
   2.1. Un aumento en la cantidad de clientes.
   2.2. Fallas en la comunicación entre los nodos del sistema.
   2.3. Integración con otros sistemas que alimenten o necesiten información del sistema.

4.2. Identificación del Sistema a Documentar

El sistema que se documentó es un producto de la empresa Microsoft Corporation llamado System Center Configuration Management 2007 (SCCM 2007). Entre las funcionalidades de este sistema se destacan:

- Recolección de Inventario de Software y Hardware
- Distribución e Instalación de Software
- Control remoto de workstations

Este sistema fue adquirido e instalado en el año 2011 para cubrir las necesidades de administración del parque de workstations que utilizan los empleados del banco.

4.3. Caso de Negocio que Resuelve el Sistema

El Banco en estudio tiene una Oficina Central y 200 Sucursales, y sus sistemas están alojados en un Site Central. La Oficina Central accede a los sistemas a través de un enlace dedicado de 100MB, y las Sucursales a 2MB cada una. La conexión a internet del Site Central tiene un ancho de banda de 20MB.

Las workstations son aproximadamente 10000 y tienen sistema operativo Windows XP Profesional. Esta plataforma debe ser actualizada periódicamente para mantener el soporte entregado por el proveedor del sistema operativo y las aplicaciones que se montan en él. El no contar con soporte puede generar pérdidas a la organización por la indisponibilidad de alguna de las funcionalidades que entregan las aplicaciones que soportan las operaciones del Banco.

Las políticas del Banco impiden que se realicen actualizaciones de sistemas sin un proceso de validación previa. Esto se realiza con el objetivo de prevenir problemas operacionales producto de incompatibilidades, que puedan existir entre la actualización y el software pre-existente en las estaciones.

Las workstations están configuradas en base a perfiles que caracterizan conjuntos de usuarios que requieren realizar funciones similares. Estos perfiles se actualizan de acuerdo a la evolución de las aplicaciones y el negocio. También puede haber cambios en las responsabilidades de cada usuario. Por ejemplo: un ejecutivo que es ascendido a un cargo de Agente de Sucursal puede mantener su workstation pero esta se le debe actualizar con otras aplicaciones de acuerdo a sus nuevas labores. Además, el sistema operativo y las aplicaciones de escritorio deben permanecer actualizados para reducir riesgos de infección por virus informáticos, accesos no autorizados o fallas identificadas con posterioridad a su paso a producción. Estas actualizaciones tienen un impacto directo en la estabilidad de la plataforma de atención a clientes del Banco.

Por otra parte, se debe mantener un control de la configuración de las workstations para prevenir multas producto de la instalación de software no licenciado por el Banco.

También, existe una necesidad de acogerse a distintas regulaciones de cumplimiento de la Superintendencia de Valores y Seguros (SVS) como el resguardo de información de los clientes del Banco.

La solución a los problemas descritos anteriormente fue la utilización de una plataforma distribuida y escalable, que permite la actualización y el control remoto de las workstations y las aplicaciones instaladas en el Banco.

4.4. ¿Por qué Documentar la Arquitectura del Sistema Elegido?

Es deseable que la operación del sistema esté acompañada de documentación que facilite la toma de decisiones ante incidentes que puedan afectarlo. La documentación existente en el banco consiste en un conjunto de manuales de usuario y de administración demasiado extensa para cumplir con dicho deseo, por ello existe la oportunidad de mejorarla a través de la documentación de su arquitectura. La información para realizar esta documentación se obtendrá de los documentos existentes en el Banco, en el sitio del proveedor, y de entrevistas a sus stakeholders.

4.5. Documentación

4.5.1. Identificación de los Stakeholders

Para identificar los stakeholders, se ubicaron los actores del proceso que apoya el sistema y que la documentación de su arquitectura tiene un impacto sobre la toma de sus decisiones. En este caso, el proceso es la administración de la configuración de workstations del Banco. En este contexto, se identificaron los siguientes actores directos:

- **Microsoft:** Compañía que desarrolló, distribuye y entrega el soporte de software ante fallas.
- **Datco**: Compañía que entrega servicios de consultoría e instalación del software.
- **Departamento de Plataforma Distribuida del Banco:** se encarga de la operación, mantención y soporte a usuarios.
- **Departamento de Redes de Comunicación:** se encarga de la disponibilidad de los canales de comunicación de datos.
- **Departamento de Seguridad Informática del Banco:** se encarga de habilitar los puntos de acceso a través de las redes del Banco.

Como se trata de dar soporte a los procesos internos en el Banco, se considerarán sólo los stakeholders que pertenecen a él. Por lo cual, la lista se reduce a:

- Departamento de Plataforma Distribuida del Banco
- Departamento de Redes de Comunicación
- Departamento de Seguridad Informática del Banco

Análisis de la importancia de los involucrados

Como en todo proyecto puede haber riesgos que deben ser controlados para asegurar una implementación exitosa. En el caso de la documentación de una arquitectura, podrían existir detractores que pongan en riesgo el éxito de la iniciativa de documentación. Para evaluar el riesgo de que un stakeholder no esté de acuerdo con la realización del proyecto, se definieron tres niveles: Alto, Medio y Bajo, donde Alto implica que es muy probable que un stakeholder se oponga a la documentación y Bajo implica que estará a favor de la documentación. También, para identificar el grado de influencia que puede tener para afectar negativamente el proyecto, se definió tres niveles: Alto, Medio y Bajo, donde Alto implica que el stakeholder puede afectar gravemente el éxito del

proyecto y Bajo implica que su influencia puede ser mínima. Finalmente, para evaluar la importancia de la participación de cada stakeholder, se definieron los niveles de importancia: Alto, Medio y Bajo, donde Alto significa que el apoyo del stakeholder es muy importante para el proyecto, y Bajo significa que es posible realizar el proyecto sin un impacto significativo.

Stakeholder	Riesgo	Influencia	Importancia
Depto. de Plataforma Distribuida	Bajo	Alto	Alta
Depto. de Redes de Comunicación	Bajo	Alto	Media
Depto. de Seguridad Informática	Bajo	Alto	Media

Tabla IV-1 Análisis de Importancia de los Involucrados

Después del análisis del riesgo de que algún stakeholders pueda afectar la realización del proyecto, de la Tabla IV-1 se concluyó que no existen riesgos relevantes y que es muy importante la participación del departamento de plataforma distribuida, ya que sus integrantes son los que regularmente utilizan el sistema. Por otra parte, los departamentos de redes de comunicación y seguridad informática tienen una importancia media por ser unidades de apoyo a la continuidad operativa del sistema. De existir riesgos relevantes, sería necesario buscar mitigantes como el intentar convencer a los detractores sobre la utilidad de la documentación de la arquitectura y motivarlos a participar en él.

Categorización de los Involucrados

La categorización de los stakeholders involucrados permitirá conocer sus intereses y, con ello, poder dirigir las entrevistas.

Stakeholder	Intereses	Rol
Depto. de Plataforma Distribuida	• Contar con una visión de alto nivel que permita dar respuesta a solicitudes de clientes internos	Facilitador
Depto. de Redes de Comunicación	• Contar con una visión de alto nivel que permita dimensionar y optimizar el nivel de uso de los canales de comunicación	Interesado
Depto. de Seguridad Informática	• Estar informado sobre la seguridad de uso del sistema y sus datos • Conocer los puntos críticos donde es importante la seguridad de las redes de comunicación	Regulador, Interesado

Tabla IV-2 Categorización de los Involucrados

La categorización mostrada por la Tabla IV-2 servirá como guía para dibujar los atributos que le interesan a los stakeholders. Por ejemplo: si el sistema falla al intentar conectarse a una workstation, el departamento de plataforma distribuida podrá indicarle a los técnicos dónde están los flujos de datos y cuáles son los elementos que participan de la funcionalidad.

Utilidad de los Atributos de Calidad para los Stakeholders

Según Clements, et al en [1], las vistas de arquitectura útiles para los stakeholders están relacionadas a los atributos de calidad que interesa a cada uno. Como resultado de las entrevistas

realizadas a los stakeholders, la Tabla IV-3 muestra la relación entre los stakeholders y los atributos de calidad que están relacionados directamente con la arquitectura del sistema que fueron identificados por Stephen T. Albin [7] y Losavio et al. [13].

Stakeholder	Funcionalidad	Confiabilidad	Eficiencia	Mantenibilidad	Portabilidad
Depto. de Plataforma Distribuida	*	*			
Depto. de Redes de Comunicación			*		
Depto. de Seguridad Informática	*				

Tabla IV-3 Stakeholders y sus Atributos de Calidad

De la identificación de los atributos de calidad mostrada en la Tabla IV-3, se concluye que los atributos más importantes para los stakeholders son: Funcionalidad, Confiabilidad y Eficiencia. Al analizar la Tabla III-6, mostrada en el capítulo anterior, se deduce que estos atributos de calidad son apoyados por los estilos de vistas de asignación. Los stakeholders no consideraron que la mantenibilidad fuera importante, ya que no visualizaron ningún requerimiento que implicara una modificación del sistema. Asimismo, no es necesario que el sistema sea portable, ya que sus funcionalidades están relacionadas únicamente a workstations de la familia de sistemas operativos Microsoft Windows.

4.5.2. Diagrama de Contexto

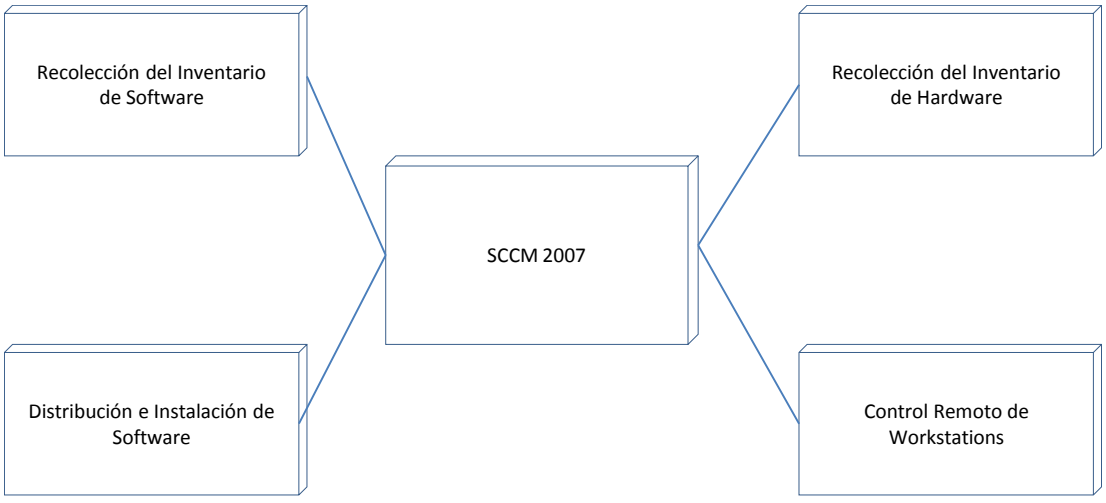


Figura IV-2 Diagrama de Contexto del Sistema

El sistema en estudio (SCCM 2007), es utilizado en el Banco para las funcionalidades mostradas por el diagrama de contexto mostrado en la Figura IV-2, donde:

- **Recolección del inventario de software:** permite obtener información de las características del software instalado en cada workstation conectada a la red del Banco.
- **Recolección del inventario de hardware:** permite obtener información de las características de los dispositivos de cada workstation conectada a la red del Banco.
- **Distribución e instalación de software:** permite la distribución de actualizaciones de software existente en los equipos y, además, la instalación de paquetes de software.
- **Control remoto de workstations:** permite tomar el control de las workstations conectadas a la red para realizar labores de soporte y configuración de los mismos.

Todas estas funcionalidades están orientadas a equipos con sistema operativo Windows.

4.5.3. Presentación Primaria

En esta sección se identificarán las vistas de arquitectura que se utilizarán para mostrar cómo se estructura el sistema, y que son consideras útiles de acuerdo a los atributos de calidad que buscan los stakeholders.
De acuerdo a la Tabla III-6 que identifica la relación entre los estilos del tipo de vista de asignación, para representar la funcionalidad, confiabilidad y la eficiencia se deben utilizar vistas de despliegue.
Las vistas de despliegue elegidas serán las siguientes:

- Vista de Nodos que Componen el Sistema: identificará los nodos, sus relaciones principales y el ancho de banda disponible para el intercambio de datos, requeridos por el Departamento de Redes de Comunicación.
- Vistas de Funcionalidades y Protocolos de Comunicación: identificarán las funcionalidades y los protocolos de comunicación utilizados y el ancho de banda requerido.

En cada vista se identificarán las propiedades según las necesidades que los stakeholders identifican como útiles para sus intereses. Para representar cada vista, se utilizará UML 2.0 con los estereotipos estándar del lenguaje.

Nodos del Sistema
La Figura IV-3 muestra una vista de despliegue del sistema, en la cual, se identifican los nodos, su ubicación geográfica, dónde están los flujos de datos principales, y qué ancho de banda de red existe entre las oficinas físicas del Banco. En ella, se observa que existe un Primary Server, que corresponde al servidor central del sistema, que obtiene actualizaciones de los servidores de Microsoft Windows Update, se comunica con el Domain Controller, un servidor de bases de datos llamado Site Database Server, utiliza 6 Distribution Points, que son los encargados de la distribución de paquetes de software y actualizaciones. Además, para reducir la cantidad de transmisiones de paquetes y actualizaciones de software a oficinas que están distribuidas a lo largo del país, cada sucursal tiene una workstation llamada Branch Distribution Point, que funciona como punto de distribución dentro de ellas.

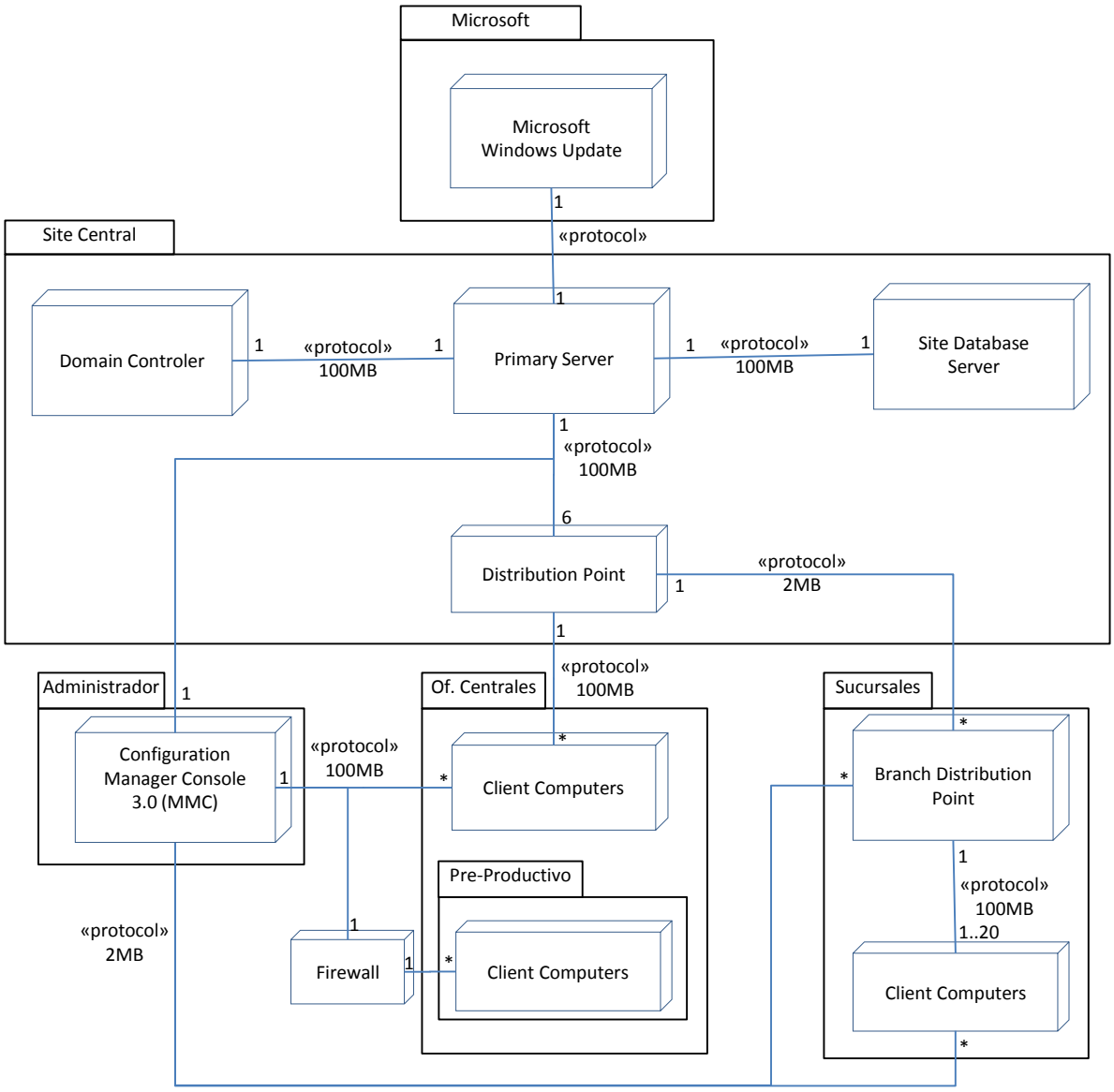


Figura IV-3 Vista de Despliegue de los Nodos que componen el Sistema

Recolección del inventario de hardware

El tener un parque de workstations (estaciones de trabajo) actualizadas, que agilicen las labores cotidianas de los trabajadores, se apoya en un inventario actualizado que permite priorizar la actualización o remplazo de los equipos más antiguos o de menor capacidad. Sin SCCM, el proceso sería realizado manualmente, elevando los costos de administración ya que las workstations cambian constantemente.

El proceso de recolección del inventario de hardware de las workstations hace uso de un componente que se incluye en los sistemas operativos Microsoft Windows llamado WMI (Windows Management Instrumentation), permitiendo acceder y compartir datos detallados de dispositivos y programas a través de la red.

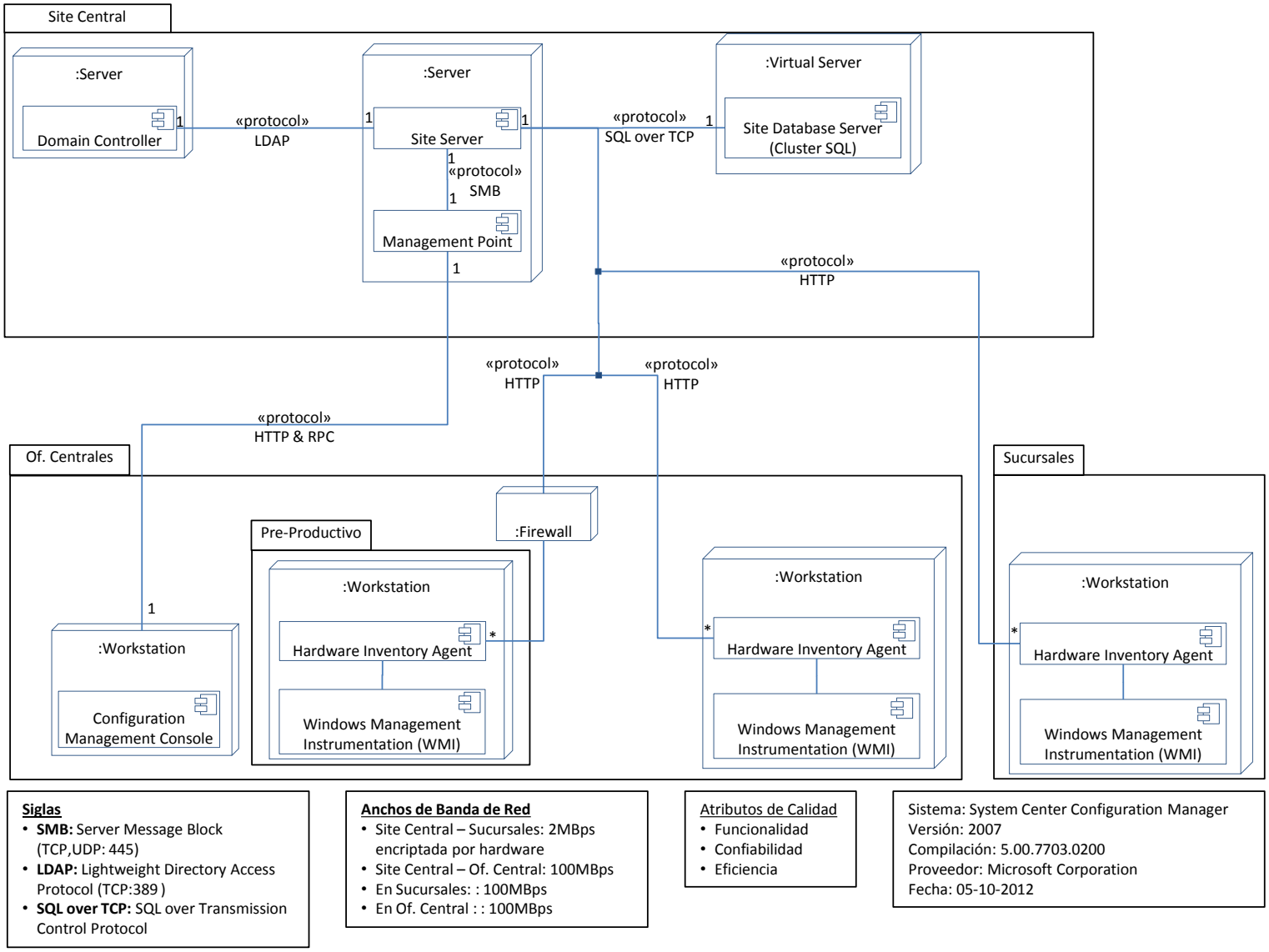


Figura IV-4 Vista de despliegue de la recolección de inventario de hardware

En la Figura IV-4, para realizar la recolección del inventario de hardware, el sistema ejecuta los siguientes pasos:

1. A través del **Configuration Management Console,** el **Administrador** solicita habilitar el inventario de hardware en clientes en el **Site Server**, indicándose cuándo inicia y con qué frecuencia debe realizarse. El **Site Server** verifica que el usuario **Administrador** tenga permisos para habilitar el inventario de hardware y registra el cambio en el **Site Database Server**. Esta configuración crea una **Política de Máquinas** para indicar al **Hardware Inventory Agent** que realice un inventario
2. La **Política de Máquinas** se copia al **Management Point** (SMB – 3Kbytes).
3. Las **Workstations** rescatan la **Política de Máquinas** desde el **Management Point** (SMB – 3Kbytes).
4. **WMI** recolecta información del hardware de las **Workstations** y se envía al **Management Point** un archivo llamado InventoryAgent.log.
5. Los reportes de cada **Workstation** (aprox. 200KB) son enviados por el **Hardware Inventory Agent** al **Management Point** y este se los entrega al **Site Server** para ser registrados en el **Site Database Server**.

Desde el punto de vista de los departamentos de Seguridad y de Plataforma Distribuida, esta vista cumple con requerimientos de Funcionalidad, ya que el sistema entrega mecanismos de control de acceso para habilitar las capacidades de inventario de hardware que le permiten al departamento de Plataforma Distribuida coordinar sus esfuerzos para mantener actualizada la infraestructura de hardware del Banco.
Desde el punto de vista del departamento de Comunicaciones, la vista permite identificar que los flujos de datos corresponden sólo al detalle del inventario de hardware de cada workstation. Por tratarse de un sistema distribuido y delegar la coordinación de la obtención de los datos a los Hardware Inventory Agents, se reduce el tráfico de red que existiría si se utilizara un modelo de sistema centralizado, donde la coordinación de la obtención de los datos la realizara el Site Server o el Management Point.

Recolección del inventario de software

La recolección del inventario de software, al igual que la de hardware, es una tarea muy trabajosa de realizar y mantener manualmente. Con SCCM 2007 esta tarea se realiza automáticamente, por lo que todo proyecto que dependa de tener el inventario actualizado de la configuración de software de las workstations, es realizado disminuyendo al mínimo el riesgo de incompatibilidades de software por desconocimiento del entorno.
El proceso de recolección del inventario de software de las workstations, hace uso de un componente que se incluye en los sistemas operativos Microsoft Windows llamado WMI (Windows Management Instrumentation), permitiendo el acceso y compartir datos detallados de los dispositivos y programas instalados.

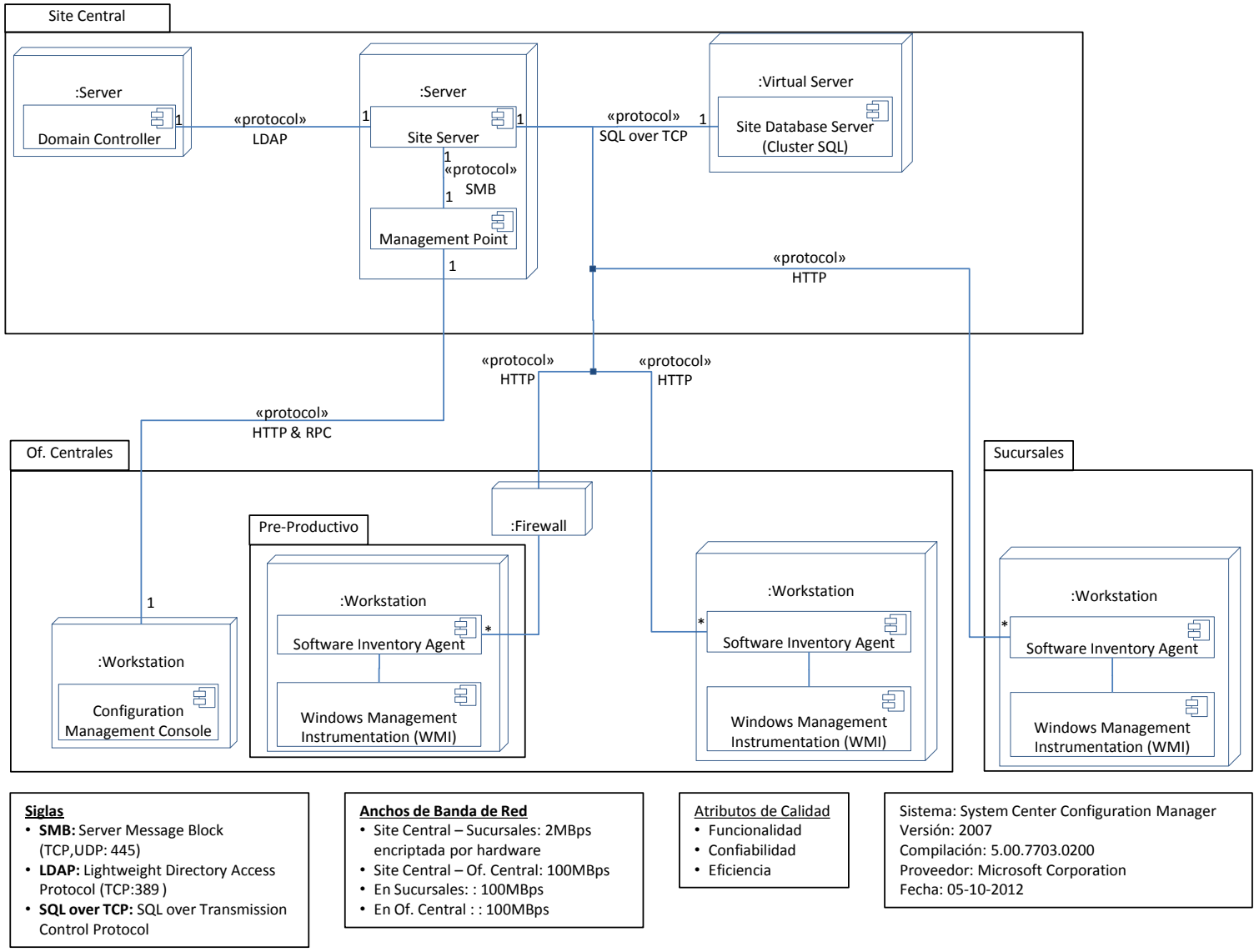


Figura IV-5 Vista de despliegue de la recolección del inventario de software

En la Figura IV-5, para realizar la recolección del inventario de software, el sistema ejecuta los siguientes pasos:

1. A través del **Configuration Management Console,** el Administrador solicita habilitar el inventario de Software en clientes en el **Site Server**, indicándose cuándo inicia y con qué frecuencia debe realizarse. El **Site Server** verifica que el usuario Administrador tenga permisos para habilitar el inventario de Software y registra el cambio en el **Site Database Server**. Esta configuración crea una **Política de Máquinas** para indicar al **Software Inventory Agent** que realice un inventario
2. La **Política de Máquinas** se copia al **Management Point (SMB – 3Kbytes)**.
3. Las **Workstations** rescatan la **Política de Máquinas** desde el **Management Point (SMB – 3Kbytes)**.
4. **WMI** recolecta información del Software de las **Workstations** y se envía al **Management Point** un archivo llamado InventoryAgent.log.
5. Los reportes de cada Workstation (aprox. 200KB) son enviados por el **Software Inventory Agent** al **Management Point** y este se los entrega al **Site Server** para ser registrado en el **Site Database Server**.

Desde el punto de vista de los departamentos de Seguridad y el de Plataforma Distribuida, esta vista cumple con los requerimientos de Funcionalidad, ya que el sistema entrega mecanismos de control de acceso para habilitar las capacidades de inventario de software, que le permiten al departamento de Plataforma Distribuida coordinar sus esfuerzos para identificar y controlar el uso de licencias de software en el Banco.
Desde el punto de vista del departamento de Comunicaciones, la vista permite identificar que los flujos de datos corresponden sólo al detalle del inventario de software de cada workstation, ya que al ser un sistema distribuido y delegar la coordinación de la obtención de los datos a los Software Inventory Agents se reduce el tráfico de red que existiría si se tratara de un sistema centralizado donde la coordinación se realizara desde el Site Server o el Management Point.

Distribución e instalación de software
Una de las funcionalidades más importantes de SCCM 2007 es la posibilidad de distribuir e instalar paquetes y actualizaciones de software en las workstations. Esto ayuda a mantener un parque de equipos con software actualizado y estandarizado, restringiendo su configuración a una cantidad acotada software licenciado.

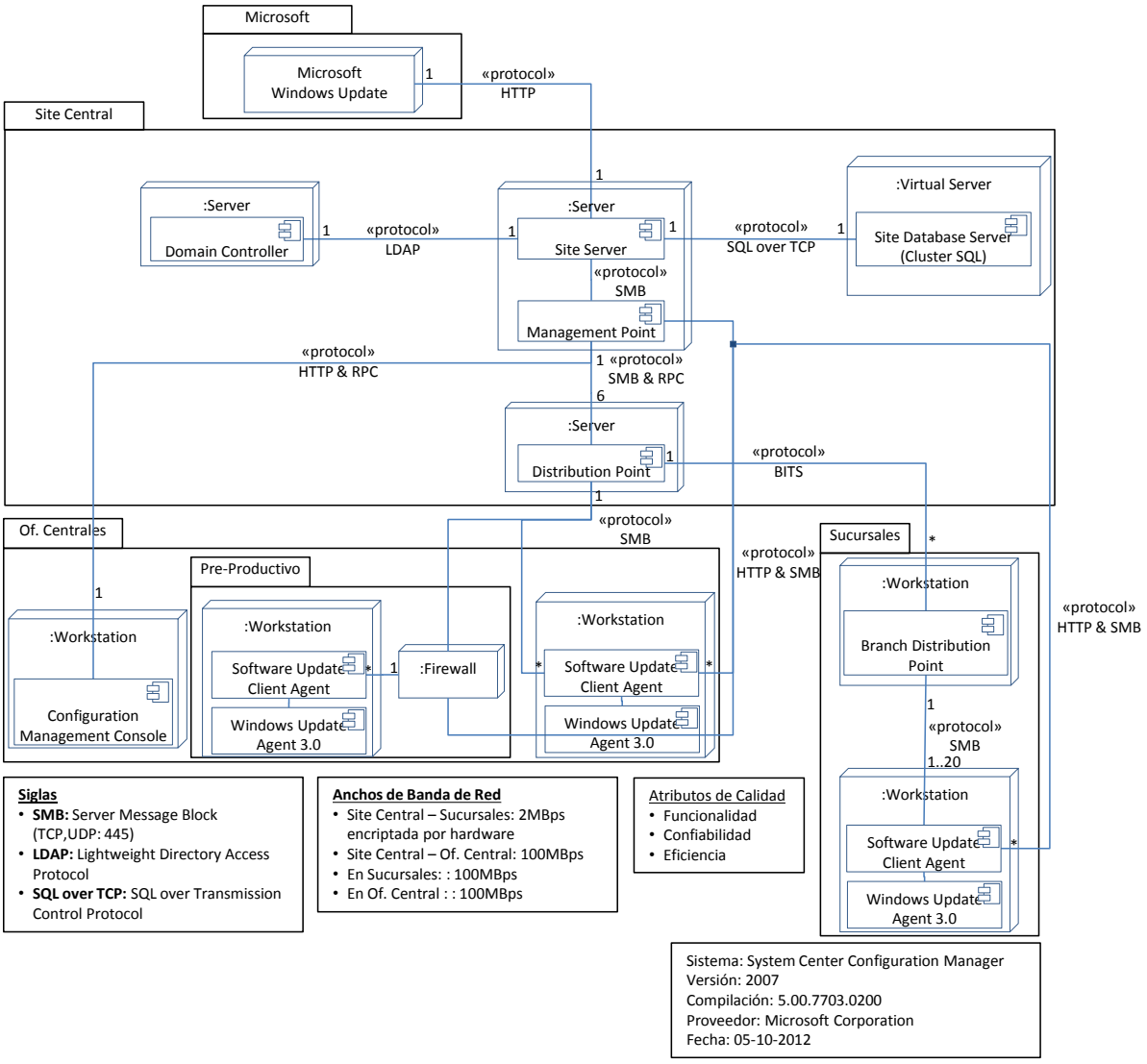


Figura IV-6 Vista de despliegue de la distribución e instalación de software

En la Figura IV-6, para realizar la distribución e instalación de software, el sistema ejecuta los siguientes pasos:

1. A través del **Configuration Management Console,** el Administrador solicita crear un nuevo **Paquete de Instalación** en el **Site Server** para ser instalado en las **Workstations**. El **Site Server** verifica que el usuario Administrador tenga permisos para crear el Paquete de Instalación y registra el cambio en el **Site Database Server**.
2. El **Site Server** solicita los binarios desde donde especifica el **Paquete de Instalación**. Los binarios pueden ser obtenidos localmente o desde el sitio de **Microsoft Windows Update**.
3. El **Site Server** copia los binarios a los **Distribution Point (SMB)**. Además, agrega una **Política de Máquinas** y la copia al **Management Point (SMB – 3Kbytes)**.
4. Los **Branch Distribution Point** rescatan los binarios desde los **Distribution Point (BITS).**

5. Las **Workstations** rescatan la **Política de Máquinas** desde el **Management Point (SMB – 3Kbytes)**, obteniendo la información del nuevo **Paquete de Instalación** y verifican que es necesario para su configuración actual.
6. Si el nuevo **Paquete de Instalación** es necesario, los **Software Update Client Agents** solicitan los binarios al **Distribution Point** o **Branch Distribution Point (SMB),** asignado mediante un algoritmo que depende del segmento de red al que pertenece la workstation, y lo almacenan en su cache local, indicándole al **Windows Update Agent** que realice la instalación.
7. Los **Software Update Client Agents** indican al **Management Point** cuándo el **Paquete de Instalación** fue descargado y éste lo indica al **Site Server** para que lo registre en el **Site Database Server**.

Desde el punto de vista de los departamentos de Seguridad y de Plataforma Distribuida, esta vista cumple con los requerimientos de Funcionalidad, ya que el sistema entrega mecanismos de control de acceso para habilitar la funcionalidad de instalación y actualización de software.
Desde el punto de vista del departamento de Comunicaciones, la vista permite identificar que los flujos de paquetes de instalación están optimizados tomando en cuenta la distribución geográfica de las sucursales del Banco y sus limitaciones de ancho de banda al incluir los Branch Distribution Point que permiten que cada paquete que debe ser instalado en los equipos de una sucursal sea transmitido una única vez.

Control remoto de workstations
La funcionalidad de control remoto permite un control descentralizado de las workstations al personal especializado encargado de soporte, utilizando dos posibilidades que entrega la familia de sistemas operativos Windows:

- Control remoto a través de Remote Desktop
- Control remoto a través de Remote Assistance

La diferencia entre ambas funcionalidades es que con Remote Desktop se crea una sesión de usuario independiente a la que puede existir cuando se utiliza físicamente la workstation, en cambio, con Remote Assistance se toma el control de la sesión actual que tiene el equipo, por lo que el usuario actual puede ver las acciones que se están realizando sobre él.

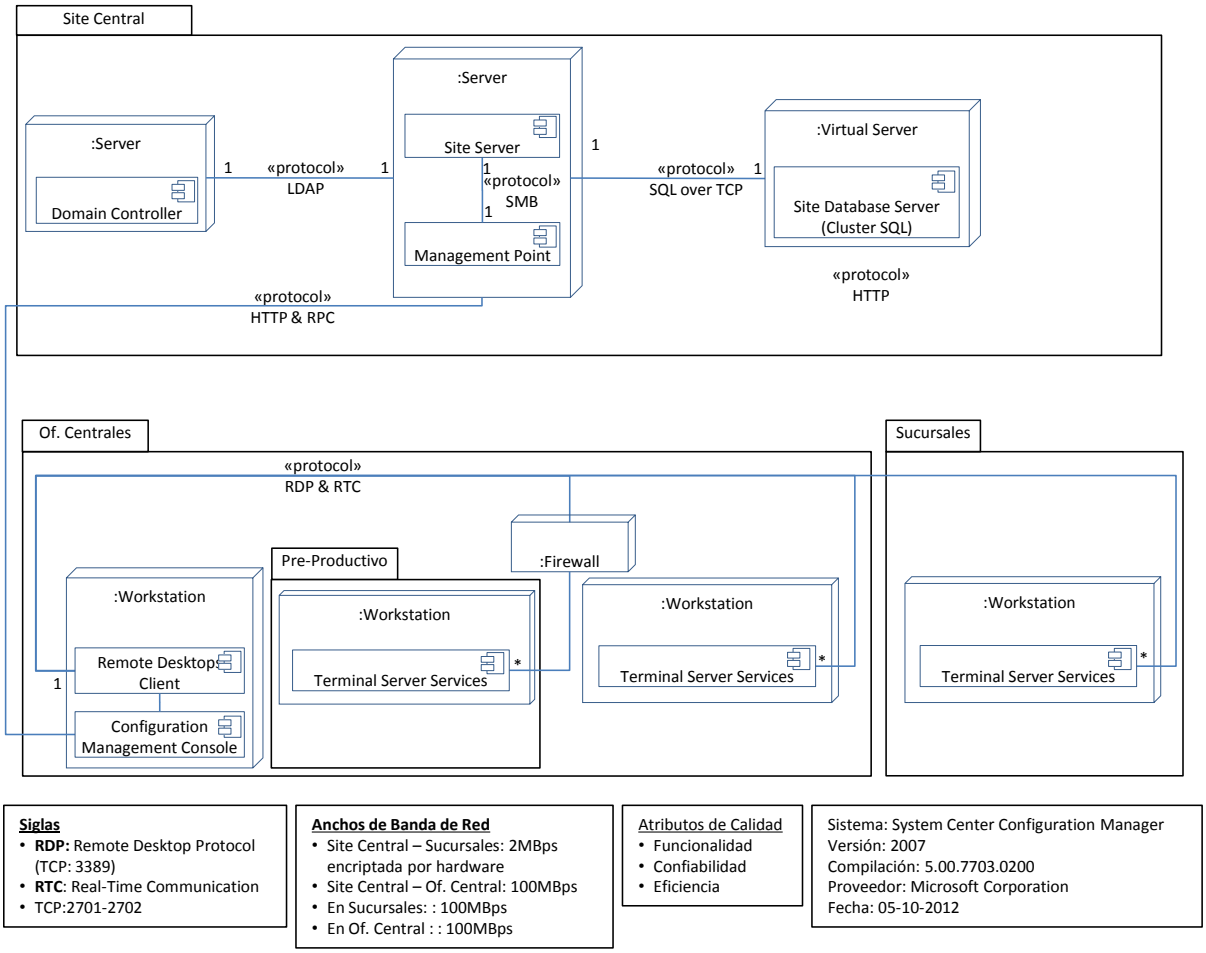


Figura IV-7 Vista de despliegue del control remoto de workstations – remote desktop

En la Figura IV-7, para controlar remotamente las workstations utilizando remote desktop, el sistema realiza los siguientes pasos:

1. El **Configuration Management Console** obtiene los alias de las Workstation que pueden ser controladas remotamente y su operador selecciona la que va a controlar.
2. A través del **Configuration Management Console,** se llama a **Remote Desktop Client** de una **Workstation** seleccionada, y se ingresa el nombre de usuario y la contraseña para acceder a ella.

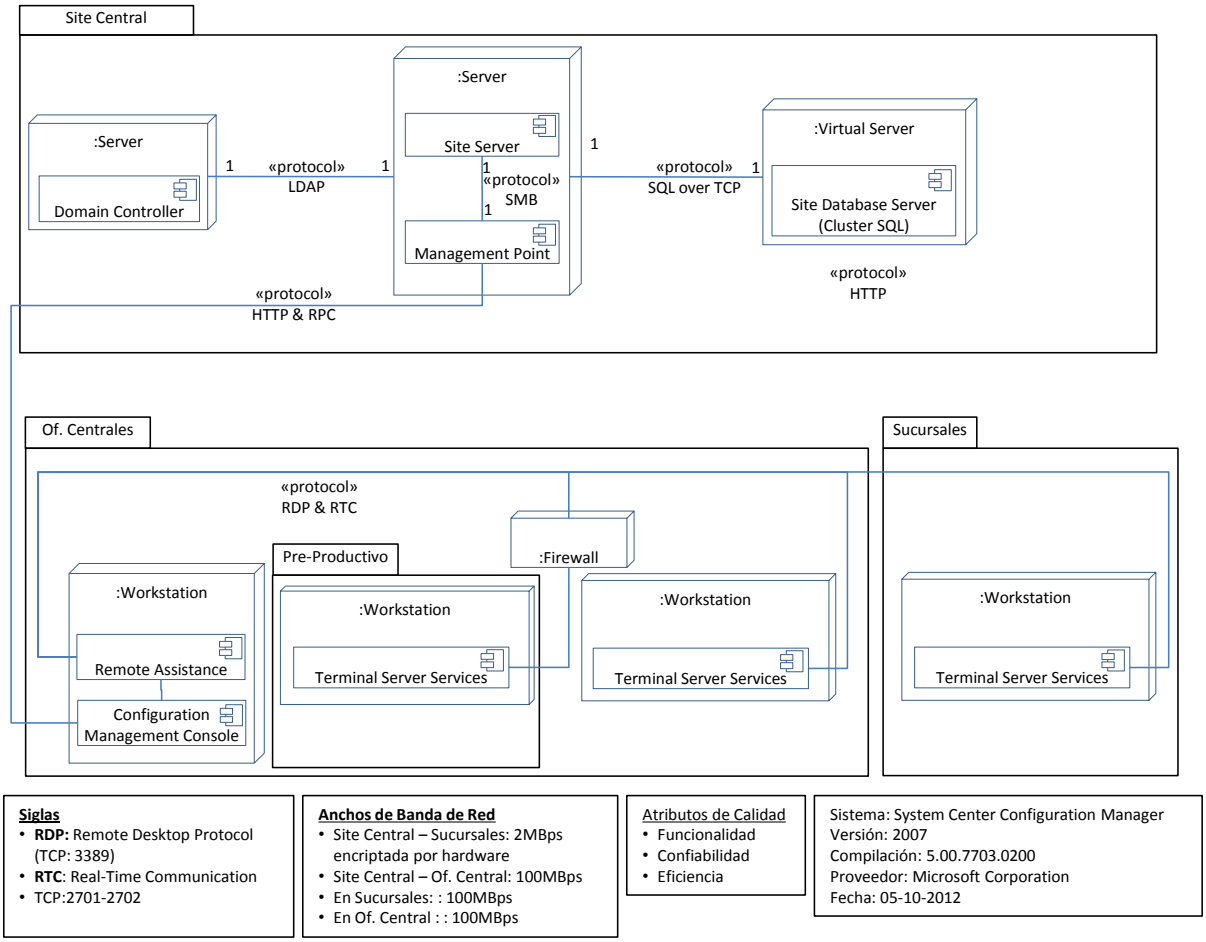


Figura IV-8 Vista de despliegue del control remoto de workstations – remote assistance

En la Figura IV-8, para controlar remotamente las workstations utilizando remote assistance, el sistema realiza los siguientes pasos:

1. El **Configuration Management Console** obtiene los alias de las Workstation que pueden ser controladas remotamente y su operador selecciona la que va a controlar
2. A través del **Configuration Management Console,** se llama a **Remote Assistance** para que el usuario actual acepte la conexión y el operador pueda tomar el control de la máquina.

Desde el punto de vista de los departamentos de Seguridad y el de Plataforma Distribuida, esta vista cumple con los requerimientos de Funcionalidad, ya que el sistema entrega mecanismos de control de acceso para habilitar la funcionalidad de control remoto de workstations. Además, en el caso de Remote Assistance, permite que los usuarios tengan la posibilidad de rechazar accesos no autorizados.

4.5.4. Catálogo de Elementos

En las Tablas IV-4 y IV-5 se describen los elementos que se muestran en las vistas de la arquitectura del sistema. Este catálogo sirve de guía para identificar las funciones de cada componente.

Elemento	Descripción
Branch Distribution Point	Está pensado para atender pequeños sitios u oficinas distribuidas que necesiten un punto de distribución y que no posean un servidor dedicado a tal propósito. Un Branch Distribution Point puede ser instalado en un equipo cliente con sistema operativo Windows XP o superior.
Configuration Manager Console	Consola de Administración del Sistema destinada a trabajos de administración.
Distribution Point	Es un rol que puede estar en el Primary Server o un servidor que no tenga SCCM instalado. Puede utilizar IIS para comunicarse con los agentes y es un repositorio del software que será distribuido a los clientes.
Domain Controller	Servidor que contiene el Active Directory donde están registrados todos los usuarios del dominio Windows, y sirve de apoyo a la administración de usuarios del sistema.
Hardware Inventory Client Agent	Es un proceso que recolecta información relacionada al hardware del cliente en que se ejecuta. La información recolectada puede incluir tipo de procesador, tarjeta de red, tamaño de la memoria, e información de discos.
Management Point	Es el punto de intercambio entre los Clientes de SCCM y el Site Server. Este provee mecanismos de balanceo de carga que son útiles cuando se requiere aumentar la cantidad de clientes.
Microsoft Windows Update	Sito de Microsoft Corporation destinado a la distribución de actualizaciones de software para los productos de software de esta compañía.
Remote Assistance	Componente que permite a un usuario tomar el control de un equipo remoto con sistema operativo Windows que utiliza los servicios de Terminal Services.
Remote Desktops Client	Componente que permite a un usuario ingresar a un equipo remoto con sistema operativo Windows utilizando los servicios de Terminal Services.
Remote Tools Client Agent	Permite el Control Remoto de Computadores y la integración con Asistencia Remota.
Site Database Server	Servidor que contiene la base de datos del sistema, que guarda configuraciones e información de inventario de hardware y software. Adicionalmente, contiene la base de datos del rol Software Update Point, que mantiene registro de las actualizaciones y del estado de ellas.

Tabla IV-4 Catálogo de Elementos del Sistema

Elemento	Descripción
Site Server	Es el servidor principal del sistema SCCM, que accede a los datos de las workstations y del SW a distribuir. Todos los clientes Windows están registrados en este sitio y se deben comunicar directa o indirectamente con él.
Software Inventory Client Agent	Es un componente que recolecta información relacionada al software del cliente en que se ejecuta. La información recolectada puede incluir datos sobre el sistema operativo, programas instalados, y cualquier archivo que se quiera inventariar o recolectar.
Software Update Client Agent	Es un componente que se encarga del análisis y la evaluación del cumplimiento de solicitudes, evaluación de solicitudes de actualización de software, las políticas de implementación en clientes y las solicitudes de descarga de contenido.
Terminal Server Services	Es el componente que permite a un usuario acceder a las aplicaciones y datos de un equipo remoto.
Window Management Instrumentation	Es un componente de la familia de sistemas operativos Windows, que permite la administración de información de sistemas, aplicaciones, redes, dispositivos y servicios. Además, define mecanismos para automatizar tareas administrativas.
Windows Update Agent 3.0	Es un componente que permite la actualización de paquetes de software y actualizaciones en equipos con sistema operativo Windows.
Workstation	PCs registrados en el dominio del Banco que son controlados por el sistema.

Tabla IV-5 Catálogo de Elementos del Sistema (continuación)

4.5.5. Antecedentes de la Arquitectura

La arquitectura representada anteriormente hace uso principalmente de componentes disponibles en la familia de sistemas operativos Microsoft Windows, coordinándolos de tal manera de tener un ambiente centralizado que controla la totalidad de los equipos (servidores y workstations).
Según el proveedor, esta arquitectura está dimensionada para la administración de hasta 25.000 equipos, teniendo un uso eficiente de los recursos de red disponibles al utilizar puntos de distribución, reduciendo la cantidad de transferencias de información entre el servidor central y los equipos que están en sucursales.

4.5.6. Información Adicional

Como una medida para resguardar la integridad de los datos, el Departamento de Administración de Bases de Datos solicitó que los archivos de datos sean almacenados en un storage que entrega mecanismos de redundancia y facilidad de respaldo. Asimismo, para instalar el servidor de la base de datos del sistema, se utiliza una configuración de clúster, lo cual entrega redundancia ante fallas del motor de base de datos.

4.5.7. Paquetes de Vistas Relacionados

Un paquete de vistas es un conjunto de vistas simplificadas, que agrupadas permiten una mejor comprensión de la que se tendría con sólo una vista con una gran cantidad de elementos. Para el caso de estudio particular, se podría decir que la Recolección del Inventario de hardware y el de software pertenecen al mismo paquete.

CAPÍTULO V.
VALIDACIÓN DE LA UTILIDAD DE LA DOCUMENTACIÓN

Las siguientes son situaciones comentadas con los stakeholders en las cuales la arquitectura podría resultar útil:

1. Aumento en la cantidad de workstations.
2. Fallas en la instalación o actualización de software.
3. Problemas para tomar el control remoto de workstations.

5.1. Aumento de la Cantidad de Workstations

Un aumento significativo en la cantidad de workstations que el sistema debería atender, se produciría si el Banco se fusiona con otra entidad.
A continuación se analizará el impacto de este escenario en cada una de las funcionalidades documentadas, suponiendo que se aumenta la cantidad de workstation desde 10.000 (cantidad actual) a 15.000:

- Para las funcionalidades de recolección de inventario de hardware y software, con sus diagramas mostrados en las figuras de las páginas 32 y 34, el agregar nuevas workstations implica analizar si la capacidad actual de almacenamiento del Site Server y el Site Database Server son suficientes para soportar las nuevas workstations, sin que esto implique un cambio en la arquitectura del sistema. Como los reportes de inventario de hardware y software de cada workstation tienen un tamaño aproximado de 200KB, si la infraestructura actual no fuese suficiente, se debe solicitar un aumento de por lo menos 1GB (200KB x 5000Workstations) para el Site Server y el doble[1] (2GB) en el Site Database Server para almacenar cada ciclo de recolección de inventario. Por lo tanto, si requieren almacenar la historia de 36 ciclos de inventario, equivalentes a un inventario por mes durante tres años, se necesitaría aproximadamente 36GB adicionales en el Site Server y 64GB en el Site Database Server.
- Para la funcionalidad de control remoto de workstations no hay impacto ya que las conexiones se realizan punto a punto.
- Para la funcionalidad de distribución e instalación de software, la arquitectura del sistema permite identificar que actualmente existen 6 Distribution Points, encargados de atender el parque actual de servidores. Esta funcionalidad, representada en la Figura V-1, permite determinar que, ante un aumento en la cantidad de workstations, sería recomendable analizar un aumento en la cantidad de Distribution Points para reducir el impacto en los tiempos de respuesta del sistema. Para el supuesto de agregar 5000 workstations a la

[1] Como práctica los administradores de Bases de Datos solicitan el doble de espacio para almacenar la misma cantidad de información que si utilizaran archivos.

solución existente, y considerando que la escalabilidad de los Distribution Points es lineal, se necesitaría agregar 3 Distribution Points para soportar el crecimiento.

En cada funcionalidad analizada, se hace necesario revisar si la infraestructura de red existente es suficiente para absorber la nueva demanda. Por lo tanto, las vistas presentadas ayudarán a los equipos de fusión a comprender cómo se integrarían las nuevas workstations.
El no contar con la documentación de arquitectura presentada, los equipos de fusión contratarían los servicios de una empresa externa que realice el análisis de la solución existente, para proponer las modificaciones propuestas. **Esto implica que la documentación presentada en este trabajo evitaría los costos que podrían existir antes de su desarrollo.**

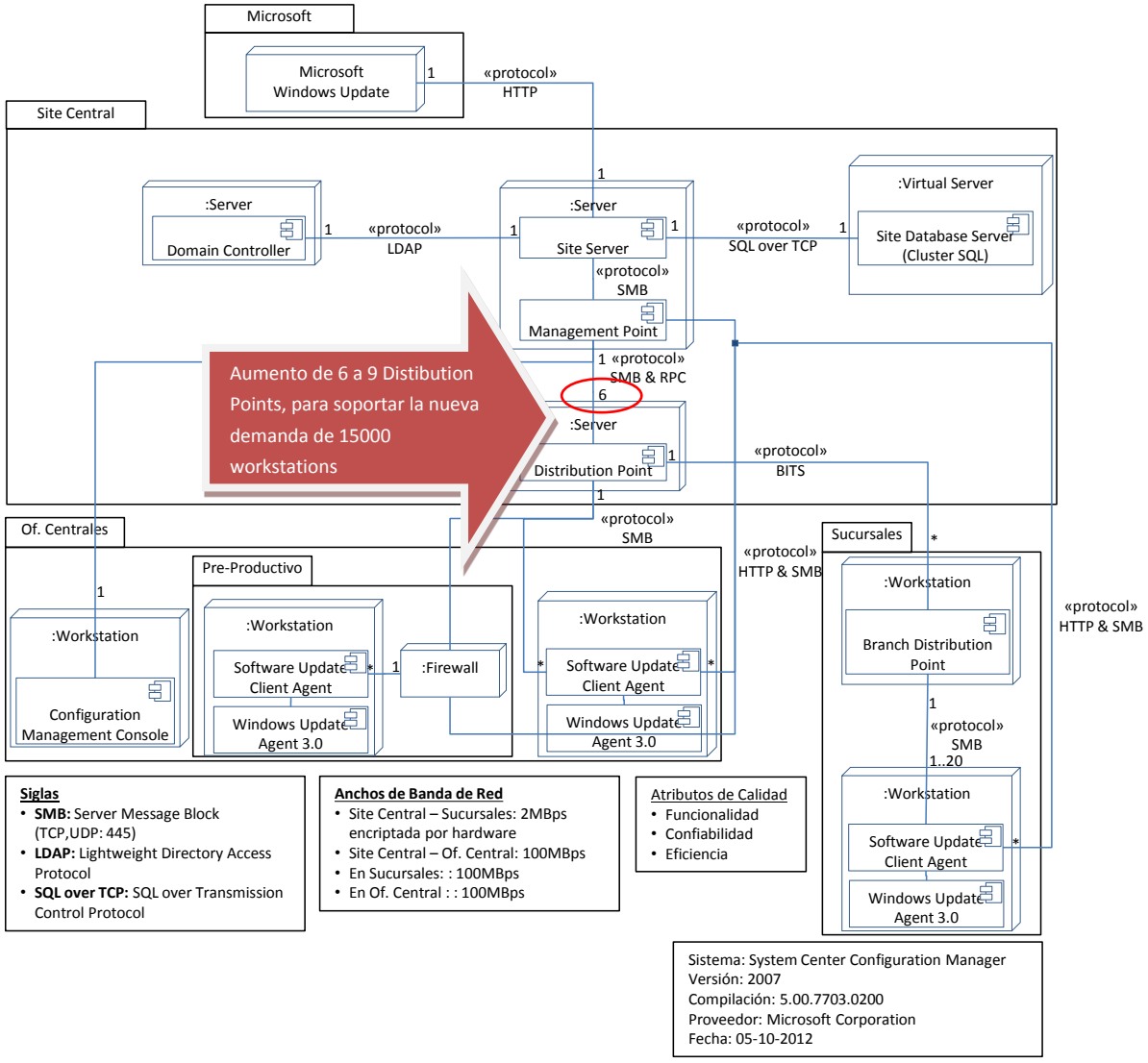


Figura V-1 Identificación de modificaciones ante un aumento en la cantidad de workstations

5.2. Errores en la Distribución e Instalación de Software

Si existen problemas al actualizar las workstations, el analizar la Figura V-1 permite visualizar los flujos de datos existentes entre el Site Server y las Workstations. En este caso, el diagrama sirve para guiar a los equipos técnicos en el análisis de las posibles fallas. Por ejemplo: si el error se produce en las workstations ubicadas en el sector pre-productivo un punto a descartar sería que el firewall esté configurado para permitir el paso de los mensajes con protocolo SMB entre el Site Server y las Workstations. Si el error se produce en alguna sucursal, sería recomendable analizar el estado del Branch Distribution Point de la sucursal, partiendo por verificar si está habilitado el servicio BITS en las estaciones de trabajo afectadas como se muestra en la Figura V-2, y, también, la conectividad con el Site Server.

El no contar con la funcionalidad, llevaría a contratar un consultor para solucionar el problema, el que para comenzar tendría que investigar cómo está implementado el sistema en producción, tarea que ya está realizada con la documentación presentada. Como contingencia, la distribución e instalación de software se realizaría presencialmente en cada workstation afectada. **Por lo tanto, la documentación presentada reduciría el esfuerzo necesario para enfrentar estos errores.**

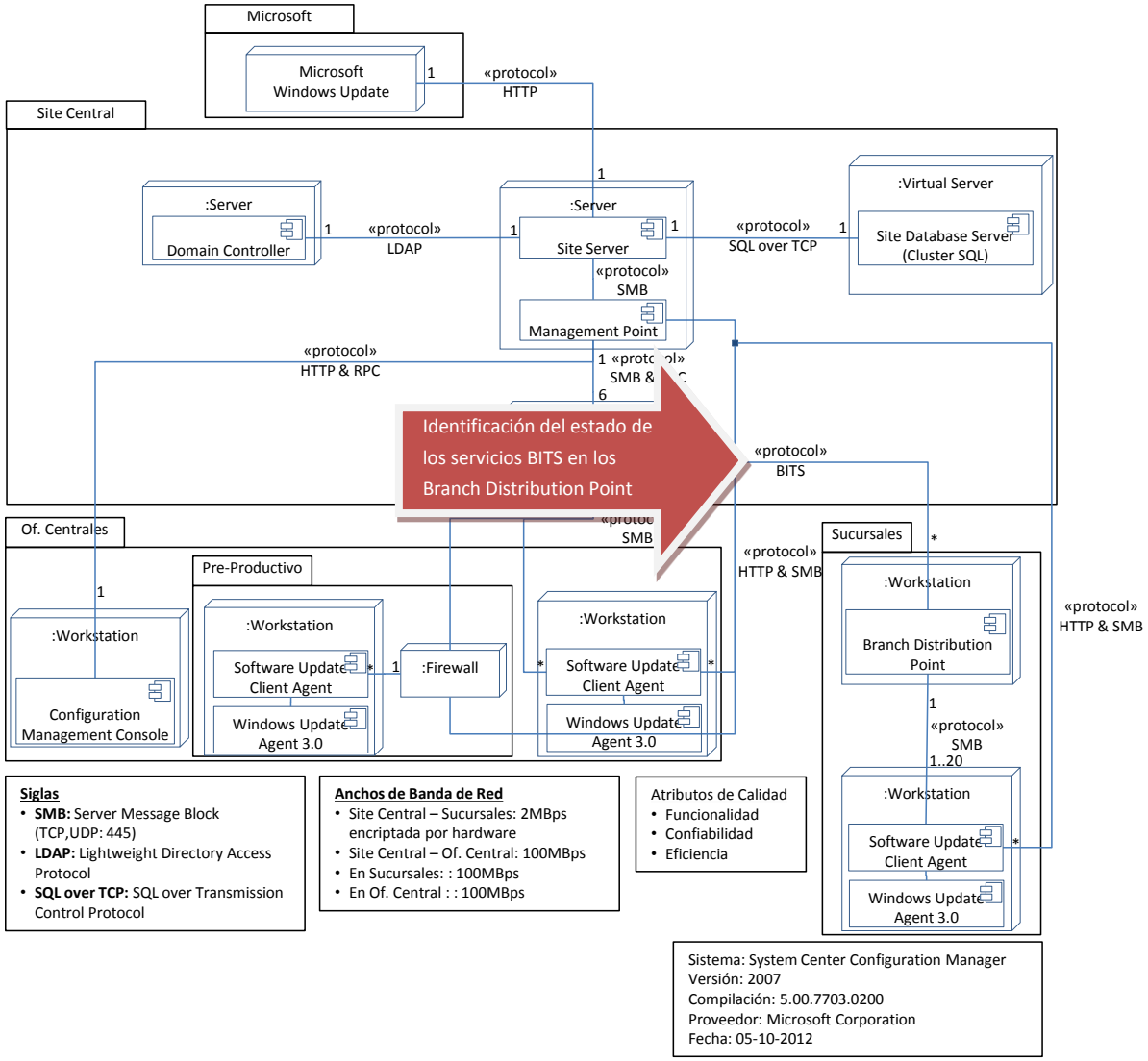


Figura V-2 Identificación de errores en la distribución e instalación de software

5.3. Problemas en el Control Remoto de Workstations

Si no se puede tomar el control de una workstation, al analizar los diagramas en las figuras de las páginas 49 y 50, que representan esta funcionalidad, se puede verificar que si se trata de una workstation en oficina central ubicada en el sector pre-productivo, se debe verificar las reglas del firewall existentes y, además, revisar que estén habilitados los servicios de Terminal Server en la workstation.

De no contar con la documentación de arquitectura presentada, los equipos de soporte tendrían que revisar extensos libros de administración del sistema para analizar cuáles pueden ser las causas del problema. Como contingencia, los equipos de soporte tendrían que realizar presencialmente todas las actividades necesarias en cada workstation. **Por lo tanto, la documentación presentada ayuda a reducir los costos de enfrentar este tipo de problemas disminuyendo los tiempos de solución.**

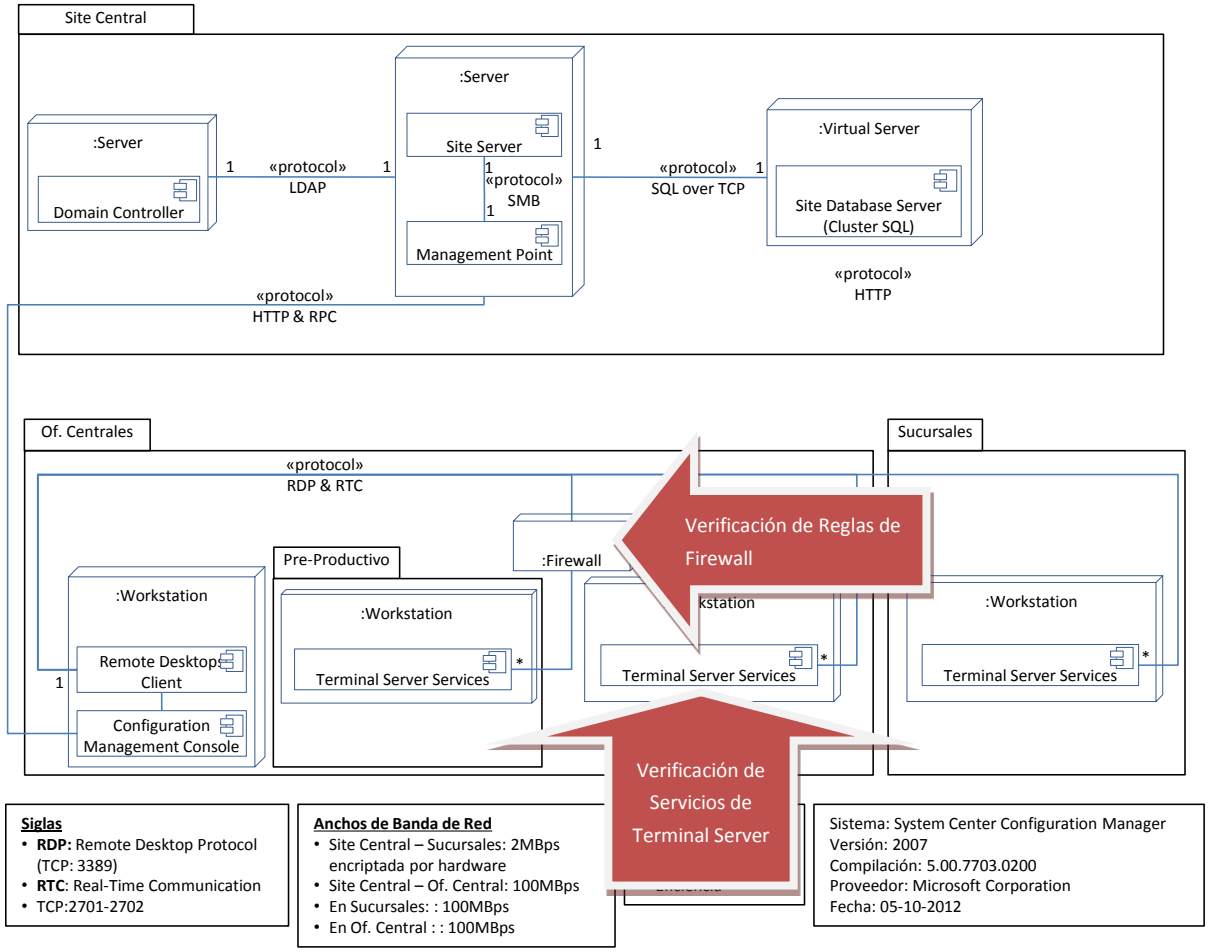


Figura V-3 Algunos puntos posibles de falla en el control remoto de workstation – remote desktop

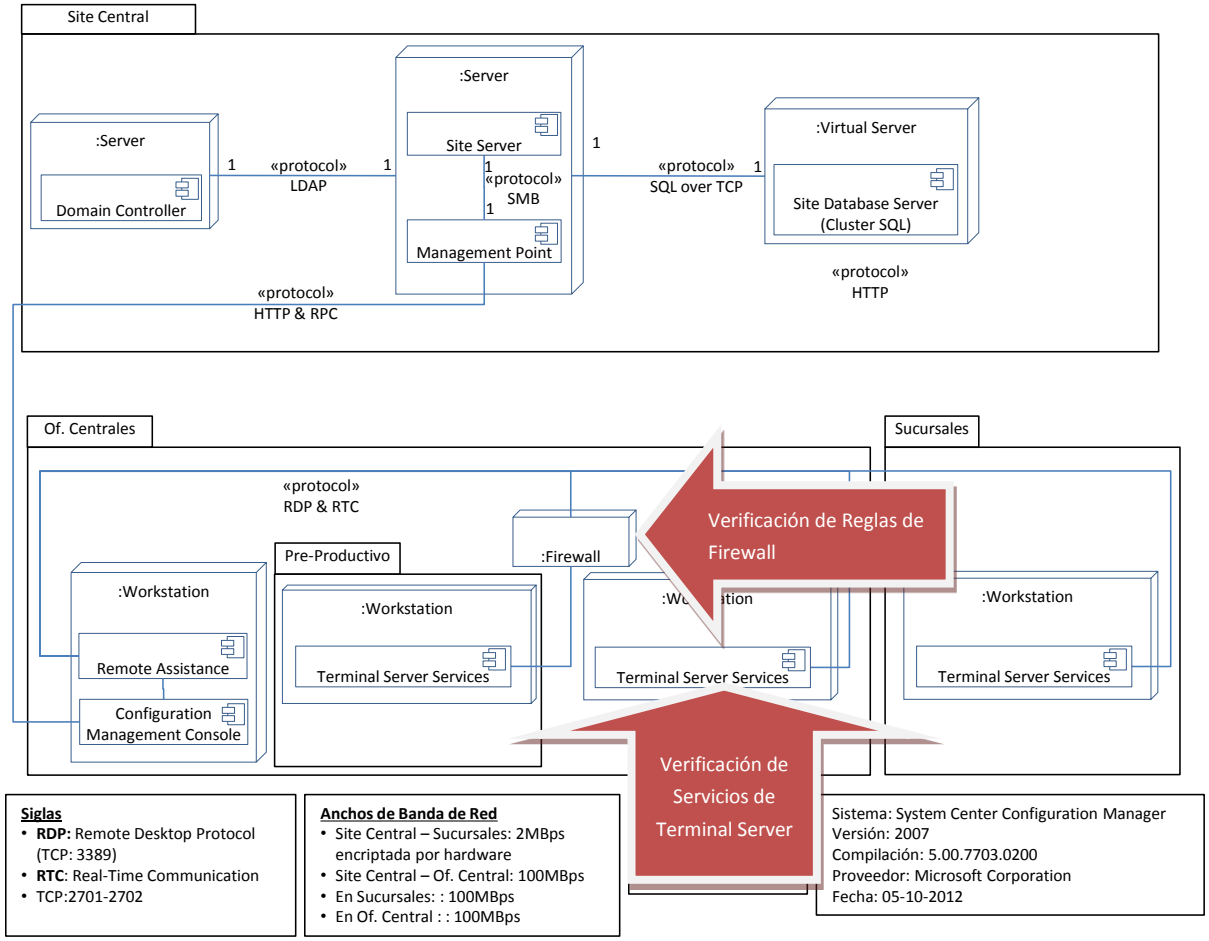


Figura V-4 Algunos puntos posibles de falla en el control remoto de workstation – remote assistance

CAPÍTULO VI.
CONCLUSIONES

Como arquitecto de sistemas, cuando se inició este proyecto de tesis, la motivación principal fue el control de la complejidad de las relaciones existentes entre los sistemas de un banco. Era necesario tener una forma de uniformizar la información de los sistemas para tener una vista eficaz para identificar las situaciones que se enfrentaban día a día y así tomar decisiones correctas. En ese contexto, y de manera natural se buscaron propuestas para representar o documentar arquitecturas de sistemas.

En este trabajo de tesis se encontró que, para documentar la arquitectura de un sistema, existen varios modelos que buscan guiar la identificación de vistas de arquitectura, que cumplan con los atributos de calidad que buscan sus stakeholders. Uno de esos modelos es el propuesto por el SEI, donde se da la facilidad de documentar sólo las vistas que se consideran más relevantes para cumplir con los atributos de calidad requeridos. En este modelo, es muy importante que los stakeholders se involucren en la confección de las vistas, ya que sus opiniones serán la guía para incluir datos que justificarán la existencia de cada una.

Una de las características principales del modelo del SEI es el contar con una guía de estilos de arquitectura, que son comúnmente utilizados para estructurar sistemas. Esta guía, además de servir como referencia para el diseño de sistemas, indica cuáles son los atributos de calidad que apoya cada estilo. Con esta relación Estilo-Atributo de Calidad, se encontró que es posible ubicar los estilos candidatos que podría utilizar un sistema para cumplir los atributos de calidad que buscan sus stakeholders, tanto para el diseño de un nuevo sistema como para la documentación de un sistema existente. Por lo tanto, cuando se conocen los estilos de arquitectura que utiliza un sistema y cuáles son los atributos de calidad que buscan sus stakeholders, se produce una identificación casi directa de las vistas que se deben incluir en la representación de su arquitectura.

La representación gráfica de las vistas de arquitectura se puede realizar utilizando varias notaciones e incluso creando las propias. Para este trabajo se utilizó UML 2.0, que es un lenguaje muy utilizado y flexible para el modelamiento de sistemas, y resultó ser útil y claro para explicar a los stakeholders los elementos que componen la solución.

Cuando se comenzó a desarrollar la representación de la arquitectura del sistema en estudio, se encontró una serie de elementos técnicos y funcionales que debieron ser claramente identificados para lograr una consistencia entre lo que hace el sistema y los elementos que permiten sus funcionalidades. La metodología propuesta en este trabajo, que permitió aplicar el modelo, fue una guía útil, ya que permitió avanzar ordenadamente en la documentación y considerar los elementos relevantes de acuerdo a las necesidades del proyecto.

El resultado de la comparación de los escenarios propuestos para validar la utilidad del modelo del SEI, permitió identificar secciones en las vistas de arquitectura que llevaron a inferir posibilidades de solución ante los escenarios planteados.

Comparando las soluciones obtenidas en base a la documentación realizada, versus las soluciones que se entregarían como resultado del análisis de la documentación existente antes de este proyecto, que incluye la descripción de todas las funcionalidades posibles del sistema y no especifica claramente el escenario real que está implementado en producción, se concluye que el modelo del SEI ayuda a seleccionar las vistas que deben documentarse para representar la arquitectura de un

sistema, y que son adecuadas para cumplir con los atributos de calidad requeridos por los stakeholders que participan del proceso de documentación.

6.1. Lecciones Aprendidas

Como en todo trabajo, existen recomendaciones que pueden ser útiles al momento de documentar una arquitectura cuando se utiliza el modelo del SEI y UML:

- Documentar la arquitectura tomando en cuenta las necesidades e intereses de los equipos que forman parte del proyecto y del uso normal del sistema.
- UML no es el único lenguaje para documentar la arquitectura. Existen diferentes notaciones y lenguajes para este propósito que quedaron fuera de los objetivos de este estudio. Por ejemplo, existen diversos lenguajes para descripción de arquitecturas (llamados ADLs, por sus siglas en inglés) que describen aspectos particulares de ésta.
- Se debe mantener una relación consistente entre las vistas. Esta consistencia aumenta su importancia cuando se debe documentar más de un tipo de vista.
- Elaborar plantillas de estilos para promover la reutilización de artefactos dentro de la organización. En este caso sólo se utilizaron los estereotipos estándar de UML, sin embargo podría haberse definido una especialización.
- Mantener actualizada la matriz de trazabilidad entre los requisitos y los elementos de la arquitectura.
- Tener bajo una línea base el documento de la arquitectura. En este caso sólo es una versión pero debería mantenerse un historial de modificaciones.

El tener disponible una buena documentación de la arquitectura de un sistema en estudio, permite comprender las relaciones entre los componentes que participan en la solución y ayuda a identificar posibles puntos de falla o puntos de integración con otros sistemas.
Una vez que se cuenta con las vistas documentadas, es recomendable generar casos que permitan verificar si la documentación realizada será útil para los equipos que la utilicen, y encuentren en ella un apoyo real para sus análisis futuros. También es recomendable que la información relacionada a variaciones en los componentes de la documentación sea fácil de actualizar, para reducir los costos de mantención de la documentación en el tiempo.

6.2. Trabajos Futuros

Para el banco de este estudio, la documentación de arquitecturas de sistemas presentada puede ser el punto de partida para implementar una metodología para documentar las arquitecturas de sus sistemas considerando mecanismos para la distribución de las vistas de arquitectura de acuerdo a las necesidades de cada stakeholder. El uso de un lenguaje estándar como UML en la documentación de las vistas de arquitectura permitiría que los proveedores que desarrollan sistemas participen de manera natural en la documentación de las vistas sin la necesidad de capacitarse por ser un lenguaje ampliamente difundido en la industria.

En general, como trabajos futuros para otros investigadores, se podría analizar qué tipos de vistas son más utilizadas por distintos tipos de organizaciones. Por ejemplo: para empresas que se dedican al desarrollo de sistemas podría ser recomendable tener tipos de vistas de componentes y conectores, y para empresas dedicadas a la distribución y venta de sistemas se utilizarían vistas de asignación.

Otras líneas de investigación podrían enfocarse a generar modelos de gestión de información de arquitectura, que permitan representar vistas de arquitectura dinámicamente, en base a estructuras de datos que entreguen datos técnicos de los sistemas, como sus componentes y relaciones. Esto ayudaría a la administración de la información de la arquitectura de sistemas existentes.

Bibliografía y Referencias

[1] CLEMENTS, PAUL et al. “Documenting Software Architectures: Views and Beyond”. Addison-Wesley. 2002

[2] PHILIPPE KRUCHTEN, Architectural Blueprints—The “4+1” View Model of Software Architecture, IEEE Software 12 (6), November 1995, pp. 42-50

[3] D. SONI, R.L. NORD AND C. HOFMEISTER. “Software Architecture in Industrial Applications,” Proceedings of the 17th International Conference on Software Engineering, Seattle, Washington, pp. 196-207, April 1995. © 1995 ACM.

[4] ISO, “Reference Model of Open Distributed Processing (RM-ODP)”. International Organization for Standardization. Technical Report 10746.

[5] Nicholas May, “A Survey of Software Architecture Viewpoint Models”, RMIT University, Melbourne, Australia, 2004

[6] IEEE, “IEEE Recommended Practice for Architectural Description of Software-Intensive Systems”. Institute of Electrical and Electronics Engineers, Sept. 2000. IEEE Std 1471-2000.

[7] Stephen T. Albin, “The Art of Software Architecture: Design Methods and Techniques”, John Wiley & Sons, 2003

[8] D. Garlan & M. Shaw, “An Introduction to Software Architecture,” Advances in Software Engineering and Knowledge Engineering, Vol. 1, World Scientific Publishing Co. (1993).

[9] Barbacci, M., Klein, M., Longstaff, T., & Weinstock, C. (1995). Quality Attributes. Carnegie Mellon University. Technical Report.

[10] Bass, L., Clements, P., & Kazman, R. (1998). Software Architecture in practice. Addison-Wesley.

[11] Carlos Reynoso, Nicolás Kiccillof (2004). Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.

[12] Nicolae-Zoran Constantinescu-Fülöp [2008]. A Desktop Grid Computing Approach for Scientific Computing and Visualization.

[13] Francisca Losavio et al: “Quality Characteristics for Software Architecture”, in Journal of Object Technology, vol. 2, no. 2, March-April 2003, pp. 133-150. http://www.jot.fm/issues/issue_2003_03/article2

[14] Ocharán Hernández, J. O., Cortés Verdín, M. K. (2008). Documentando Arquitecturas Orientadas a Aspectos para Líneas de Productos de Software. Research in Computing Science. Vol. 38.

[15] Krechetov, I., Tekinerdogan, B., Garcia, A., Chavez, C., and Kulesza, U., “Towards an integrated aspect-oriented modeling approach for software architecture design”. 8th workshop on aspect-oriented modelling, aosd.06. IEEE Trans. Soft. Eng, 1995.

[16] Tekinerdogan, Bedir, y otros [2007]. AOSD-Europe-UT-D76. AOSD-Europe,

[17] Booch, Grady (1993). Object-oriented Analysis and Design with Applications (2nd ed. ed.). Redwood City: Benjamin Cummings. ISBN 0-8053-5340-2.

ANEXO A. MODELO DE VISTAS DEL SEI

1.1. Tipo de Vista de Módulo

Un módulo [1] es una unidad que implementa un conjunto de responsabilidades, puede ser una clase, una colección de clases, una capa, o cualquier descomposición de código. Cada módulo tiene una colección de propiedades que intentan expresar información importante asociada a él, así como sus restricciones. Por ejemplo, sus propiedades son sus responsabilidades, visibilidad de la información, y autor o dueño. También, tienen relaciones con otros módulos, por ejemplo: "es-parte-de" o "deriva-de".

El concepto de módulos surgió en los 60s y 70s, basado en la idea de tener unidades de software con interfaces bien definidas, entregando un conjunto de servicios –generalmente procedimientos y funciones- conjuntamente con aplicaciones que esconden total o parcialmente sus estructuras de datos internas y algoritmos. Estos conceptos se encuentran en lenguajes de programación orientados a objetos y notaciones de modelamiento, como UML.

La forma en que un sistema de software es descompuesto en unidades manejables es una de las formas más importantes de la estructura del sistema. Como mínimo, determina cómo el código fuente del sistema está dividido en partes separadas, qué tipo de supuestos se puede hacer de cada parte sobre los servicios entregados por otras partes, y cómo esas partes son agregadas en grandes conjuntos. La elección de la modularización generalmente determina cómo un cambio en una parte de un sistema puede afectar otras partes y, en consecuencia, la habilidad de un sistema para soportar modificabilidad, portabilidad y reúso.

La documentación de tipos de vista de módulo describe las unidades principales -o módulos- de un sistema y las relaciones entre ellas. Estas descripciones en el modelo de vistas del SEI son llamadas vistas de módulos.

1.1.1. Elementos, Relaciones y Propiedades

La Tabla A-1 resume los elementos, relaciones y propiedades del tipo de vista de módulo.

Elementos	El elemento de una vista de módulo es un **módulo**, el cual es una unidad de aplicación de software que entrega una unidad coherente de funcionalidad.
Relaciones	Las relaciones mostradas en una vista de módulo están formadas de: • **es-parte-de:** define una parte o la relación completa entre el submódulo A (la parte, o hijo) y el módulo agregado B (su totalidad, o padre) • **depende-de:** define una relación de dependencia entre A y B. Un estilo de módulo específico indica que significa la dependencia • **es-un o deriva-de:** define una relación de generalización entre un módulo más específico (el hijo A) y un módulo más general (el padre B)
Propiedades de	Las propiedades de un módulo incluyen:

Elementos	• **Nombre:** el cual puede cumplir con reglas en un espacio de nombres. • **Responsabilidades:** define las responsabilidades del módulo. • **Información de Implementación:** un conjunto de unidades de código que implementan el módulo
Propiedades de Relaciones	• La relación **es-parte-de** puede tener una propiedad **visible** asociada que define si un módulo es visible fuera del módulo agregado • La relación **depende-de** puede tener **restricciones** asignadas para especificar en más detalle cual es la dependencia entre dos módulos • La relación **es-un** o **deriva-de** puede tener una propiedad de **aplicación**, indicando que es un módulo más específico –el hijo A- deriva de la aplicación de un módulo más general –el padre B- pero no garantiza soportar las interfaces del padre y de tal modo no entrega reemplazo para el padre.
Topología	El tipo de vista de módulo no tiene limitaciones inherentes a topología.

Tabla A-1 Resumen del Tipo de Vistas de Módulos

1.1.1.1. Elementos

Existen diseñadores de sistemas que utilizan el termino módulo para referirse a una variedad de estructuras de software, incluyendo unidades de lenguajes de programación como paquetes Ada, módulos Modula, Smalltalk, o clases C++, o simplemente agrupamientos generales de unidades de código fuente.

La caracterización de un módulo se realiza mediante la descripción de un conjunto de responsabilidades que son principalmente entre las propiedades de un módulo. El término "responsabilidades" corresponde a tipos de características que una unidad de software puede entregar, incluyendo servicios, así como sus variables internas y externas.

Los módulos pueden ser agregados y descompuestos. Se pueden identificar diferentes vistas de módulos en un conjunto diferente de módulos y agregarlos o descomponerlos basados en diferentes criterios de estilo. Por ejemplo: el estilo de capas identifica módulos y los agrega basado en una relación **puede-usar**, mientras que una vista de generalización identifica y agrega módulos basados en los que ellos tienen en común.

1.1.1.2. Relaciones

El tipo de vista de módulo se caracteriza por tener las siguientes relaciones:

- **es-parte-de**. La relación es-parte-de define una parte o toda relación entre el submódulo A –la parte- y el módulo agregado B –el todo. En su forma más general, la relación es-parte-de indica simplemente agregación. En general, un módulo podría ser incluido en muchas agregaciones.
- **depende-de**. A depende de B define una relación de dependencia entre A y B. La relación depende-de generalmente se utiliza en el proceso de diseño cuando todavía se debe decidir la forma precisa de dependencia. Una vez que la decisión es tomada, la relación **depende-de** puede ser reemplazada por una relación más específica como: **usa, puede-usar, transversal y relacionamiento** de entidades de datos, en los estilos de usos, de capas, de aspectos, y de modelos de datos respectivamente. Otros ejemplos más específicos de la

relación **depende-de** incluyen **comparte-datos-con** y **llama**. Una dependencia **llama** puede ser refinada a: **envía-datos-a**, **transfiere-control-a**, **impone-ordenes-a**, y así sucesivamente.

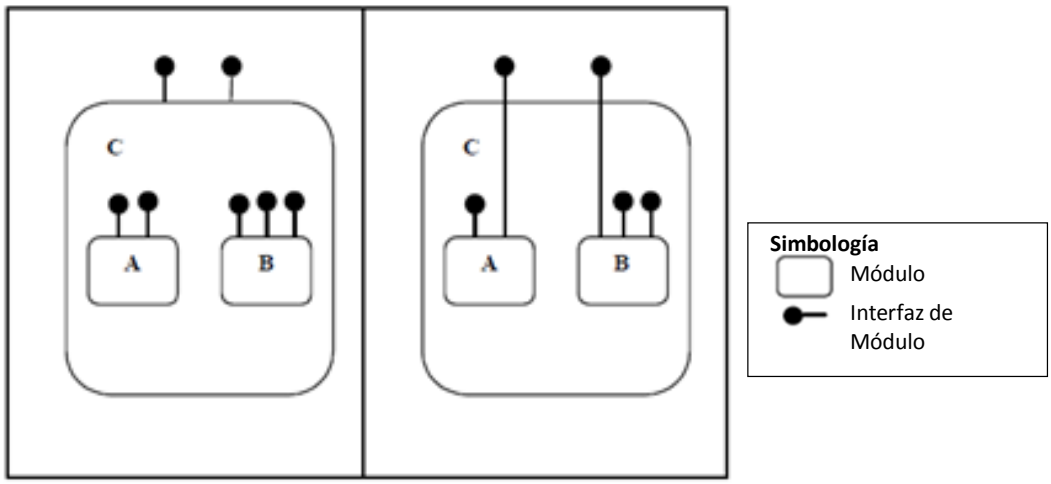
- **es-un** o **deriva-de**. La relación **es-un** define una relación de generalización entre un módulo más específico –el hijo A- y un módulo más general – el padre B. El módulo hijo es capaz de ser usado en contextos en los cuales el padre es usado. Luego, miramos su uso en más detalle en el estilo de generalización de módulo. La herencia de Orientación a Objetos es un caso especial de la relación es-un.

1.1.1.3. Propiedades

Las propiedades son documentadas como parte de la documentación de ayuda para una vista. La lista de propiedades adecuadas a un conjunto de módulos dependerá de muchas cosas pero es importante incluir las siguientes:

- **Nombre**. El nombre de un módulo es la manera principal para referirse a él. El nombre de un módulo debe sugerir algo acerca de sus roles en el sistema: por ejemplo, un módulo llamado “account_mgr” tiene probablemente poco que ver con simulaciones numéricas de reacciones químicas. También, el nombre de un módulo puede reflejar su posición en una descomposición jerárquica; el nombre A.B.C.D, por ejemplo, se refiere a un módulo D que es un submódulo de un módulo C, el mismo un submódulo de B, y así sucesivamente.
- **Responsabilidad**. La propiedad de responsabilidad para un módulo es una manera de identificar sus roles en el sistema y establece una identidad aparte del nombre del módulo. Aunque el nombre de un módulo puede sugerir sus roles, un estado de responsabilidades lo establece con mucha más certeza. Las responsabilidades deberían ser descritas con suficiente detalle para dejar claro al lector qué hace cada módulo.
- **Visibilidad de Interfaces**. Un documento de interfaces de un módulo debe establecer con precisión sus roles en el sistema para especificar exactamente como puede ser llamado. Un módulo puede tener cero, una, o muchas interfaces.

En una vista de documentación de una relación **es-parte-de**, algunas de las interfaces de los submódulos existen sólo para propósitos internos; esto significa que las interfaces son usadas sólo por los submódulos dentro del módulo padre contenedor. Esas interfaces nunca están visibles fuera del contexto y por lo tanto no tienen relación directa con las interfaces padre. Se pueden utilizar diferentes estrategias para aquellas interfaces que tienen una relación directa con las interfaces padre. Por ejemplo, la estrategia mostrada en la Figura A-1 es encapsulación, debido a que esconde las interfaces en submódulos. En ella, el módulo padre entrega sus propias interfaces y mapea todas las solicitudes, usando las capacidades entregadas por los submódulos. Alternativamente, las interfaces de un módulo de agregación pueden ser un subconjunto de las interfaces del agregado. Por ejemplo, un módulo cercano simplemente agrega un conjunto de módulos y selectivamente expone algunas de sus responsabilidades. Generalmente, las capas y los subsistemas se definen de esta forma. Un ejemplo más concreto es si el módulo C es una agregación de los módulos A y B, las interfaces implícitas de C serán un subconjunto de las interfaces de los módulos A y B.



a) El módulo C entrega su propia Interfaz, escondiendo las interfaces de los módulos A y B.

b) El módulo C expone como propias un subconjunto de las interfaces de los módulos A y B

Figura A-1 Encapsulación de Módulos

- **Información de Aplicación.** Debido a que los módulos son unidades de aplicación, es útil recordar información relacionada a su aplicación desde el punto de vista de la administración del desarrollo y construcción del sistema que los contiene. Es conveniente recordar en la documentación de la arquitectura donde está definido el módulo. La información de aplicación puede incluir:

    - Unidades de Asignación de Código. Identifica los archivos que constituyen la aplicación de un módulo. Por ejemplo, un módulo ALPHA, si se implementó en C, puede tener muchos archivos que constituyen su aplicación: ALPHA.c, ALPHA.h, ALPHA.o –si las versiones pre-compiladas son mantenidas- y quizás ALPHA_t.h para definir cualquier tipo de datos entregado por ALPHA.
    - Información de Pruebas. Es importante almacenar el plan de pruebas de módulos, casos de prueba, pruebas de scaffolding[2], datos de prueba, e historias de pruebas.
    - Información de Administración. Un administrador puede necesitar la planificación de cuando se esperaba un módulo y su presupuesto.
    - Restricciones de Implementación. En muchos casos, un módulo tendrá una estrategia de implementación y restricciones que se deben seguir. Esta información es privada al módulo y, por lo tanto, no aparecerá, por ejemplo, en las interfaces del módulo.

Además de estas propiedades, los estilos en el tipo de vistas de módulo pueden tener otras propiedades.

[2] El scaffolding (andamiaje) en aplicaciones es una técnica que permite a un desarrollador definir y crear aplicaciones básicas que pueden crear, leer, actualizar y borrar objetos.

1.1.2. Algunos Estilos del Tipo de Vista de Módulos

En el tipo de vista de módulo se pueden encontrar los siguientes estilos:

- Estilo de Descomposición (Decomposition style)
- Estilo de Usos (Uses style)
- Estilo de Generalización (Generalization style)
- Estilo de Capas (Layered style)
- Estilo de Aspectos (Aspects Style)
- Estilo de Modelo de Datos (Data Model)

A continuación de describirá cada uno de estos estilos.

1.1.2.1. Estilo de Descomposición

El estilo de descomposición representa una vista top-down del sistema, descomponiendo el código en sistemas, subsistemas, sub-subsistemas, y así sucesivamente. Además, el término sistema o subsistema tiene una interpretación en tiempo de ejecución, y se utiliza para aclarar un contexto particular. Este estilo es utilizado para dar una vista de alto nivel del sistema y sus piezas a los stakeholders, es particularmente útil para la educación y la comunicación a niveles de administración. Ayuda a los nuevos miembros del equipo de desarrollo a entender cuáles son sus roles en términos de desarrollo de código y, generalmente, es la base para la asignación de trabajo y tener medidas de completitud.

Elementos, Relaciones y Propiedades

La Tabla A-2 resume las características del estilo de descomposición. Los elementos de este estilo son los módulos, algunos módulos contienen otros que son llamados subsistemas. La relación principal, la relación **descompuesto-en**, que es una forma de la relación **es-parte-de** y tiene como principal restricción la garantía de que un elemento puede ser una parte de a lo más una agregación. La descomposición de módulos puede definir si los submódulos son accesibles dentro del módulo agregado –módulo padre- o además a otros módulos. La visibilidad de los submódulos puede ser descrita en el catálogo de elementos de la vista o transmitida gráficamente.

Resumen	El estilo de descomposición es usado para descomponer el sistema en unidades de implementación. Una vista de descomposición describe la organización del código en módulos y sub-módulos y muestra cómo se distribuyen las responsabilidades del sistema
Elementos	• Módulo
Relaciones	• Relación **descompuesto-en**, la cual es una forma de una relación **es-parte-de**. La documentación debe especificar el criterio usado para definir la descomposición
Restricciones	• No se permiten loops en el gráfico de descomposición • Un módulo puede tener sólo un padre
Para qué sirve	• Para analizar y comunicar en trozos digeribles la estructura del software a los recién llegados. • Para proveer un punto de apoyo a la asignación de trabajo • Para analizar la ubicación de los cambios a realizar en un sistema

Tabla A-2 Resumen del Estilo de Descomposición

¿Para qué es este estilo?

Una vista de descomposición representa las responsabilidades de un sistema en piezas que sean manejables intelectualmente que se refinan para transmitir más detalles. Por lo tanto, este estilo es una herramienta excelente para el aprendizaje y la exploración de los que recién llegan a un proyecto y otras personas que no tienen por qué saber los detalles funcionales del mismo. La agrupación de responsabilidades en este estilo también constituye una base útil para definir elementos de configuración dentro de un framework de gestión de configuración.

Una vista de descomposición generalmente sirve como entrada para la vista de asignación de trabajo del sistema, la cual asigna las partes del software a las unidades organizativas o equipos que lo implementarán y probarán. Además, una vista de descomposición entrega ayuda para analizar los efectos de los cambios, pero debido a que esta vista no muestra todas las dependencias entre módulos, no se puede esperar realizar un análisis de impacto completo. Aquí se requieren vistas como las del estilo de usos que muestran las relaciones de dependencia.

Relación con otros estilos

Es posible, y deseable, relacionar una vista de descomposición con una o más vistas de componentes y conectores. Este tipo de relaciones sirve para indicar cómo las implementaciones de estructuras de software se relacionan a estructuras en tiempo de ejecución, siendo esta una relación muchos-a-muchos. El mismo módulo podría implementar todas o partes de muchos componentes o conectores. Inversamente, un componente podría requerir muchos módulos para su implementación.

El estilo de descomposición está estrechamente relacionado al estilo de asignación de trabajo, un tipo de vista de asignación descrito en detalle en la sección 1.3.2. El estilo de asignación de trabajo relaciona los módulos resultantes de una descomposición a un conjunto de equipos responsables de implementar y probar esos módulos.

Notaciones

Notaciones Informales

En notaciones informales, los módulos en el estilo de descomposición se representan generalmente como cajas nombradas que contienen otras cajas nombradas. También puede mostrarse a través de listas con los nombres de módulos y usando indentación para indicar la relación es-parte-de cómo muestra la Figura A-2.

Hardware Hiding Module
- Extended Computer Module
  - Data Module
  - Input/Output Module
  - Computer State Module
  - Parallelism Control Module
  - Program Module
  - Virtual Memory Module
  - Interrupt Handler Module
  - Timer Module
- Device Interface Module
  - Air Data Computer Module
  - Angle of Attack Sensor Module
  - Audible Signal Device Module
  - Computer Fail Device Module
  - Doppler Radar Set Module
  - Flight Information Displays Module
  - Forward Looking Radar Module
  - Head-Up Display Module
  - Inertial Measurement Set Module
  - Input-Output Representation Module
  - Master Function Switch Module
  - Panel Module
  - Projected Map Display Set Module
  - Radar Altimeter Module
  - Shipboard Inertial Nav System Module
  - Slew Control Module
  - Switch Bank Module
  - TACAN Module
  - Visual Indicators Module
  - Waypoint Info. System Module
  - Weapon Characteristics Module
  - Weapon Release System Module
  - Weight on Gear Module

Behavior Hiding Module
- Function Driver Module
  - Air Data Computer Module
  - Audible Signal Module
  - Computer Fail Signal Module
  - Doppler Radar Module
  - Flight Information Display Module
  - Forward Looking Radar Module
  - Head-Up Display Module
  - Inertial Measurement Set Module
  - Panel Module
  - Projected Map Display Set Module
  - Shipboard Inertial Nav System Module
  - Visual Indicator Module
  - Weapon Release Module
  - Ground Test Module
  - Shared Services Module
- Mode Determination Module
  - Panel I/O Support Module
  - Shared Subroutine Module
  - Stage Director Module
  - System Value Module

Software Decision Hiding Module
- Application Data Type Module
  - Numeric Data Type Module
  - State Transition Event Module
- Data Banker Module
  - Singular Values Module
  - Complex Event Module
  - Filter Behavior Module
- Physical Models Module
  - Aircraft Motion Module
  - Earth Characteristics Module
  - Human Factors Module
  - Target Behavior Module
  - Weapon Behavior Module
- Software Utility Module
  - Power-Up Initialization Module
  - Numerical Algorithms Module
- System Generation Module
  - System Generation Parameter Module
  - Support Software Module

Figura A-2 Notación Informal de una vista de descomposición representando la relación descompuesto-en a través de indentación [10]

En la notación de anidación puede usarse la negrilla –y explicarlo en la simbología- indicando que los hijos no son visibles fuera del padre. Si no se dispone de una notación visual para indicar visibilidad, se puede definir textualmente, como se realiza para otras propiedades.

Notaciones Semiformales

Notación UML

En UML, se pueden utilizar paquetes para representar módulos que contienen otros módulos. Un paquete puede contener clases y otros paquetes; normalmente se utilizan cajas de clase para terminar la descomposición. En UML la descomposición se puede representar de dos formas:

1. Anidando módulos (como en la Figura A-3).
2. Mostrar una sucesión de dos diagramas, primero una representación de alto nivel y, luego, una representación de los contenidos del módulo mostrado al principio (como en la Figura A-4 y la Figura A-5)

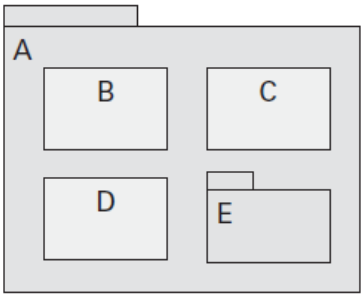


Figura A-3 En UML, la descomposición de módulos se representa por anidamiento, la agregación de módulo se representa como un paquete

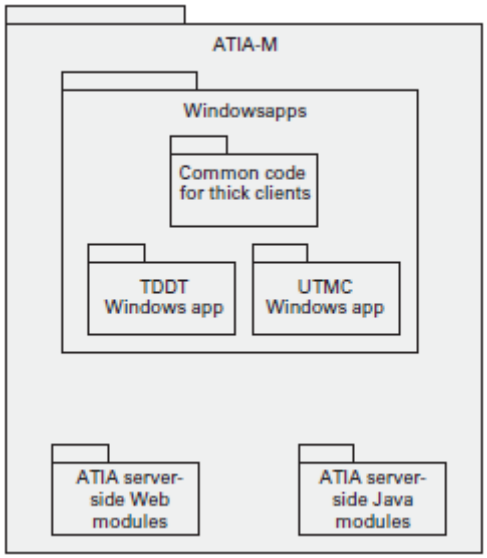


Figura A-4 Vista de Descomposición de alto nivel de un sistema ATIA-M

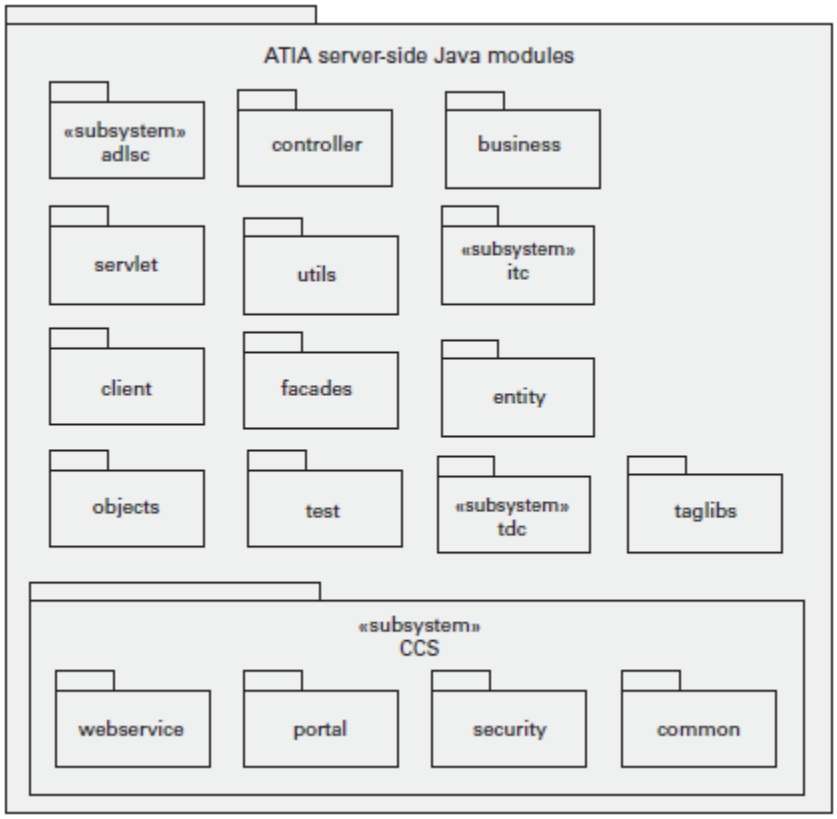


Figura A-5 Refinamiento de los módulos JAVA del lado del servidor del sistema ATIA-M y la forma en que se descompone en submódulos

1.1.2.2. Estilo de Usos

Los estilos de usos son un resultado donde la relación **depende-de** se especializa a una relación de **uso**. Por ejemplo: una unidad de software P_1 se dice que usa otra unidad, P_2, si la ejecución de P_1 depende de que P_2 se ejecute correctamente. Este estilo indica a los desarrolladores los otros módulos que deben existir para que su porción del sistema sea ejecutado correctamente. Ayuda al desarrollo incremental, debido a que posibilita la identificación de subconjuntos útiles del sistema que pueden ser entregados tempranamente.

Elementos, Relaciones y Propiedades

En la Tabla A-3 se resumen las características del estilo de usos. Los elementos de este estilo son los módulos. Este define que la especialización de la relación **depende-de** es la relación de **uso**, debido a que un módulo requiere la implementación correcta de otro módulo para su funcionamiento correcto. Esta vista indica explícitamente que módulos usan a otros para especificar sus responsabilidades.

Resumen	El estilo de usos muestra como cada módulo depende de otros; es útil para la planificación porque ayuda a definir subconjuntos e incrementos del sistema que está siendo desarrollado
Elementos	• Módulo
Relaciones	• La relación **usa**, que es una forma de la relación **depende-de**. Por ejemplo: Un módulo A usa un módulo B si A depende de la presencia y del funcionamiento correcto de B para satisfacer sus propios requerimientos.
Restricciones	• No tiene restricciones topológicas. Sin embargo, si las relaciones de usos presentan loops, o grandes cadenas de dependencias, perjudicará el rendimiento de la arquitectura que se entregue.
Para qué sirve	• Planeación de desarrollos incrementales y partes de ellos • Depuración y pruebas • Evaluar el impacto de cambios • Administración de dependencias de sistemas en construcción

Tabla A-3 Resumen del Estilo de Usos

¿Para qué es este estilo?

Este estilo es utilizado para planeación de desarrollos incrementales, extensiones de sistemas y partes de estos, depuración y pruebas, y evaluar el impacto de cambios específicos. La Figura A-6 muestra la presentación principal de una vista de usos y cómo esta puede ayudar con el desarrollo incremental. Para definir subconjuntos incrementales, los módulos deben ser definidos con un nivel de granularidad adecuado. En el ejemplo, puede ocurrir que admin.core no necesite el paquete dao, sólo un submódulo de mismo; con esto, el diagrama debería mostrar los submódulos de dao.

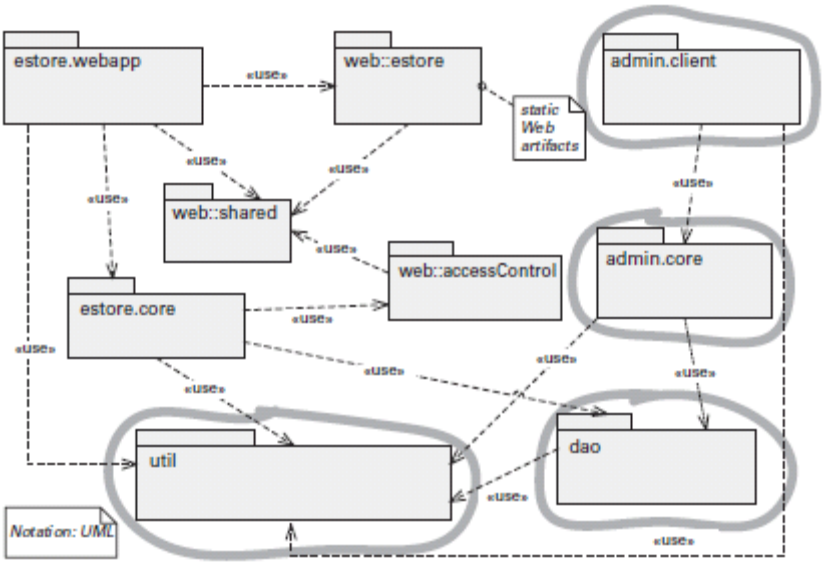


Figura A-6 Ejemplo de Vista de Usos

Además, la vista de usos ayuda en la administración de las dependencias de sistemas en construcción o mantención. La meta de esta tarea es controlar la complejidad y evitar la degradación de la mantenibilidad del sistema debido a la inclusión de dependencias indeseadas.

Relación con otros estilos

El estilo de usos se complementa directamente con el estilo de capas para establecer una relación puede-usar. La relación puede-usar generalmente se usa primero y contiene directivas generales que definen los grados de libertas para la implementación. Una vez que se han elegido las alternativas de implementación, aparece la vista de usos y gobierna la producción de subconjuntos incrementales.

Cuando un módulo contiene submódulos, la descomposición requiere que la relación de uso involucre el módulo agregado mapeado al submódulo usando esa relación. En la Figura A-8 (b) el módulo User Interface es descompuesto en los módulos A, B y C. Al menos uno de los módulos debe depender del módulo de acceso a dato; de otra forma, la descomposición no es consistente.

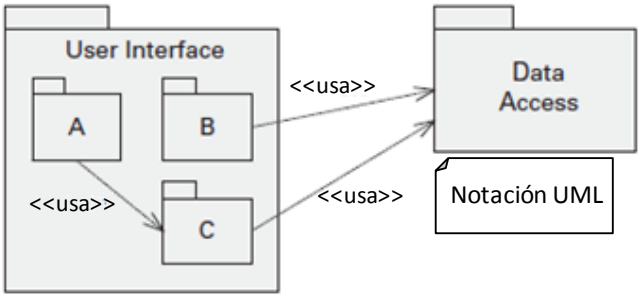


Figura A-7 Relación de usos de módulos por submódulos agregados.

Notaciones

Notaciones Informales

La relación de usos puede ser documentada con una tabla de dos columnas con los elementos que usan a la izquierda y los elementos usados a la derecha (como en la Tabla A-4). Alternativamente, las notaciones gráficas informales pueden mostrar la relación usando diagramas de cajas y líneas con una llave. Para definir subconjuntos, a veces la mejor alternativa es una tabulación –no gráfica-. Es más fácil buscar las relaciones detalladas en una tabla que encontrarlas en un diagrama, el que rápidamente puede crecer desordenadamente a menos que sea particionado usando un refinamiento por descomposición.

Módulo	Módulo Usado
A	C
B	Data Access
C	Data Access

Tabla A-4 Notación informal para una vista de usos

Notaciones Semiformales

UML

En UML el estilo de usos se representa fácilmente utilizando paquetes UML para representar módulos; la relación de usos se representa como una dependencia con el estereotipo <<use>>. En la Figura A-8 (a), el módulo User Interface tiene una dependencia de usos con el módulo Data Access.

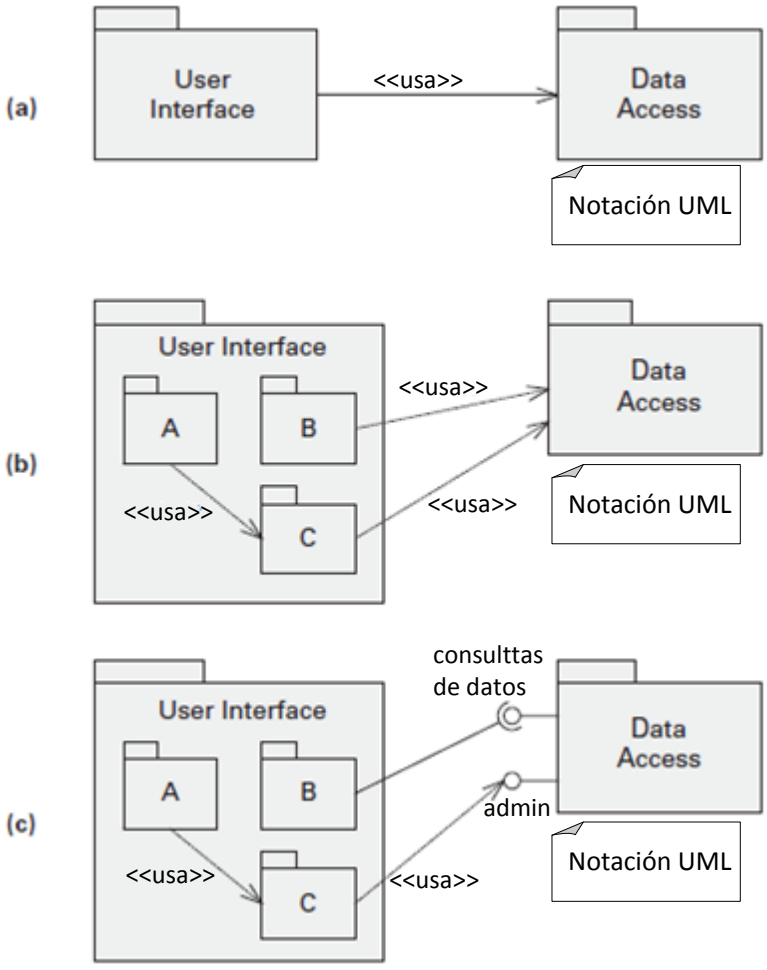


Figura A-8 Ejemplo de Vistas de Usos con UML

Matriz de Dependencia

La relación de usos también puede ser documentada a través de una matriz cuadrada, con los módulos listados en filas y columnas. Un diagrama como el de la Figura A-9 puede verse como un grafo dirigido donde los paquetes son vértices y las dependencias son arcos (como lo muestra la matriz de dependencia de la Figura A-10).

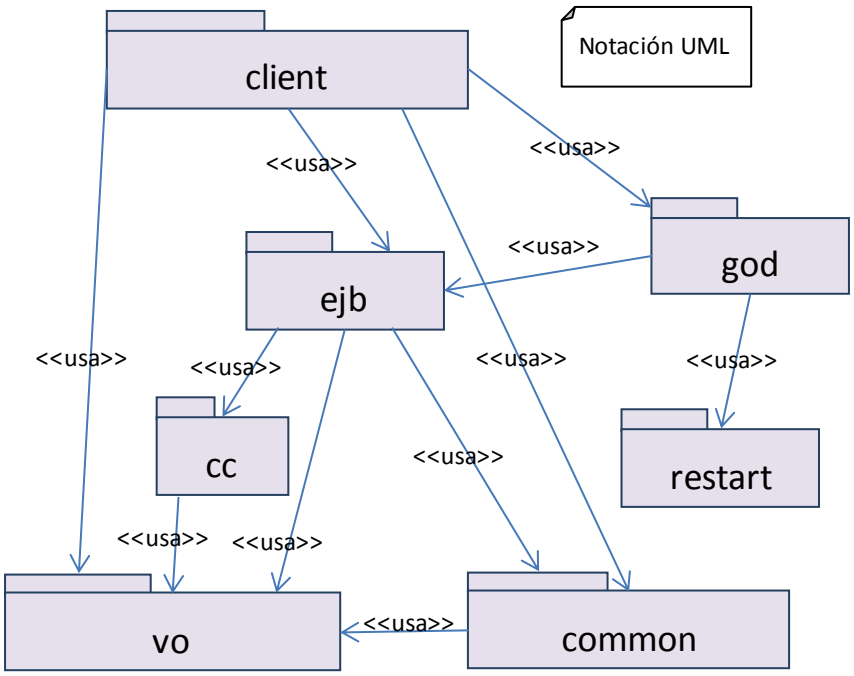


Figura A-9 Diagrama de Paquetes UML mostrando dependencias de uso

Usando Módulo / Módulos Usados	client	ejb	cc	god	restart	common	vo
client							
ejb	1			1			
cc		1					
god	1						
restart				1			
common	1	1					
vo	1	1	1			1	

Figura A-10 Ejemplo de Vistas de Usos con una matriz de dependencia

1.1.2.3. Estilo de Generalización

El estilo de generalización se ocupa cuando existe una relación es-un. Este estilo es útil cuando un arquitecto quiere apoyar la extensión y evolución de arquitecturas y elementos individuales. Muestra cómo se relacionan entre sí las diferentes unidades de código, como en una jerarquía de clases. Mirando el estilo de generalización se puede determinar qué clases derivan de qué otras. Este estilo es utilizado para expresar diseños orientados a objetos y, también, para ayudar de una variedad de formas al mantenimiento.

La generalización puede representar herencia de interfaces, implementación o ambas. En una descripción de arquitectura, se enfatiza el compartir y reusar interfaces y no mucho en su implementación.

Elementos, Relaciones y Propiedades

La Tabla A-5 resume las características del estilo de generalización. El elemento del estilo de generalización es el módulo, la generalización es definida como una relación **es-un**. En esta relación un módulo es una generalización (un padre) de otros módulos (hijos), y estos otros módulos son una especialización del primero. Un módulo puede ser abstracto, por lo que no contiene una implementación completa, Los módulos hijos de un módulo abstracto necesitan ser implementados, de lo contrario también deberían ser abstractos. Un módulo que hereda información se conoce como un descendiente; el módulo que provee la información es un ancestro.

Resumen	El estilo de generalización emplea la relación es-un para apoyar la extensión y la evolución de arquitecturas y elementos individuales. En este estilo, los módulos son definidos de tal manera que capturan las características comunes y sus variaciones.
Elementos	• Módulo. Un módulo puede tener propiedades abstractas para indicar que no contiene una implementación completa.
Relaciones	• **generalización**, que es una especialización de la relación **es-un**. La relación puede ser más especializada para indicar, por ejemplo, si es herencia de clase, de interface, o la implementación de una interface.
Restricciones	• Un módulo puede tener múltiples padres, aunque la herencia múltiple es considerada como un enfoque de diseño peligroso. • No se permiten ciclos en la generalización; por ejemplo, un módulo hijo no puede ser una generalización de uno o más de sus módulos predecesores en una vista.
Para qué sirve	• Expresar herencia en diseños orientados a objetos • Descripción de la evolución y extensiones incrementales • Captura de puntos en común, con sus variaciones como hijos • Apoyar el reúso

Tabla A-5 Resumen del Estilo de Generalización

¿Para qué es este estilo?

El estilo de generalización se utiliza para apoyar:

- Diseños orientados a objetos. Es el principal medio para expresar herencia en el diseño de sistemas orientados a objetos
- Extensiones. Generalmente, es más fácil entender cómo un módulo se diferencia del otro bien conocido en lugar de tratar de entender un nuevo módulo desde cero.
- Cambios Locales o Variaciones. Uno de los propósitos de la arquitectura es proporcionar una estructura estable global que permita cambios locales o variaciones. La generalización es un enfoque para definir elementos comunes en los niveles superiores y para definir las variaciones en los hijos de un módulo.
- Reuso. Encontrar módulos reutilizables es un subproducto para otros fines. Las abstracciones adecuadas pueden ser reutilizadas a nivel de interfaz o también se puede realizar a nivel de implementación. La definición de módulos abstractos genera una oportunidad para el reúso.

Relación con otros estilos

Las relaciones de herencia e implementación de interfaces complementan otras relaciones de módulos y generalmente se encuentran en vistas de módulos junto con las relaciones de uso y paquetes de descomposición. Pero para los diseñadores esto implica una jerarquía de módulos compleja. Es útil mostrar relaciones de herencia en diagramas separados de otros tipos de relaciones.

Notaciones

Notaciones Semiformales

UML

Expresar la generalización es la esencia de UML, en ella los módulos generalmente se muestran como clases o interfaces.

La Figura A-11 muestra una notación básica para representar la herencia de clases o interfaces utilizando UML, donde el concepto de herencia se expresa a través de una línea sólida con una punta en forma de triángulo. UML permite puntos suspensivos en lugar de submódulos para indicar que se puede tener más elementos hijos como los ya mostrados.

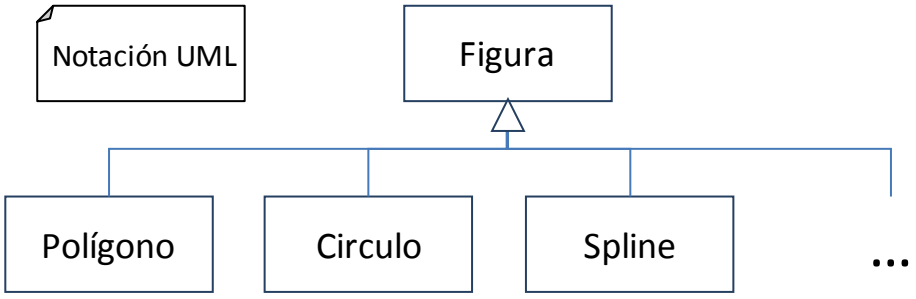


Figura A-11 Ejemplo de herencia en UML

La Figura A-12 muestra cómo se realiza una implementación de interfaces en UML. La implementación de interfaces es un tipo de generalización y puede ser representada de dos formas: una línea segmentada con una punta en forma de triángulo desde el módulo hasta la interface que implementa, o un símbolo en forma de paleta para la interface que se conecta al módulo que es implementado.

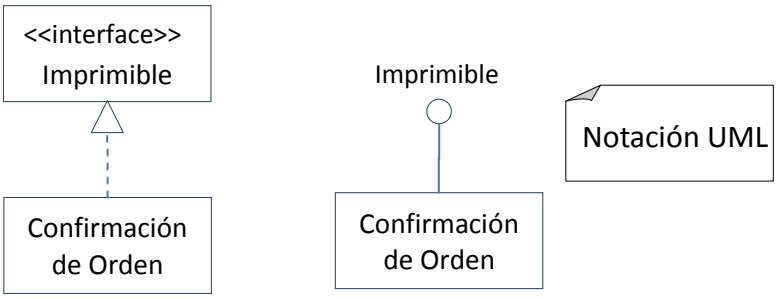


Figura A-12 Ejemplo de Implementación de Interfaz en UML

1.1.2.4. Estilo de Capas

El estilo de capas, como otros estilos de módulos, refleja la división del código en unidades. En este caso, las unidades se agrupan en capas separadas donde se permite que el código de las capas superiores utilice el código de capas inferiores de acuerdo a reglas predefinidas, por lo que estas reglas reflejan una relación unidireccional **puede-usar**. Por ejemplo, las reglas pueden especificar que sólo el código de la próxima capa inferior pueda ser usado, que el código de cualquiera de las capas inferiores pueda ser usado, o que el código de las capas inferiores o de la capa de utilidades pueda ser usado. Este estilo es utilizado para mostrar cómo el código es descompuesto en máquinas virtuales. Generalmente, los niveles más bajos involucran aquellas porciones del sistema que están más cerca del hardware (incluyendo el sistema operativo), mientras que las capas superiores contienen más software dependiente de la aplicación. El Estilo de Capas es utilizado para la educación, y permitir la reutilización y la portabilidad. Teniendo el código dependiente del hardware localizado, por ejemplo, ayuda al cambio de hardware.

Las capas particionan completamente el software y cada partición –a través de su interfaz publica— entrega un conjunto integrado de servicios. Pero eso no es todo, la Figura A-13 muestra deliberadamente ejemplos de divisiones donde no se identifican las unidades de software o cómo interactúan.

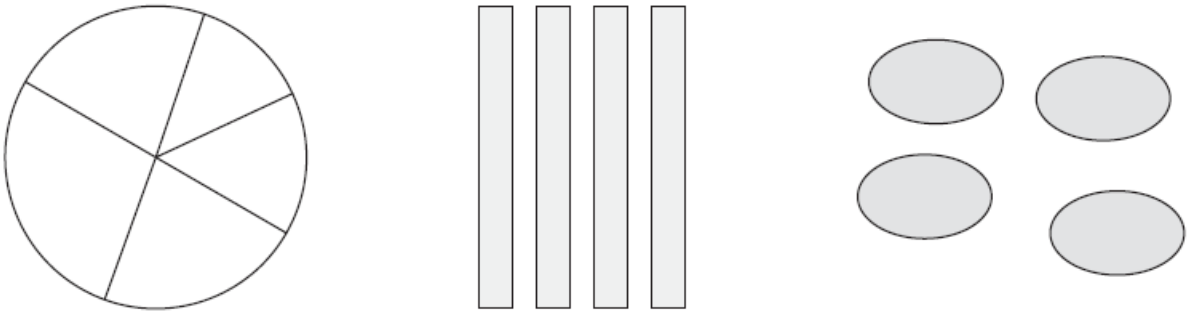


Figura A-13 Tres divisiones diferentes de software

La propiedad fundamental de las capas es la estricta relación de orden donde cada una puede interactuar con la siguiente. Por ejemplo, si (A, B) es esta relación, se dice que la implementación de la capa A permite que se use cualquiera de las facilidades provistas por la capa B.

Al utilizar el término uso, existen algunas indefiniciones. Por ejemplo, si A se implementa utilizando sólo las facilidades que entrega B ¿se está implementando solo con B? esto puede ocurrir o no, por ejemplo, si se supone que las capas muestran horizontalmente, uno encima del otro, Algunos

esquemas de capas permiten que una capa use las facilidades públicas de cualquier capa inferior, no sólo la capa inferior más cercana. Otros esquemas colocan capas que son colecciones de utilidades que pueden ser utilizadas por cualquier capa. Pero ninguna arquitectura que pueda nombrarse como de capas permite a una capa utilizar, sin restricción alguna, las facilidades de una capa superior. Permitir el uso sin restricciones de las capas superiores destruye las propiedades deseables que proveen las arquitecturas de capas. El uso de capas fluye generalmente hacia abajo. Sólo se permite un pequeño número de casos bien definido y son considerados como la excepción a la regla. Por lo tanto, la Figura A-14 se asemeja un estilo de capas pero no lo es.

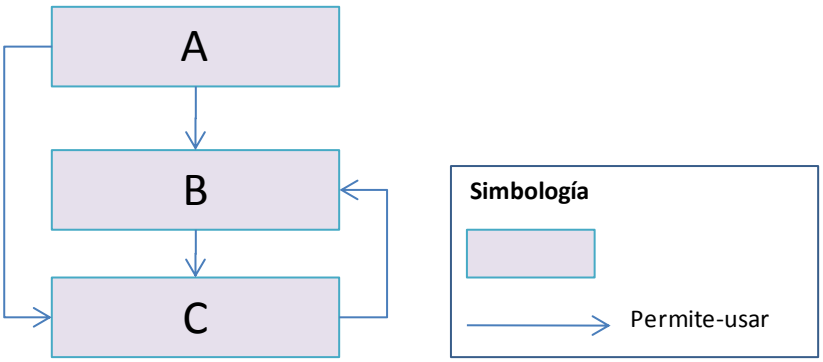


Figura A-14 Tal vez existen 3 capas pero no es un estilo de capas debido al uso de capas superiores

En algunos casos, los módulos de niveles muy altos requieren utilizar módulos de capas muy inferiores donde normalmente se permite utilizar sólo la próxima capa inferior. En esos casos se debe documentar la excepción en el diagrama o un documento anexo. A esta situación se le llama puente de capas. Si existen muchos de estos puentes el sistema estará pobremente estructurado, al menos con respecto a los objetivos de portabilidad y la modificabilidad que estilo de capas intenta apoyar. Los sistemas con usos de capas superiores no están de acuerdo estrictamente con la definición de capas. Aunque en tales casos, el estilo de capas puede representar la aproximación más cercana a la realidad y además cubrir el diseño ideal que un arquitecto este tratando de modelar.

Las capas no pueden estar derivadas del examinar el código fuente, ya que son agrupamientos lógicos de gran ayuda para la creación y comunicación de una arquitectura que generalmente no están delimitados explícitamente en el código fuente. El código fuente puede revelar lo que se usa pero las capas expresan la relación **puede-usar**.

Algunos de los criterios usados en la definición de capas de sistemas están basados en la esperanza de que se volverán independientes en diferentes escalas de tiempo, que diferentes personas con diferentes capacidades trabajaran en diferentes capas, y que se esperarán diferentes niveles de reúso en diferentes capas.

Elementos, Relaciones y Propiedades

La Tabla A-6 resume las características del estilo de capas.

Resumen	El estilo de capas pone las capas (agrupaciones de módulos que ofrecen un conjunto coherente de servicios) en una relación puede-usar unidireccional con cualquier otra.
Elementos	• Capa: la descripción de una capa debería definir qué módulos contiene la capa
Relaciones	• **puede-usar**, que es una especialización de una relación de dependencia genérica **depende-de**. El diseño debería definir las reglas de uso de la capa (por ejemplo, "una capa puede usar cualquier capa inferior") y cualquier excepción posible.
Restricciones	• Cada pieza de software es asignada a sólo una capa. • Existen al menos dos capas (generalmente tres o más) • Las relaciones puede-usar no deben ser circulares (una capa inferior no puede usar una capa superior)
Para qué sirve	• Promover la modificabilidad y la portabilidad • Controlar la complejidad y facilitar la comunicación de la estructura del código a los desarrolladores. • Promover el reúso • Enfrentar la separación de incumbencias

Tabla A-6 Resumen del Estilo de Capas

Los elementos de una vista de capas son las capas, que son colecciones coherentes de módulos, cada uno de los cuales puede ser invocado o accesado. Los módulos en una capa pueden ser cualquier cosa; desde módulos que implementan servicios web a subrutinas en lenguaje ensamblador para compartir datos. Un requerimiento es que los módulos tienen una interface con la cual pueden ser llamados o accesados.

La relación entre capas es **puede-usar**. Para dos capas que tienen esta relación, cualquier módulo en la primera capa puede usar a cualquier módulo en la segunda capa. Se dice que el Módulo A usa el Módulo B si la ejecución correcta de A depende de que B esté correcto y presente.

Las capas tienen las siguientes propiedades que deberían ser documentadas en el catálogo que acompaña al diagrama de capas.

- Contenido. La descripción de una capa debería entregar guías de qué módulos deberían estar en una capa y como implementarlos. También puede listar explícitamente los módulos de software que contiene cada capa. Cada módulo debería estar asignado a solo una capa. Las capas generalmente tienen identificadores descriptivos pero vagos, tales como "Capa de Comunicación de Red" o "Capa de Reglas de Negocio", por lo que se necesita que identifique el contenido completo de la capa.
- Las capas de software que se permiten usar. ¿sólo se permite usar una capa inferior, cualquier capa inferior, o alguna otra? Si se segmenta una capa horizontalmente, ¿está permitido que los módulos de un segmento usen módulos de otro segmento que existe en la misma capa? Esta parte de la documentación debe explicar las excepciones, si existen, al uso de las reglas implícitas en la geometría.

Se debe documentar la justificación de la elección de cada capa. Explicar cómo cada capa entrega un conjunto coherente de responsabilidades. Estas descripciones ayudarán a asignar módulos futuros a la capa correspondiente. Por ejemplo, para ubicar un módulo en una capa pueden realizarse los siguientes cuestionamientos: Si se supone que el módulo P1 puede usar el módulo P2. ¿El módulo P2 debería estar en una capa inferior al módulo P1, o deberían estar en la misma capa?
La elección de una capa no es sólo una función de quién usa qué, son el resultado de una decisión de diseño consciente que ubica los módulos en capas, basados en consideraciones como cohesión y la naturaleza de los cambios probables. En general, P1 y P2 deberían estar en la misma capa si existe la probabilidad que sean portables a otra aplicación unidos o si juntos proporcionan diferentes aspectos de una máquina virtual a una comunidad de usuarios. La anterior es una definición operacional de cohesión, esta explicación también sirve como una guía de portabilidad, describiendo los cambios que pueden realizarse a cada capa sin afectar a otras.

¿Para qué es este estilo?

Las capas apoyan los atributos de calidad de modificabilidad y portabilidad a los sistemas de software. Una capa es la aplicación del principio de ocultamiento de información, que en teoría permitiría que un cambio en una capa inferior pueda ser ocultado detrás de sus interfaces y no impactará las capas superiores. Como en toda teoría, existen ventajas y desventajas que la acompañan:

- La ventaja es que esta técnica ha sido utilizada con gran éxito para ayudar a la portabilidad. Las maquinas, sistema operativo, u otras dependencias de plataforma se esconden en una capa; siempre y cuando la interfaz de la capa no cambie, es posible intercambiar partes o tecnologías específicas, y los niveles superiores que dependen sólo de la interfaz trabajarán normalmente.
- La desventaja es que la interfaz significa mucho más que una interfaz de programación de aplicación (API). Una interfaz incorpora todas las suposiciones que una entidad externa –en este caso una capa- pueden hacer. Los cambios en una capa inferior pueden afectar, por ejemplo, el rendimiento se escapará a través de su interfaz y puede afectar una capa superior.

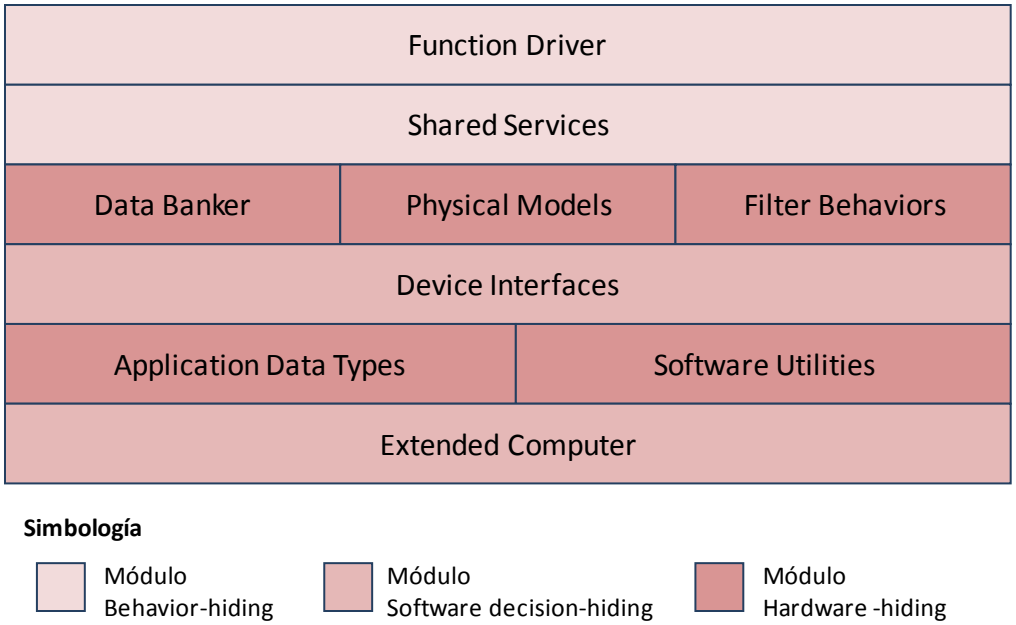
Un error común es que las capas introducen una sobrecarga de ejecución adicional. Aunque esto puede ser cierto en algunas implementaciones, las facilidades de compilación, ensamblado y carga pueden reducir esta sobrecarga adicional.
En algunos casos, una capa puede contener servicios no utilizados, estos servicios pueden consumir recursos en tiempo de ejecución innecesarios, como la memoria utilizada por el código no utilizado o un hilo de ejecución nunca ejecutado. Si estos recursos son escasos, será muy útil tener una capacidad de compilación, ensamblado y carga que tenga la facilidad de eliminar el código no utilizado.
Las capas son una parte del rol modelador que juega la arquitectura para la construcción de sistemas. Conociendo las capas donde reside el software, los desarrolladores saben cuál es el código con que pueden contar. Las capas pueden definir la asignación de trabajo para los equipos de desarrollo, aunque no siempre.

Las capas son una parte del rol comunicador que tiene la arquitectura. En grandes sistemas, el número de módulos y las dependencias entre ellos se expande rápidamente. El organizar los módulos en capas con interfaces es una herramienta importante para controlar la complejidad y comunicar la estructura de un sistema a los desarrolladores.
Agrupar los módulos en capas que tienen la misma abstracción tecnológica o estar cohesionado con respecto a sus responsabilidades ayuda a asignar el trabajo de implementación a los equipos especializados. Por ejemplo, los módulos en una capa de presentación pueden ser asignados a los desarrolladores con conocimientos de interfaces de usuario.
Las capas apoyan el rol de análisis que desempeña la arquitectura. Ello permite realizar análisis de impacto de los cambios a un diseño, permitiendo un grado de determinación del alcance de los cambios.
Existen capas que entregan una máquina virtual que promueve la portabilidad. Por esta razón, es importante controlar las interfaces para asegurar que se considera la portabilidad. Las interfaces no deberían exponer funciones que dependen de una plataforma particular; esas funciones deberían esconderse tras interfaces más abstractas que sean independientes de la plataforma.
Debido a que la relación de orden entre las capas debe realizarse con una implementación **puede-usar** la capa inferior, se dispone de pocas facilidades para realizarla. Es decir la visión de mundo de las capas inferiores tiende a ser más pequeña y más dependiente de la plataforma. Las capas inferiores tienden a estar construidas usando el conocimiento del sistema operativo, canales de comunicación, bases de datos y elementos similares. Aquellas capas específicas de la plataforma son en gran parte independientes de la aplicación particular que corre en ellas; esto hace que la aplicación sea fácilmente portable a una plataforma diferente.

Relación con otros estilos

Los diagramas de capas a menudo son confundidos con otros estilos de arquitectura cuando se introduce información ortogonal a la relación **puede-usar** sin una decisión consciente.

- Descomposición de Módulos: Las capas en vistas de capas y los módulos en vistas de descomposición están siempre relacionadas pero casi nunca existe una relación uno a uno entre ellas. Una capa puede componerse de más de un módulo. Dos submódulos de un módulo pueden ser parte de diferentes capas. En muchos casos se debe entregar un mapeo entre capas y módulos en la vista de descomposición. Si un módulo aparece en más de una capa, esto se puede indicar utilizando colores o patrones de relleno, como se muestra en la Figura A-15.



El software en una capa puede-usar software de la misma capa o de cualquier capa inferior.

Figura A-15 Un diagrama mostrando capas y módulos en una vista de descomposición

En el ejemplo de la Figura A-15, el mapeo entre capas y módulos no es uno a uno. En esta arquitectura, el criterio para particionar en módulos fue la encapsulación de los posibles cambios. El sombreado de los elementos muestra la descomposición del sistema en módulos generales; por ejemplo, Function Driver y Shared Servces son submódulos del módulo Behavior-hiding. Por lo tanto, en este sistema, las capas corresponden a partes de módulos de mayor nivel. Además, es fácil imaginar los casos en el cual los módulos son parte de una capa.

- Niveles. Las capas generalmente son confundidas con niveles en arquitecturas multi-nivel. Las capas no son niveles. El estilo de capas muestra agrupaciones de unidades de implementación y, por lo tanto, es un tipo de estilo de módulos. El estilo multi-nivel es un estilo de componentes y conectores porque los niveles agrupan componentes de tiempo de ejecución.
- Estilo de Usos. Debido a que las capas expresan una relación puede-usar, existe una estrecha correspondencia con el estilo de usos. Por supuesto que ninguna relación de usos permite violar una relación puede-usar. Si un desarrollo incremental o en espiral de subconjuntos es el objetivo, el arquitecto comenzará con una especificación puede-usar. Esta especificación será la guía para el diseño de las relaciones de uso de cualquier subconjunto de interés.

Notaciones

Notaciones Informales

Pila de Cajas

Las capas casi siempre son dibujadas como una pila de cajas. La relación puede-usar se denota por la geometría adyacente y es leída de arriba hacia abajo, como en la Figura A-16 (nótese que la simbología podría haber dicho “Una capa tiene permitido usar cualquier capa inferior”).

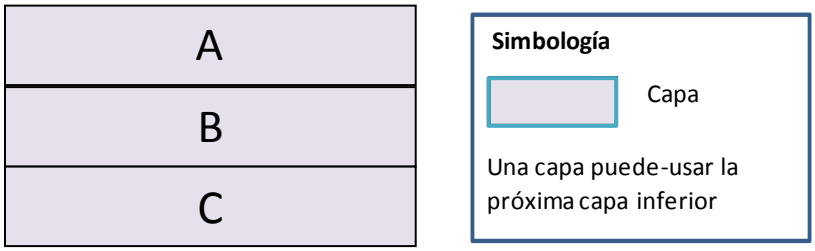


Figura A-16 Notación de Cajas apiladas para diseños de capas

Los estilos de capas por lo tanto son uno de los pocos en los cuales se muestra la conexión entre los componentes por adyacencia geométrica y no una simbología explícita, como por ejemplo una flecha, aunque está permitido utilizar flechas como se muestra en la Figura A-17.

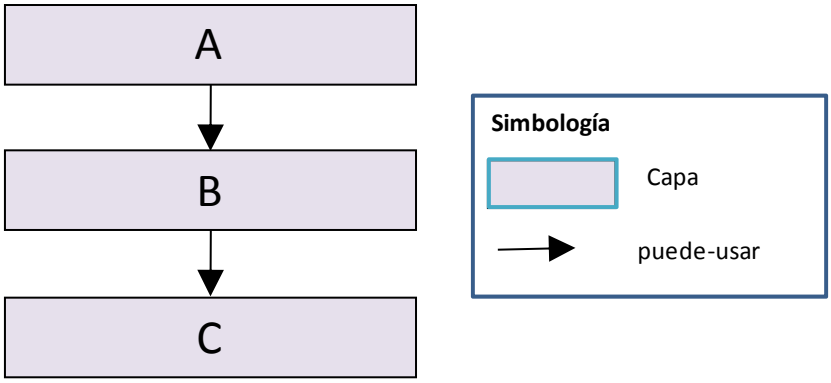


Figura A-17 Diseño de Capas con la relación puede-usar mostrada con flechas

Capas Segmentadas

A veces las capas son divididas en segmentos para indicar una agregación de módulos más fina. Generalmente, esto ocurre cuando un conjunto de unidades pre-existente, como módulos importados, comparten la misma relación puede-usar. Cuando ocurre esto, el creador del diagrama debe especificar cuáles son las reglas de uso que afectarán a cada segmento. Es posible utilizar muchas reglas, pero debe ser de forma explícita. En la Figura A-18, las capas superior e inferior están segmentadas. Los segmentos de la capa superior no tienen permitido el uso de otros segmentos, pero si en los segmentos de la capa inferior. Si se dibuja el mismo diagrama sin las flechas, será más difícil diferenciar las reglas de uso entre los segmentos incluidos en las capas. Los diagramas de capas son generalmente origen de ambigüedad debido a que no hacen explícitas las relaciones puede-usar.

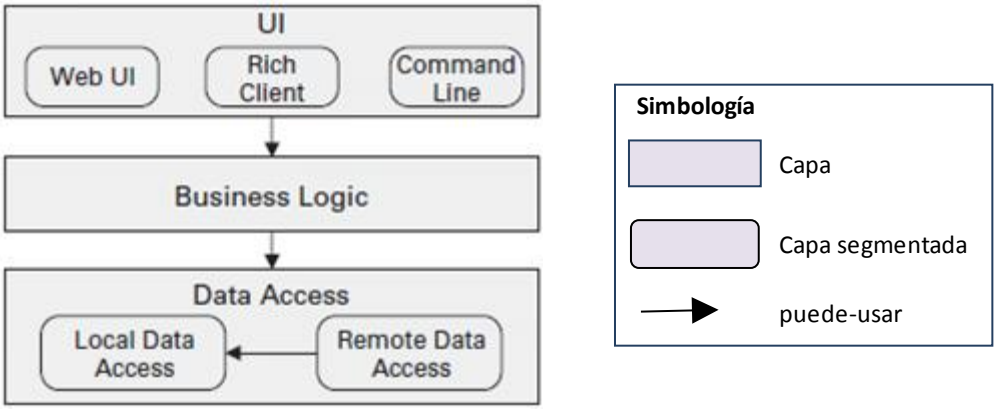


Figura A-18 Diseño de Capas con capas segmentadas

Anillos

Una variación notacional es mostrar las capas como un conjunto concéntrico circular, o anillos. Donde el anillo más interno corresponde a la capa inferior, teniendo el mismo significado que su correspondiente capa si fuera segmentada.

No existe diferencia semántica entre un diagrama que usa una pila de rectángulos y una que usa el paradigma de anillos, excepto cuando las capas segmentadas tienen restricciones en la relación puede-usar dentro de una capa. En la Figura A-19, se asume que a los segmentos del anillo se les permite usar el anillo que tocan. No es posible desarmar el diagrama de anillo para producir un diagrama de pilas (como el mostrado a su derecha) con exactamente el mismo significado, debido a que los arreglos circulares permiten más adyacencias que utilizando arreglos lineales. (En el diagrama de capas, B1 y B3 están separados; mientras que en el diagrama de anillos están adyacentes). Los casos como este son los únicos en el cual el diagrama de anillo puede mostrar una adyacencia geométrica que los diagramas de pilas no pueden.

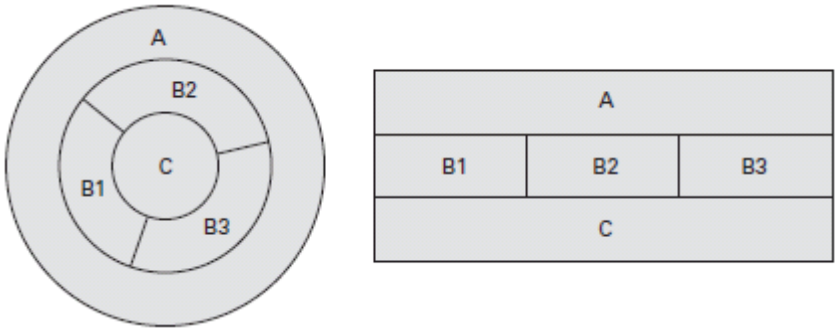


Figura A-19 Diseño de Capas mostrado como anillos concéntricos y como pila de cajas

Capas con un Sidecar

Un sidecar es un carro que se acopla a un lado de una moto. Para el caso de una arquitectura de capas, con un sidecar se trata de una caja que se ubica lateralmente a la pila de cajas. Muchas arquitecturas que son descritas utilizando estas vistas lucen como en la Figura A-20. Este tipo de notación puede significar dos cosas:

- Los módulos en D pueden usar módulos en A, B, o C
- Los módulos de A, B, o C pueden usar módulos en D

Técnicamente, el diagrama podría indicar que ambas afirmaciones son correctas, además esto podría significar que es una arquitectura de capas pobre. Por ello, el creador del diagrama debe especificar cuáles son sus reglas de uso. Una variación como esta tiene sentido sólo para reglas de uso de un solo nivel en el stack principal, por ejemplo, cuando A sólo puede usar B y nada más. Por otra parte, D podría simplemente ser la capa inferior de la pila principal, y el sidecar sería innecesario.

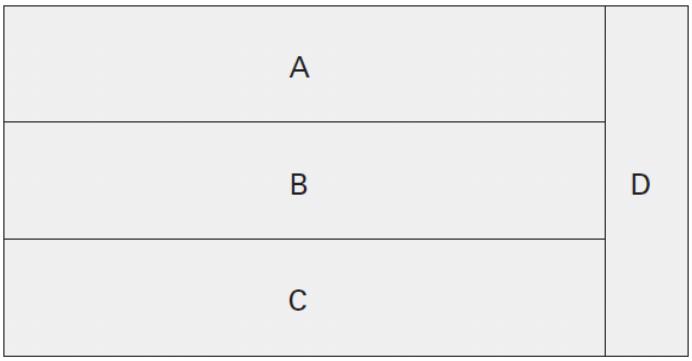


Figura A-20 Diseño de Capas con Slidecar

En algunos casos, la arquitectura de capas es descrita como una figura de tres dimensiones, para representar una capa que es accesible a todas las otras capas, esto se muestra en la Figura A-21.

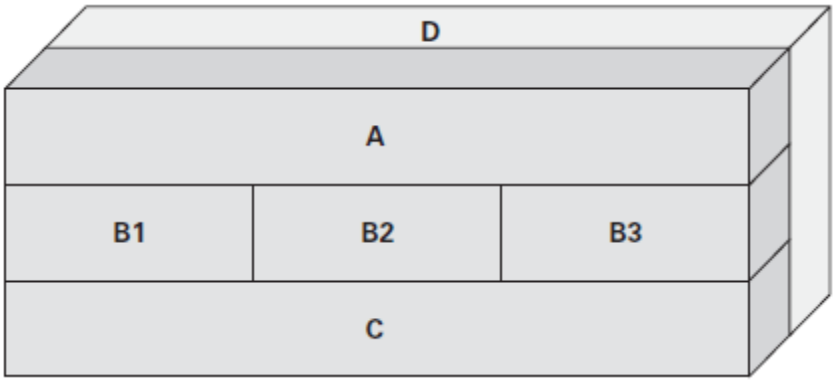


Figura A-21 Diagrama de capas tridimensional

Las capas laterales representan generalmente librerías de utilidades o servicios de plataforma (como el sistema operativo o el ambiente de ejecución).

Tamaño y Color

A veces las capas son coloreadas para indicar sus equipos responsables u otra característica distinguible. Otras veces se utilizan diferentes colores sólo para mejorar la lectura del diagrama. El tamaño a veces es utilizado para dar ideas del tamaño relativo de los módulos que constituyen cada capa. Si esto tiene algún significado, se debe explicar el tamaño y color en la simbología del diagrama.

Notaciones Semiformales

UML

En UML no existe una primitiva específica para identificar una capa. Sin embargo, se pueden representar capas en UML utilizando paquetes con estereotipos, como se muestra en la Figura A-22. Un paquete es un mecanismo de propósito general para organizar elementos en grupos, y esto da la sensación de capas. La relación puede-usar puede ser una dependencia estereotipada entre los paquetes de capas.

Las dependencias de acceso no son transitivas. Si un paquete 1 puede accesar a los módulos de un paquete 2 y el paquete 2 puede accesar al paquete 3, no se entiende que automáticamente el paquete 1 puede accesar al paquete 3.

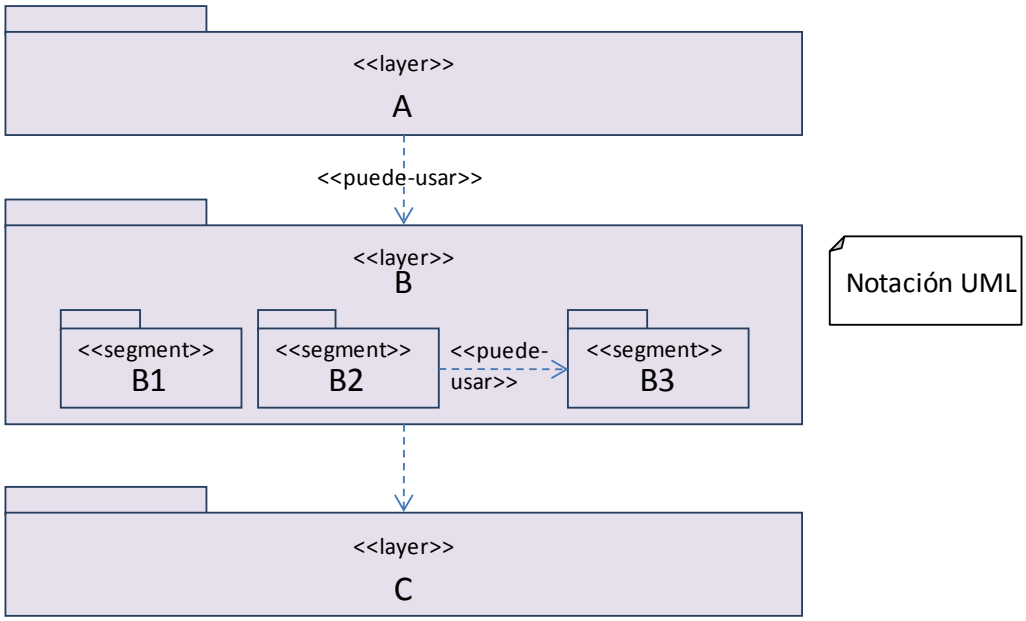

<<layer>>
A
<<puede-usar>>
<<layer>>
B
<>
B1
<>
B2
<<puede-
usar>>
<>
B3
Notación UML
<<layer>>
C

Figura A-22 Documentación de Capas Segmentadas en UML

1.1.2.5. Estilo de Aspectos

El estilo de aspectos es un estilo de módulos usado para aislar los módulos responsables de funcionalidades transversales en la arquitectura.

En general, cuando se implementan módulos de software, el código de la lógica de negocios se termina mezclando con el código de funciones transversales al sistema. Por ejemplo, si se está escribiendo el código para un sistema de automatización bancaria, puede haber módulos como el de contabilidad, clientes y ATM. El módulo de contabilidad idealmente contendrá sólo el código para controlar la lógica de negocios de la contabilidad bancaria (abrir/cerrar cuentas, depósitos, giros, transferencias, y otros). Pero en la práctica se debe agregar código para controlar las funciones transversales, como el control de acceso, administración de transacciones, y el log de eventos.

El estilo de aspectos establece que los módulos responsables de las funcionalidades transversales deberían ubicarse en una o más vistas de aspectos. Estos módulos son llamados aspectos, y están basados en la terminología empleada en la programación orientada a aspectos (AOP, de sus siglas en inglés Aspect-Oriented Programming). Las vistas de aspectos deben contener información para enlazar cada módulo de aspecto a los otros módulos que requieren la funcionalidad transversal.

El estilo de aspectos es útil cuando se planea utilizar AOP en la implementación. Aunque, también es aplicable cuando una funcionalidad transversal es implementada de forma tradicional a través de la herencia de clases e interfaces, inserción de macros, inyección de dependencias, librerías de utilidad, y otras alternativas. La meta del diseño e implementación de funcionalidades transversales en módulos de aspectos separados es mejorar la modificabilidad de los módulos que compiten con las funcionalidades del dominio del negocio.

Elementos, Relaciones y Propiedades

La Tabla A-7 resume las características del estilo de aspectos. Los elementos en este estilo son los módulos, estos contienen código que afecta a otros módulos del sistema.

Las relaciones encontradas en el estilo de aspectos son generalmente llamadas **es-transversal-a**. Un aspecto es transversal a un módulo si el aspecto contiene funcionalidades transversales que afectan al módulo. Un aspecto puede contener las mismas propiedades que un módulo regular. Además, puede contener una propiedad que describe cuales son los módulos de destino que serán afectados por el aspecto; en términos de AOP, a esta propiedad se le llama especificación de pointcut.

Resumen	El estilo de aspectos muestra módulos de aspecto que implementan funcionalidades transversales y cómo estos deben ser enlazados a otros módulos en el sistema
Elementos	• **Aspecto**. Es un módulo especializado que contiene la implementación de una funcionalidad transversal a uno o más módulos en el sistema
Relaciones	• **es-transversal-a**. Son las que enlazan un módulo de aspecto a un módulo que será afectado por la lógica transversal del aspecto.
Restricciones	• Un aspecto puede atravesar uno o más módulos regulares y también módulos de aspecto. • Un aspecto que es transversal a sí mismo puede causar una recursión infinita, dependiendo de su implementación.
Para qué sirve	• Para modelar funcionalidades transversales en diseños orientados a objeto. • Mejorar la modificabilidad.

Tabla A-7 Resumen del Estilo de Aspectos

¿Para qué es este estilo?

El estilo de aspectos puede ser utilizado para modelar la implementación de funcionalidades transversales. Promueve la modificabilidad mediante un aumento en la modularidad y evita la mezcla entre las funcionalidades transversales y la lógica del dominio del negocio.

Relación con otros estilos

En general, los aspectos muestran herencia. Los estilos de aspectos pueden combinarse con el estilo de generalización cuando se necesita mostrar herencia de aspectos.

Notaciones

UML

Aunque UML no tiene símbolos especiales para aspectos, es una notación común para mostrar vistas de aspectos. En UML los módulos de aspectos son generalmente representados como clases

estereotipadas en diagramas de clases, como muestra la Figura A-23. Especialmente, cuando la plataforma de implementación permite AOP, el mostrar módulos de aspectos como clases estereotipadas da más sentido debido a que los aspectos son estructuralmente similares a las clases: ellos pueden contener atributos y operaciones, y pueden extender otro aspecto en una relación de herencia.

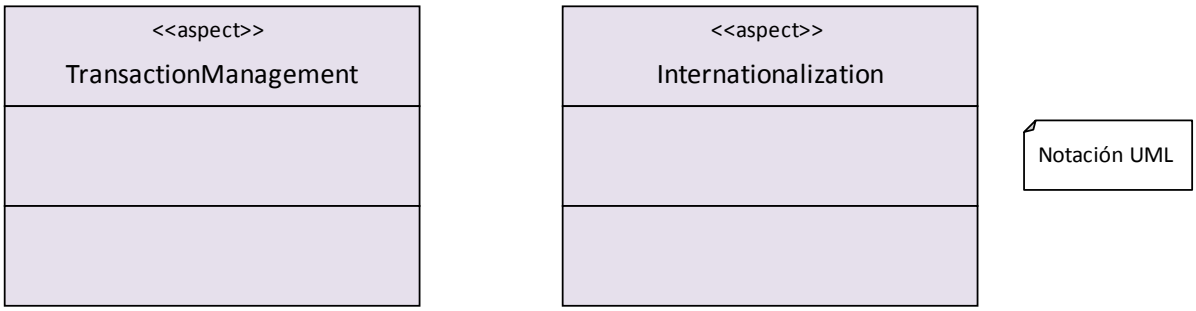


Figura A-23 En UML, los aspectos son generalmente representados como clases estereotipadas con <<aspect>>

La relación **es-transversal-a** puede ser representada como una dependencia estereotipada desde el aspecto a cada uno de los módulos en los cuales es transversal. Aunque esta alternativa no es escalable, por definición un aspecto entrega una funcionalidad transversal, y por lo tanto puede ser transversal a muchos módulos. El trazar una línea transversal entre el módulo de aspecto y cada uno de los módulos no es práctico en sistemas no triviales y sería saturar los diagramas. Una mejor alternativa es simplemente omitir de los diagramas la relación transversal. A cambio, para caracterizar los módulos que un aspecto atraviesa, se puede agregar un comentario al módulo de aspecto (en lenguaje natural o en una sintaxis formal) como lo muestra la Figura A-24.

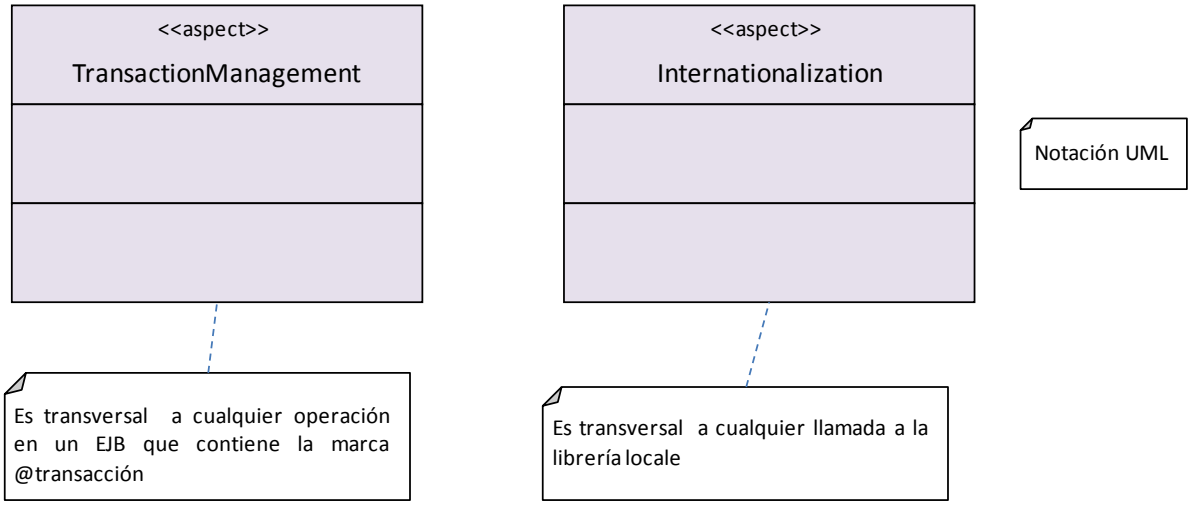


Figura A-24 En vez de trazar una línea a cada módulo donde el aspecto es transversal se agrega un comentario

1.1.2.6. Estilo de Modelo de Datos

El modelamiento de datos es una actividad común en el proceso de desarrollo de sistemas informáticos. La salida de esta actividad es el modelo de datos, el cual describe la estructura

estática de la información en términos de entidades de datos y sus relaciones. Por ejemplo, en un sistema bancario, las entidades generalmente representan Cuentas, Clientes y Depósitos. Una Cuenta tiene muchos atributos, como por ejemplo número de cuenta, tipo (corriente o ahorro), estado y saldo actual. Una relación puede indicar que un cliente puede tener una o más cuentas, y una cuenta está asociada a uno o dos clientes. El modelo de datos es generalmente representado gráficamente en diagramas Entidad-Relación (DERs) o en diagramas de clases de UML.

El primer borrador de una vista de arquitectura generalmente tiene poco nivel de detalle. Con el tiempo, se toman decisiones de diseño, la vista se detalla cada vez más hasta que el arquitecto considera que hay suficiente información capturada para esa vista. Lo mismo ocurre con el modelo de datos, este se extiende por la evolución del modelo de alto nivel que muestra las entidades de datos en un dominio de negocios específico hasta un modelo que muestra los detalles de cómo se almacenan los datos, por ejemplo, en un sistema de administración de bases de datos relacional. Como resultado, diferentes organizaciones enfocan sus esfuerzos de modelamiento y documentación en diferentes ámbitos de la evolución del modelo de datos. Así, las organizaciones a veces utilizan calificativos para el modelo de datos para distinguir esas etapas. Ejemplos de estos calificativos son:

- Conceptual: El modelo conceptual abstrae los detalles de implementación y se concentra en las entidades y las relaciones percibidas en el dominio del problema. La Figura A-25 muestra un fragmento de un diagrama de datos conceptual.
- Lógico: El modelo de datos lógico es una evolución del modelo de datos conceptual hacia la tecnología de administración de datos (como las bases de datos relacionales). La Figura A-26 muestra un ejemplo de modelo de datos lógico.
- Físico: El modelo de datos físico está relacionado con la implementación de las entidades de datos. Incluye optimizaciones que pueden considerar particionamiento o fusión de entidades, duplicación de datos, y la creación de claves de identificación e índices.

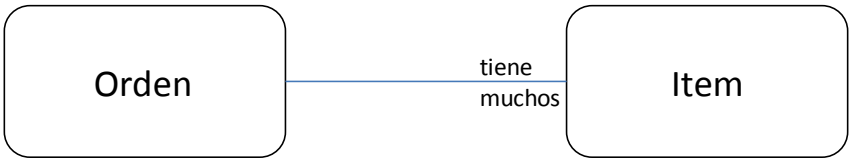


Figura A-25 Ejemplo de modelo de datos conceptual

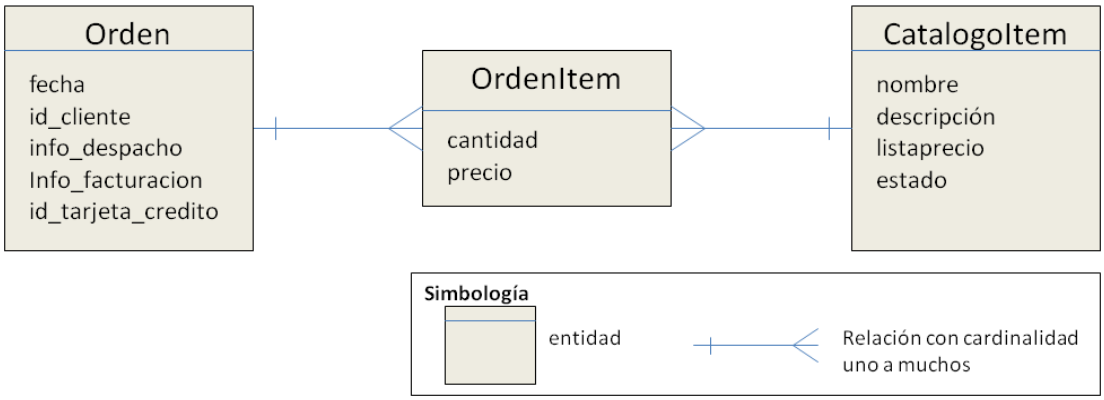


Figura A-26 Ejemplo de modelo de datos lógico

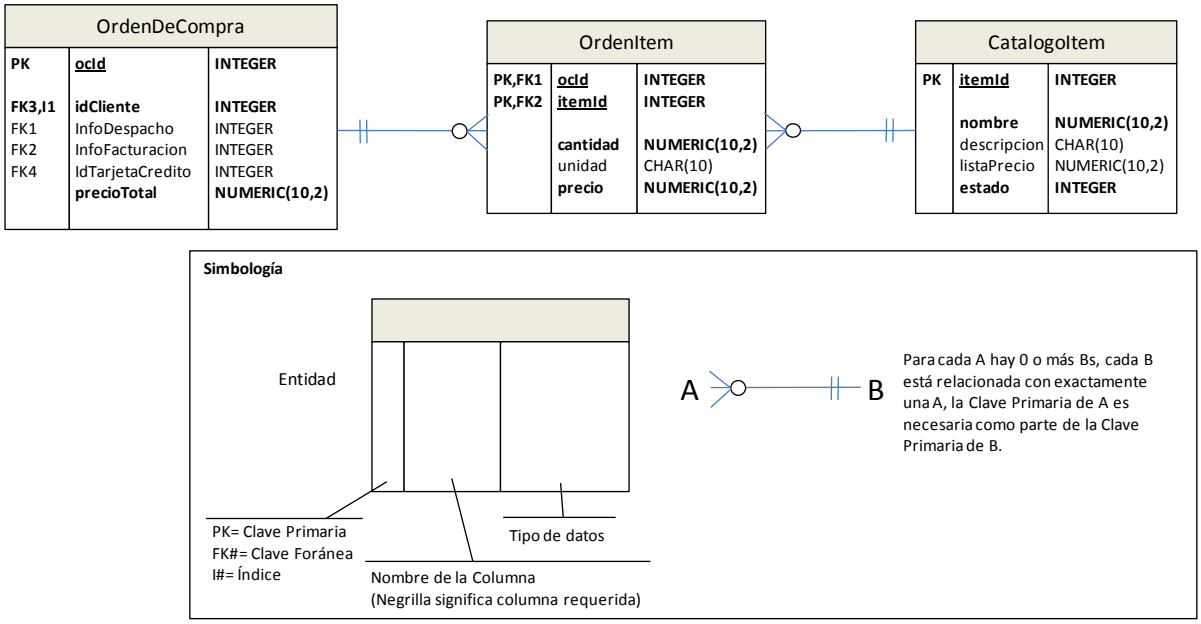


Figura A-27 Modelo de datos físico creado agregando detalles de implementación y optimizaciones del modelo de datos lógico de la Figura A-26

En escenarios iniciales, la documentación de la arquitectura puede contener el modelo de datos con las entidades principales y las relaciones importantes. Luego, el modelo inicial es reemplazado por el modelo detallado aprobado por los administradores de datos.

Elementos, Relaciones y Propiedades

En la Tabla A-8 se resumen las características del estilo de modelo de datos. Los elementos en el modelo de datos son llamados entidades de datos o simplemente entidades. Cualquier objeto distinguible que contiene información que va a ser almacenada o representada en un sistema puede ser una entidad.

Resumen	El modelo de datos describe la estructura de las entidades de datos y sus relaciones.
Elementos	Entidad de Datos, es un objeto que almacena información que necesita ser almacenada o de alguna forma representada en el sistema. Sus propiedades incluyen nombre, atributos de datos, clave primaria, y reglas para entregar permisos a usuarios para acceder a la entidad.
Relaciones	• **uno-a-uno**, **uno-a-muchos**, **muchos-a-muchos**, son asociaciones lógicas entre entidades. • Generalización/especialización, indican una relación **es-un** entre entidades. • Agregación, convierte una relación en una entidad.
Topología	• Las dependencias funcionales deben ser evitadas
Para qué sirve	• Describir la estructura de los datos usados en el sistema • Realizar análisis de impacto de los cambios a un modelo de datos • Reforzar la calidad de datos evitando redundancias e inconsistencias • Guiar en la implementación de módulos que acceden a los datos

Tabla A-8 Resumen del estilo de modelo de datos

Las propiedades de una entidad pueden ser:

- Nombre de la entidad
- Descripción del significado y la importancia de la entidad
- Listado de los atributos de datos de la entidad. Por ejemplo, una entidad Auto puede tener atributos como año de fabricación, fabricante, modelo, kilometraje, precio y permiso. Cada atributo puede tener propiedades, como tipo de datos, tamaño, y si es un atributo obligatorio o no
- El atributo (o atributos) usado para identificar una entidad única
- Si una entidad es débil. Una entidad débil, también conocida como una entidad dependiente, depende de la existencia de otra entidad para existir. Por ejemplo, una OrdenItem necesita la existencia de una OrdenDeCompra en la Figura A-27.
- Las restricciones e invariantes de los valores de atributos individuales o combinados. Por ejemplo, “La fecha de regreso no puede ser previa a la fecha de llegada”.
- Reglas que serán usadas para entregar permisos a usuarios o grupos de usuarios que verán una entidad.
- Número esperado de la ocurrencia de una entidad y su tasa de crecimiento.

Otras propiedades relacionadas al modelo de datos físico y que son específicas para la implementación del modelo de datos en la plataforma de destino. Por ejemplo:

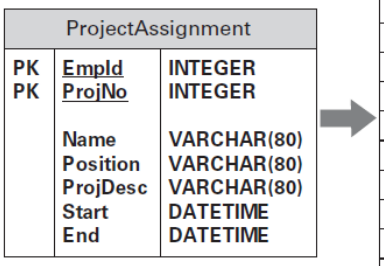
- Listado de los atributos que deberían estar indexados para optimizar los tiempos de acceso
- Listado de los atributos que deberían estar encriptados o comprimidos
- Si la entidad debería llegar a ser una vista de base de datos en vez de una tabla. Una vista es una tabla virtual que es definida por una consulta sql sobre una o más tablas.

- Si la entidad debería llegar a ser una vista materializada, lo que significa que será implementada como una tabla de base de datos que almacenará un conjunto de datos que serán copiados de una tabla maestra. Como una vista regular, el subconjunto estará definido por un comando sql.
- Listado de triggers que serán implementados para la tabla. Un trigger es un procedimiento especial que es ejecutado automáticamente por el sistema de administración de la base de datos cuando se insertan, actualizan, o eliminan datos.

Existen tres tipos de relaciones que se encuentran en los modelos de datos:

- Relacionamiento. Usado para indicar una asociación lógica entre entidades. Por lo general, contiene la cardinalidad de las entidades participantes: uno-a-uno, uno-a-muchos, o muchos-a-muchos. Además, una relación se puede ser identificada o no-identificada. Una relación es identificada cuando los elementos de una tabla no pueden existir por si solos (existe una dependencia). Una relación identificación desde A hacia B significa que la existencia de B depende de la existencia de A. En cambio en una relación no-identificada no existe una dependencia para que el elemento pueda existir.
- Generalización/Especialización. Indica una relación es-un entre entidades. Por ejemplo, la entidad seguro es una generalización de diferentes tipos de seguros; al mismo tiempo, las entidades seguro de auto y seguro de casa son especialización de la entidad seguro.
- Agregación. Es una abstracción que convierte una relación entre entidades en una entidad agregada. Por ejemplo, una relación entre paciente, un físico, y una fecha pueden ser abstraídas como una agregación de entidades llamada cita. En la práctica, esta relación raramente se usa.

Conceptualmente, no existen restricciones topológicas con respecto a las relaciones en el modelo de datos. Sin embargo, la técnica de normalización impone restricciones sobre el modelo de datos basándose en las dependencias entre los atributos de entidades. La normalización es utilizada por los administradores de datos para evitar la duplicación de información, con el fin de velar por la consistencia (integridad) de los datos. La Figura A-28 muestra una tabla antes de su normalización y la Figura A-29 muestra el modelo de datos después de la normalización.



EmpId	Name	Position	ProjNo	ProjDesc	Start	End
100	Simpson	Analyst	23	DB design	Apr-02	Jul-02
140	Beeton	Technician	14	Network cabling	Sep-02	Oct-02
160	Davis	Technician	14	Network cabling	Sep-02	Nov-02
			36	Network testing	Nov-02	Dec-02
190	Berger	DBA	45	Physical design	Aug-02	Nov-02
			48	Space allocation	Nov-02	Dec-02
100	Simpson	Analyst	25	Reports	Oct-02	Nov-02
110	Covino	Analyst	31	Forms	Mar-02	May-02
			25	Reports	May-02	Jul-02
120	Brown	Analyst	11	Order entry	Jul-02	Sep-02
180	Smith	Programmer	31	Forms	Sep-02	Nov-02
			25	Reports	May-02	Jul-02
200	Rogers	Programmer	11	Order entry	Sep-02	Oct-02
			12	Inventory control	Oct-02	Dec-02
			13	Invoicing	Nov-02	Dec-02
100	Simpson	Analyst	31	Forms	Aug-02	Oct-02
130	Clemens	Analyst	23	DB design	Apr-02	Jun-02

Figura A-28 Entidad ProjectAssignment antes de una normalización

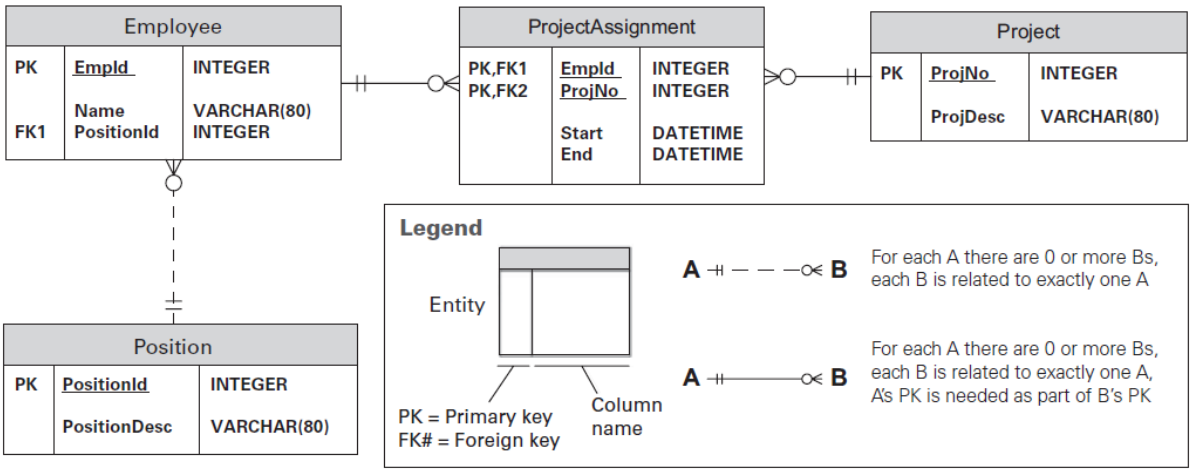


Figura A-29 Modelo de datos para ProjectAssignment después de una Normalización.

¿Para qué es este estilo?

El modelo de datos facilita la comunicación con los stakeholders durante el análisis del dominio y la obtención de requerimientos. Pero sobre todo, el modelo de datos es el esquema para la implementación de las entidades de datos, por ejemplo, en una base de datos relacional.

Un modelo de datos creado haciendo un análisis minucioso ayuda a cumplir con los requisitos de rendimiento de un sistema. En Aplicaciones centradas en datos, el acceso a datos generalmente representa una gran cantidad de tiempo para procesar las solicitudes de los usuarios. Los arquitectos y los administradores de datos deberían entender qué clases de operaciones de acceso a datos serán más críticas para el sistema y cuáles son sus requerimientos de performance. Para mejorar el rendimiento del sistema, se utilizan los requerimientos, desnormalizaciones, optimizaciones, y otras decisiones de diseño sobre el modelo de datos. Algunos ejemplos de esas decisiones de diseño son:

- La fusión de dos entidades para evitar utilizar OUTER JOIN que son costosos u operaciones de unión en una consulta.
- Incluir atributos derivados para evitar realizar FULL SCAN sobre una tabla para obtener el valor derivado.
- Crear índices sobre atributos que son parámetros utilizados en consultas.
- Cambiar la granularidad (a nivel de registro o página) y el tipo (a optimista) de bloqueos sobre ciertas entidades para evitar contención y deadlocks.

Después de que se implementa un sistema, incluso cuando el modelo de datos se crea cuidadosamente, es posible encontrar problemas de performance en las operaciones de acceso a datos. Para corregirlos, se utiliza el modelo de datos para realizar tareas de optimización de consultas.

En los sistemas de información, el modelo de datos es una herramienta esencial para realizar análisis de modificabilidad. Para analizar el impacto de las modificaciones requeridas de un sistema no se puede mirar solamente la estructura del código. Muchas modificaciones necesitan alterar el modelo de datos y por lo tanto su implementación física. Las modificaciones al modelo de datos pueden ser muy costosas, cuando provocan la modificación del código de muchas aplicaciones que comparten los mismos datos. Un simple cambio como el hacer que un atributo sea mandatorio (por ejemplo, solicitar la fecha de nacimiento de un cliente) puede requerir cambios a todas las pantallas y funciones que permiten crear o actualizar esa información. El control de versiones y la redistribución de aplicaciones son más complicados cuando se trata de cambios en el modelo de datos. Por otra parte, las grandes modificaciones del modelo de datos, como la fusión de un modelo de datos de un sistema legado, puede requerir la implementación de operaciones de extracción, transformación y carga (ETL) para convertir los datos. De hecho, el modelo de datos es un aporte importante a los proyectos de data warehouse y la integración de esquemas de datos requerida por sociedades entre empresas (por ejemplo, una aerolínea necesita compartir datos con una compañía de renta de automóviles).

El modelo de datos es una vista de arquitectura que, idealmente, debe ser creada con un conocimiento profundo de los planes de desarrollo adicionales, extensiones profundas, y la integración con otros sistemas de información. Los datos son un activo valioso, y la evidencia de un modelo de datos empresarial y un grupo de administración de datos ayuda a reforzar la integridad de datos. Si un nuevo sistema necesita obtener información de ventas, el modelo de datos empresarial puede contener esa información. El arquitecto del nuevo sistema puede no estar consciente de las entidades de datos que contienen la información de las ventas, pero el administrador de datos debería y puede indicar las entidades en vez de crear nuevas en la base de datos. Los datos dispares y redundantes contribuyen a una calidad deficiente de los datos.

Existen herramientas que basadas en un modelo de datos que pueden generar scripts para crear una base de datos física. Algunas herramientas pueden generar código de aplicación para acceder a las tablas de datos, clases para mantener los datos, formularios para que los usuarios finales ingresen los datos, esquemas de mensajes, y reportes simples.

Finalmente, el modelo de datos ayuda a los desarrolladores de aplicación a escribir código para acceder a los datos. Es más fácil entender un diagrama entidad relación que buscar sobre los comandos de creación de tablas o el diccionario del sistema de administración de base de datos.

Relación con otros estilos

Las entidades en el modelo de datos están intrínsecamente conectadas a algunos módulos en otras vistas de módulos, especialmente los módulos que contienen representaciones de datos en memoria. En sistemas orientados a objetos que utilizan bases de datos relacionales para almacenar sus datos, se pueden encontrar clases que corresponden a entidades persistentes. La relación no siempre es uno-a-uno debido a que el paradigma relacional es diferente al paradigma orientado a objetos. Este problema es conocido como la diferencia de impedancia objeto-relacional y se enfrenta con herramientas de mapeo objeto-relacional (ORM del inglés object-relational mapping) y frameworks como Hibernate para Java y LLBLGen para Microsoft .Net.

Los arquitectos pueden encontrar útil el indicar qué módulos (en la vista de módulos), qué componentes (en la vista de componentes y conectores), o incluso qué casos de uso de los requerimientos funcionales usan cuales entidades de datos. El mapeo del modelo de datos a otras vistas puede ser registrado en una tabla. Se puede indicar donde se crea, lee, actualiza, o eliminan (llamado también las siglas del inglés CRUD: Create, Read, Update, Delete) datos de cada entidad de datos. Este mapeo genérico puede ser representado en una matriz CRUD.

El modelo de datos describe la estructura de las entidades de datos y las relaciones que generalmente serán desplegadas en un componente de datos compartidos como una base de datos Oracle. Los almacenes de datos son comúnmente representados en una vista de arquitectura de datos compartida, junto con los componentes de ejecución que acceden a ellos. Además, la vista de distribución generalmente muestra donde está ubicados los almacenes de datos. Documentar el mapeo de las entidades de datos en un modelo de datos a diferentes almacenes de datos y sus respectivas máquinas es útil cuando la solución utiliza bases de datos distribuidas o replicadas.

Notaciones

El modelo de datos puede ser descrito gráficamente usando notaciones formales o semi-formales en las que se incluyen:

- Diagrama Entidad-Relación de Peter-Chen
- Diagrama Entidad-Relación de Crow's Foot
- IDEFIX
- Diagrama de Clases UML

Las primeras tres son variaciones de notaciones ERD, y la última es la alternativa UML a ERD. Las notaciones Crow's Foot y los diagramas de clase UML son las que más se utilizan en la industria y poseen más herramientas para representarlas.

Diagrama Entidad-Relación de Crow's Foot

Es una de las notaciones de diagramas entidad-relación más popular que usa líneas con símbolos especiales en cada extremo para indicar cardinalidad en las relaciones. Los símbolos incluyen un guion (para indicar uno), un círculo (para indicar cero), y una pata de gallo (indicando muchos). La notación entidad-relación de Crow'S Foot fue utilizada inicialmente en los años 80 por Richard Barker, también en el planteamiento de Ingeniería de Información desarrollado por James Martin y Clive Finkelstein en 1981. La Figura A-30 muestra un ejemplo de esta notación.

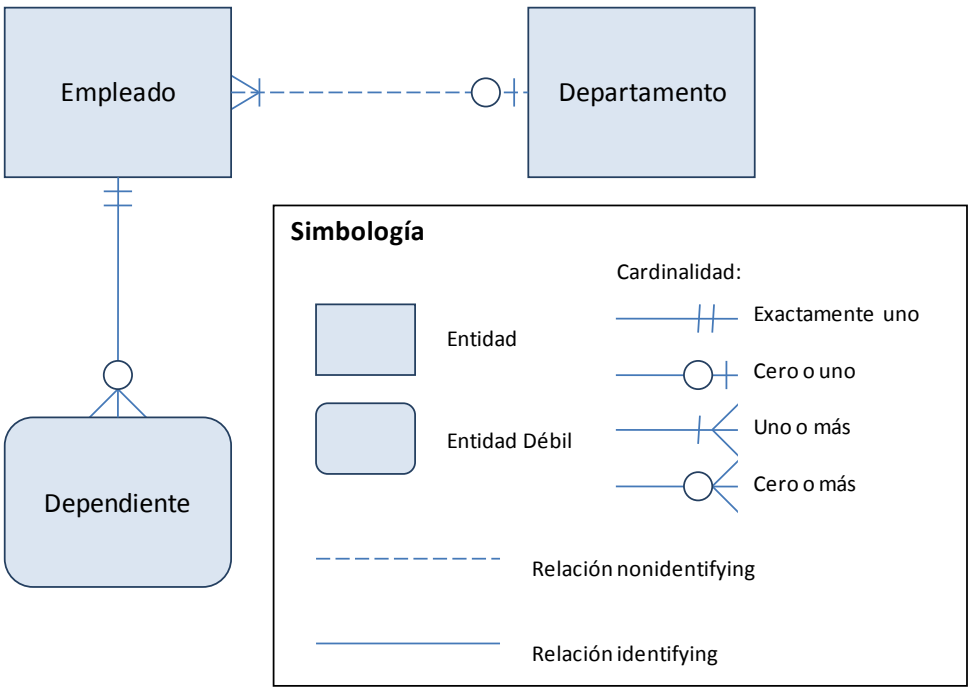


Figura A-30 Diagrama de modelo de datos simplificado de un sistema de recursos humanos utilizando notación Crow's Foot

UML

El modelo de datos puede ser representado como un diagrama de clases UML, donde las clases corresponden a entidades de datos. Aquí, el compartimiento de atributos lista los atributos de la entidad, y el compartimiento de operaciones aparece vacío. Las asociaciones UML representan relaciones entre entidades y los intervalos de multiplicidad se muestran a ambos extremos de las líneas de asociación (por ejemplo: 1..*) indicando la cardinalidad de la relación. La Figura A-31 muestra un ejemplo de esta notación.

UML fue creado originalmente para modelamiento orientado a objetos, no para modelamiento de datos. Por lo tanto, no entrega mecanismos nativos para indicar claves primarias, entidades débiles, o llaves foráneas. Sin embargo, los diagramas de clases son más flexibles que los diagramas entidad-relación. Por ejemplo, una clase Orden puede incluir una lista de ítems como un atributo, mientras que en un diagrama entidad-relación, Item sería una entidad separada.

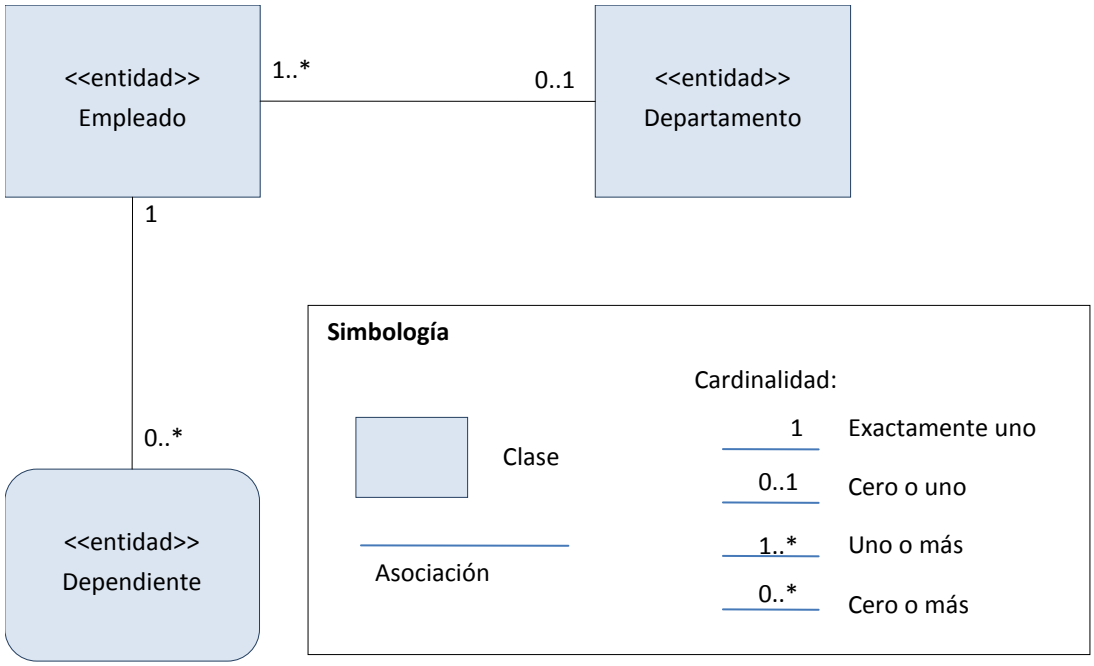


Figura A-31 Diagrama de modelo de datos simplificado de un sistema de recursos humanos utilizando un diagrama de clases UML

1.1.3. ¿Para qué es el Tipo de Vista de Módulo?

Se espera que el tipo de vista de módulo se utilice para:

- **Construcción**. Una vista de módulo puede entregar un modelo para el código fuente. En este caso, los módulos y las estructuras físicas, tales como archivos y directorios de código fuente, generalmente tienen una distribución cercana.
- **Análisis**. Existen dos técnicas importantes de análisis que son: trazabilidad de requerimientos y análisis de impacto. Con la trazabilidad de requerimientos debería ser posible determinar cómo los requerimientos funcionales de un sistema son enlazados con las responsabilidades de cada módulo. Generalmente, los requerimientos de alto nivel se resuelven a través de una secuencia de invocaciones, por lo cual, la documentación de esas secuencias muestra cómo cumple el sistema con sus requerimientos e permite identificar los requerimientos faltantes. En cambio, el análisis de impacto ayuda a predecir el efecto de modificar el sistema, para ello se utilizan los diagramas de contexto, que describen las relaciones de los módulos con otros módulos y con el mundo exterior. El análisis de impacto requiere cierto grado de completitud del diseño e integridad de la descripción de los módulos y, en especial, la información de dependencias debe estar disponible y correcta para obtener buenos resultados.
- **Comunicación**. Una vista de módulos puede ser utilizada para explicar las funcionalidades de un sistema a alguien que no está familiarizado con él. La cantidad de niveles de granularidad de la descomposición de módulos entrega una presentación top-down de las responsabilidades del sistema y, por lo tanto, puede guiar en el proceso de aprendizaje.

Por otra parte, es difícil utilizar el tipo de vista de módulo para realizar inferencias sobre el comportamiento en tiempo de ejecución, debido a que este tipo de vista es una partición de las funciones del software. Por ello, una vista de módulo no se usa generalmente para análisis de performance, disponibilidad, o muchas otras cualidades de tiempo de ejecución. Para ellas, generalmente se utilizan las vistas de componentes y conectores, y de despliegue.

1.1.4. Notaciones para el Tipo de Vista de Módulos

1.1.4.1. Notaciones Informales

Una notación informal común para un tipo de vista de módulos utiliza burbujas o cajas para representar los módulos, con diferentes tipos de líneas entre ellas para representar las relaciones. La anidación se utiliza para representar agregación y las flechas generalmente representan una relación depende-de.

Una segunda forma de notación común es un simple listado textual de los módulos con descripción de sus responsabilidades. Se pueden utilizar varios esquemas textuales para representar la relación es-parte-de, como la indentación, esquemas numerados, y paréntesis anidados. Otras relaciones pueden indicarse por palabras clave. Por ejemplo, la descripción de un módulo A puede incluir la línea “importa los módulos B, C” indicando una dependencia entre el módulo A y los módulos B y C.

1.1.4.2. UML

Las notaciones de modelamiento de objetos como UML proporcionan una variedad de construcciones que se pueden utilizar para representar los diversos tipos de módulos. La Figura A-33 muestra algunos ejemplos de los módulos usando la notación de paquetes en UML. Los paquetes pueden ser utilizados en caso de agrupación de funcionalidades importantes, como para representar capas y clases.

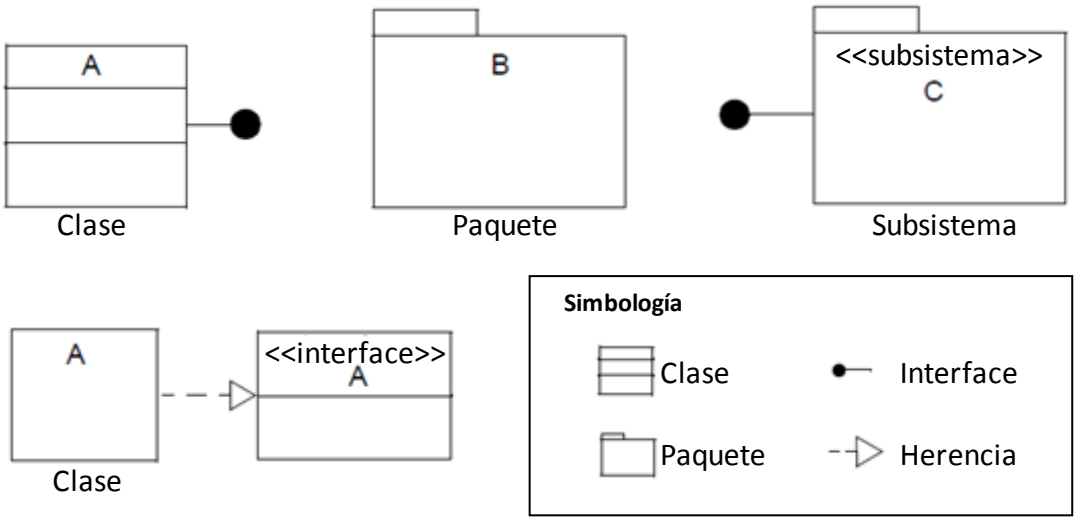


Figura A-32 Ejemplo de Notación de Módulos utilizando UML

UML tiene una clase de construcción, que es la especialización de un módulo que puede representar las relaciones nativas del tipo de vista de módulos como muestra la Figura A-33. En ella, una

relación de agregación indica que un módulo B es parte del módulo A, una relación de dependencia indica que un módulo D depende del módulo C, y en una relación de generalización un módulo F es un tipo de módulo E.

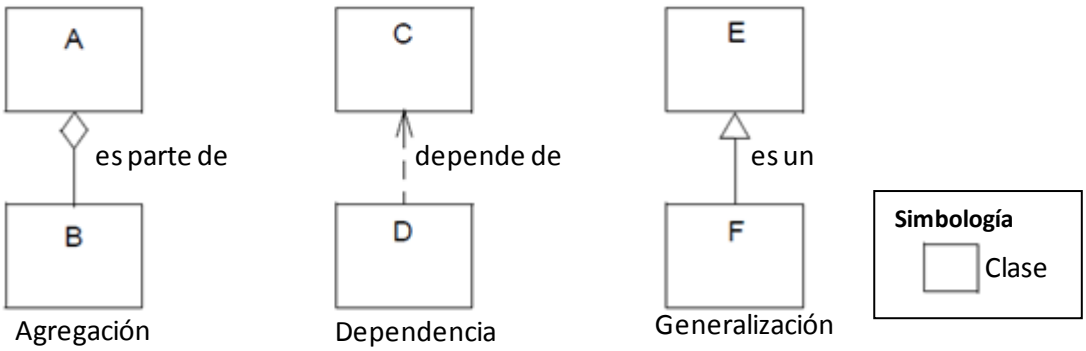


Figura A-33 Ejemplo de notación de relaciones en UML

1.2. Tipo de Vista de Componentes y Conectores

A diferencia de los estilos en el tipo de vista de módulos que apoyan la representación de elementos de un sistema en tiempo de diseño, los estilos en el tipo de vista de C&C expresan el comportamiento en tiempo de ejecución. Las vistas de C&C son descritas en términos de componentes y conectores que pueden tener muchas instancias del mismo tipo de componente. Por ejemplo, se puede tener un tipo de componente Cliente Web que es instanciado muchas veces dentro de la misma vista. Basándose en una analogía con sistemas orientados a objetos, las vistas C&C son similares a diagramas de objetos (o colaboración), y opuestas a diagramas de clases que definen los tipos de elementos.

- Un componente es una de las unidades principales de procesamiento para ejecutar un sistema. Algunos ejemplos de componentes son los procesos, objetos, clientes, servidores y los almacenes de datos.
- Un conector es un mecanismo de interacción entre componentes. Algunos ejemplos de conectores son los links de comunicación y protocolos, flujos de información, y acceso a almacenes de datos compartidos. Generalmente, las interrelaciones son llevadas a cabo usando infraestructuras complejas como frameworks, middlewares, canales de comunicación distribuidos, y programadores de procesos.

Los componentes y conectores pueden ser descompuestos en otros componentes y conectores, donde la descomposición de uno puede incluir otros componentes y conectores. Por ejemplo, el descomponer un middleware producirá varios componentes y conectores.
El elegir la forma apropiada de interacción entre elementos de procesamiento es un aspecto crítico de las tareas de arquitectura. Estas interacciones pueden representar formas complejas de comunicación. Por ejemplo, una conexión entre un componente cliente y un componente servidor puede representar un protocolo de comunicación complejo, ayudado por una infraestructura sofisticada en tiempo de ejecución. Otras interacciones pueden representar formas de comunicación

de múltiples partes, como difusión de eventos, o sincronización de datos de n-vías. Esas interacciones son capturadas como conectores en el tipo de vista C&C.

Las vistas de C&C se representan generalmente con diagramas de cajas y líneas para dar una explicación general de la arquitectura de un sistema. Sin embargo, el representar informalmente una vista de C&C puede resultar engañoso, ambiguo e inconsistente.

Se partirá con una revisión informal de la vista de C&C a través de un simple ejemplo como la Figura A-34 que muestra una representación de una vista C&C que puede encontrarse en una descripción de una arquitectura de un sistema típico en tiempo de ejecución. En la figura no queda claro qué se intenta describir o si esto está explicado en la documentación anexa. Se muestra una imagen del sistema tal y cómo aparece en tiempo de ejecución. El sistema contiene un repositorio compartido de cuentas de clientes (Base de Datos de cuentas) consultado por dos servidores y un componente de administración. Un grupo de cajeros puede interactuar con los servidores del repositorio de cuentas, que es mostrado utilizando un estilo cliente-servidor. Estos componentes clientes (los cajeros) se comunican entre sí mediante eventos de publicación y subscripción. El propósito de los dos servidores es aumentar la disponibilidad, de manera que si el servidor principal falla, puede responder el servidor de respaldo. Finalmente, un componente de Administración permite a un administrador acceder y mantener el almacén de datos compartido.

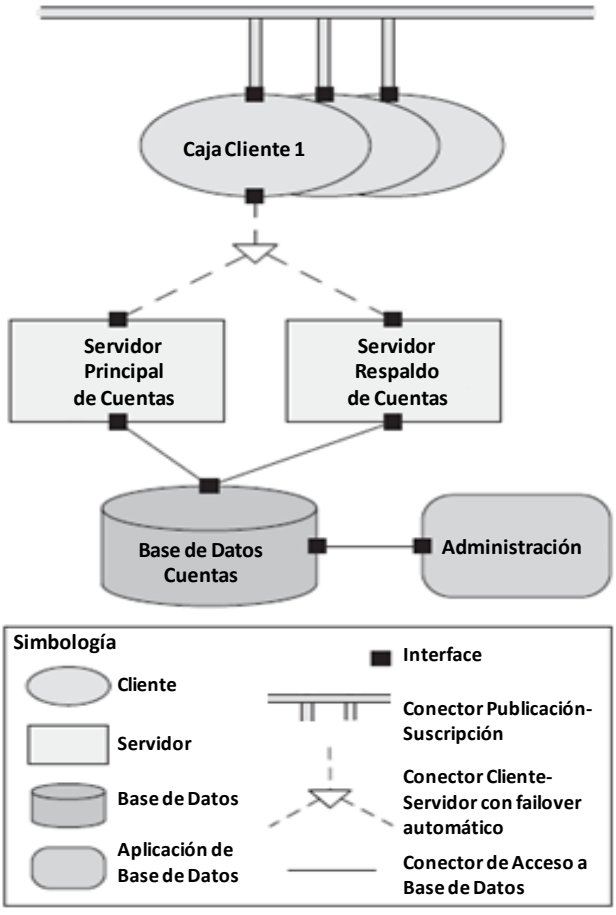


Figura A-34 Una vista superficial de un sistema de cómo podría parecer en tiempo de ejecución

Cada uno de los tres tipos de conectores mostrados en la Figura A-34 representa una forma diferente de interacción a través de sus partes conectadas.

- Los conectores Cliente-Servidor permiten entregar datos del repositorio de cuentas a un conjunto concurrente de clientes a través de solicitudes de servicios síncronas. Esta variante del estilo cliente-servidor permite la conmutación transparente ante una falla a un servidor de respaldo.
- El conector de acceso a la base de datos permite transacciones, acceso autenticado para lectura, escritura, y el monitoreo de la base de datos.
- El conector de publicación-suscripción permite la publicación de eventos asíncronos y notificaciones.

Cada uno de estos conectores representa una forma compleja de interacción y es probable que requieran mecanismos de implementación complejos. Por ejemplo, el tipo de conector cliente-servidor representa un protocolo de interacción que establece cómo los clientes inician una sesión cliente-servidor, cómo y cuándo es procesado un error, y cómo se terminan las sesiones. La implementación de este conector probablemente implicará mecanismos en tiempo de ejecución que detectarán cuando el servidor esté abajo, solicitudes de clientes encoladas, controlar la conexión y desconexión de clientes, y cosas similares.
Los conectores no necesitan ser binarios, dos de los tres tipos de conectores en la Figura A-34 pueden involucrar más de dos participantes; el conector de publicación-suscripción y los conectores cliente-servidor.
También es posible realizar análisis cuantitativos y cualitativos de las propiedades del sistema, como el rendimiento, fiabilidad y seguridad basados en este punto de vista. Por ejemplo, la decisión de diseño que la interfaz de usuario administrador sea el único componente capaz de cambiar el esquema de base de datos mejorará la seguridad del sistema. Aunque esta decisión pueda afectar el servicio o la disponibilidad. Por ejemplo, ¿el uso de la interfaz de administrador bloquea los servidores? Del mismo modo, al conocer las propiedades de confiabilidad de los servidores individuales y la base de datos, es posible generar estimaciones numéricas de la confiabilidad general del sistema, usando alguna forma de análisis de confiabilidad.
Hay algunas cosas que notar acerca de la naturaleza de la documentación gráfica de las vistas de C&C, como muestra la Figura A-34:

- Actúa como una clave para la documentación asociada al soporte, donde se pueden encontrar detalles acerca de los elementos, relaciones, y sus propiedades.
- Está restringida a la información que puede ser presentada y comprendida desde un diagrama.
- Es explicita en su vocabulario de tipos de componentes y conectores en la simbología del diagrama.
- Indica el número y el tipo de interfaces en sus componentes y conectores.
- Usa abstracciones de componentes y conectores que pueden tener semánticas ricas y una implementación compleja.

La documentación que explica el diagrama debe elaborarse en base a los elementos mostrados. La documentación de apoyo debe explicar, por ejemplo, cómo el servidor de backup mejora la disponibilidad del sistema. Algunos de los elementos mostrados en la Figura A-34 pueden representar subsistemas que tienen sus propias arquitecturas, mostradas en alguna parte.
La combinación de diagramas C&C y su documentación de soporte son una herramienta esencial para comunicar el diseño de un arquitecto, ayudando a analizar el comportamiento del sistema en tiempo de ejecución, y justificando decisiones de diseño en términos de su impacto en los atributos de calidad relevantes.

1.2.1. Elementos, Relaciones y Propiedades de los Vistas de C&C

La Tabla A-9 resume los elementos, relaciones, y propiedades que pueden aparecer en las vistas de C&C. Luego se complementa con una discusión más detallada de esos conceptos, en conjunto con guías relacionadas a su documentación.

Elementos	• **Componentes**: son las unidades de procesamiento principal y almacenes de datos. Un componente tiene un conjunto de puertos a través de los cuales interactúa con otros componentes (usando conectores). • **Conectores**: son vías de interacción entre componentes. Los conectores tienen un conjunto de funciones que indican cómo se deben usar en una interacción entre componentes.
Relaciones	• **Unión**: los puertos son asociados con las funciones de los conectores para obtener un gráfico de componentes y conectores. • **Delegación de Interfaz**: en algunas situaciones los puertos de componentes están asociados con uno o más puertos en una arquitectura interna. Similarmente a las funciones de un conector.
Restricciones	• Los componentes pueden ser conectados solo a conectores, no a otros componentes. • Los conectores pueden ser conectados únicamente a componentes, no a otros conectores. • Las uniones pueden ser sólo entre los puertos y funciones compatibles. • La delegación de Interfaz sólo se puede definir entre dos puertos compatibles (o dos funciones compatibles). • Los conectores no pueden aparecer de forma aislada, un conector debe estar conectado a un componente.
Para qué sirve	• Para mostrar cómo funciona el sistema • Para guiar el desarrollo a través de la estructura y el comportamiento de elementos en tiempo de ejecución • Para ayudar a los arquitectos y a otros a analizar los atributos de calidad en tiempo de ejecución, como el **performance**, **fiabilidad**, y **disponibilidad**.

Tabla A-9 Resumen de las Vistas de C&C

1.2.1.1. Elementos

Los elementos de una vista C&C son componentes y conectores. Cada elemento en una vista de C&C tiene una manifestación en tiempo de ejecución, consumiendo recursos de ejecución y contribuyendo al ambiente de ejecución del sistema. Las relaciones de unión de una vista de C&C asocian componentes con conectores (a través de sus respectivos puertos y funciones) para formar un gráfico que representa la configuración de un sistema en ejecución.

Componentes

Los componentes representan los principales elementos de proceso y los almacenes de datos que son presentados en tiempo de ejecución. Cada componente en una vista de C&C tiene un nombre que debe indicar la función prevista, además de permitir la relación entre el elemento gráfico con la documentación de ayuda del componente.

Los componentes tienen interfaces llamadas puertos que definen un punto específico de interacción potencial de un componente con su ambiente. Generalmente, tienen un tipo explícito, que define el tipo de comportamiento que puede tener ubicándolo en ese punto de interacción. Un componente

puede tener muchos puertos del mismo tipo. En ese sentido, los puertos difieren de las interfaces de módulo debido a que las interfaces nunca son replicadas. Por ejemplo, un filtro puede tener muchos puertos de entrada del mismo tipo para controlar múltiples tramas de entrada, o un servidor puede entregar un número de puertos de solicitud para interacción con clientes. La base de datos en la Figura A-34 tiene dos puertos y dos tipos de accesos.

Está permitido anotar en un puerto un número o rango de números para indicar replicación. Por ejemplo, un puerto con una anotación "[3]" se refiere a que existen 3 instancias disponibles de ese puerto. Un puerto con una anotación "[0..10]" significa que hay entre 0 y 10 instancias del puerto. Esta forma es útil cuando se definen tipos de componentes, permitiendo vincular el número exacto de instancias de componentes, o para componentes que crean puntos de interacción dinámicamente.

Un puerto de componente debe ser documentado explícitamente, mostrándolo en el diagrama y definiéndolo en la documentación de ayuda del mismo.

Un componente en una vista de C&C puede representar un subsistema complejo, el que en sí mismo puede ser descrito en una sub-arquitectura de C&C. Esta sub-arquitectura puede representarse gráficamente in-situ cuando no es demasiado compleja, mostrándola anidada dentro del componente que refina. Sin embargo, generalmente es documentada separadamente. Una sub-arquitectura de un componente puede estar en un estilo diferente a aquel en que aparece el componente.

Cuando un componente tiene una sub-arquitectura, se debe documentar la relación entre los puertos internos y externos. Esto se describirá más adelante a través del uso de una relación de interfaz de delegación.

Conectores

Los conectores son otro tipo de elementos en una vista de C&C. Algunos ejemplos simples de conectores son las invocaciones de servicios, las colas de mensajes asíncronos, la difusión de eventos, y tubos que representan asincronía, preservando el orden de los streams de datos. Pero los conectores representan formas mucho más complejas de interacción, como un canal de comunicación orientado a la transacción entre un servidor de base de datos y un cliente, o un bus de servicios empresariales que media la interacción entre colecciones de servicios, usuarios y proveedores.

Los conectores tienen roles en sus interfaces que definen las formas en las que el conector puede ser usado por los componentes para llevar a cabo una interacción. Por ejemplo, un conector cliente-servidor puede tener roles de invocación de servicios y proveedor de servicios. Una tubería puede tener roles de escritor y lector. Como los puertos de componentes, los roles de conectores difieren de las interfaces de módulos en que ellos pueden ser replicados, indicando cuantos componentes pueden involucrarse en la interacción. Un conector de publicación-suscripción puede tener muchas instancias de los roles publicador y subscriptor.

Un rol generalmente define las expectativas de un participante en la interacción. Por ejemplo, un rol servicios de invocación puede requerir que el servicio invocador inicialice la conexión antes de emitir cualquier solicitud de servicio. La semántica de interacción representada por el conector generalmente es documentada como una especificación de protocolo que indica qué patrones de eventos o acciones están permitidos a través del conector.

Como los componentes, los conectores complejos pueden ser descompuestos en colecciones de componentes y conectores que describen la subestructura arquitectural de esos conectores. Por ejemplo, en una descomposición de un conector cliente-servidor tolerante a fallos de la Figura A-34

probablemente incluiría componentes que son responsables de amortiguar solicitudes de clientes, determinar cuándo un servidor ha fallado, y el enrutamiento de solicitudes.

1.2.1.2. Relaciones

La relación principal en una vista de C&C es la **unión**. Las uniones indican qué conectores son unidos a qué componentes, con ello se define un sistema como un grafo de componentes y conectores. Específicamente, una unión indica una asociación de un puerto de un componente a una función de un conector.

Una unión válida es una en la cual los puertos y funciones son compatibles entre sí, bajo restricciones semánticas definidas por el estilo. Por ejemplo, en una arquitectura de llamada y retorno se debe confirmar que todos los puertos son unidos a algún conector de llamada-retorno. En un nivel más detallado, se debe verificar que el protocolo del puerto es consistente con el comportamiento esperado por la función al cual está unido.

Cuando se unen componentes a conectores se debe seguir las siguientes recomendaciones:

- Se puede representar una unión en el diagrama simplemente conectándolo a los puertos de un componente. Para algunos casos, donde el contexto deja claro qué funciones se unen, no es necesario representar las funciones explícitamente en el diagrama.
- Un conector se debe unir a un puerto de un componente, no directamente al componente.
- Si no está claro que es válido unir un puerto a una función determinada, se debe justificar en una nota en el diagrama o en la sección de justificación para la vista.
- El unir conectores entre puertos con un factor de multiplicidad (como [5] o [0..10]) es una gran fuente de ambigüedad. Por ejemplo, ¿qué significa si se conecta un puerto de multiplicidad 3 a un puerto de multiplicidad 22? Si se conectan dos puertos con la misma multiplicidad (más de 1) ¿Qué puertos en un componente están conectados a qué puerto en el otro? Si se utiliza esta notación, se debe explicar qué se quiere decir.

Un segundo tipo de relación es la **delegación de interfaces**. Cuando un componente o conector tiene una subarquitectura, es importante documentar la relación entre la estructura interna y las interfaces externas del componente o conector. La relación puede ser documentada usando relaciones de delegación de interfaces. Estas relaciones vinculan los puertos internos con los externos (en el caso de los componentes) o las funciones internas con externas (para los conectores). Algunas notaciones entregan elementos gráficos específicos para caracterizar estas relaciones. La Figura A-35 muestra un ejemplo de delegación de interfaces utilizando notación UML. En UML, la delegación de conectores es utilizada para representar delegación de interfaces.

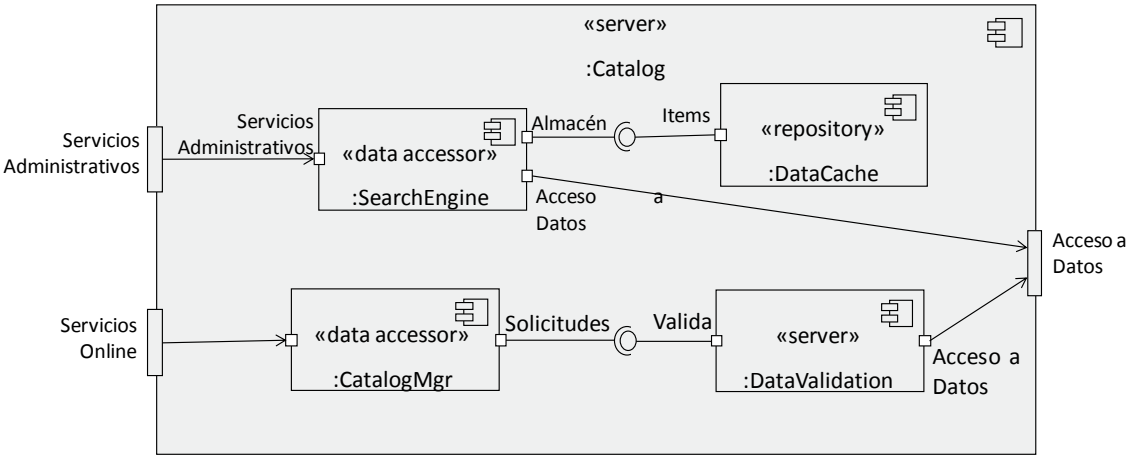


Figura A-35 Un diagrama de componentes UML mostrando la subarquitectura de un componente llamado catálogo

1.2.1.3. Propiedades

Un elemento (componente o conector) de una vista de C&C tendrá varias propiedades asociadas. Cada elemento debería tener un nombre y un tipo. Las propiedades adicionales dependen del tipo de componente o conector. Las propiedades son necesarias para guiar la implementación y la configuración de componentes y conectores, pero el arquitecto además debe definir valores para las propiedades destinados al análisis particular de la vista C&C. Por ejemplo, pueden ser necesarias si la vista va a ser utilizada en análisis de performance, latencia, capacidad de encolamiento, y prioridades de hilos de ejecución. Los siguientes son ejemplos de algunas propiedades típicas y sus usos:

- **Fiabilidad:** ¿Cuál es la probabilidad de fracaso de un determinado componente o conector? Esta propiedad puede utilizarse para ayudar a determinar la fiabilidad general del sistema.
- **Rendimiento:** ¿Qué clases de tiempos de respuestas entregará el componente bajo qué cargas? ¿Qué clases de latencias y rendimiento se puede esperar de un conector dado? Esta propiedad se puede utilizar con otras para determinar propiedades del sistema como tiempo de respuesta, rendimiento, y necesidades de almacenamiento en buffer.
- **Necesidades de Recursos:** ¿Cuáles son las necesidades de procesamiento y almacenamiento de un componente o conector? Esta propiedad se puede utilizar para determinar si una configuración de hardware propuesto es adecuado.
- **Funcionalidad:** ¿Qué funciones realiza un elemento? Esta propiedad puede ser utilizada para analizar el proceso global realizado por un sistema.
- **Seguridad:** ¿El componente o conector asegura o entrega características de seguridad, como encriptación, auditoria, o autenticación? Esta propiedad puede ser usada para determinar vulnerabilidades de seguridad del sistema.
- **Concurrencia:** ¿Este componente se ejecuta en un proceso o hilo separado? Esta propiedad puede ayudar a analizar o simular el rendimiento de componentes concurrentes e identificar posibles deadlocks.

- **Niveles:** En una topología de niveles, ¿En qué nivel reside el componente? Esta propiedad ayuda a definir los procedimientos de generación e implementación, así como los requisitos de plataforma para cada nivel.

Además, los puertos y funciones pueden tener propiedades asociadas a ellos. Por ejemplo, se pueden especificar las tasas máximas de solicitudes soportadas por un puerto de servidor.

1.2.2. Algunos Estilos del Tipo de Vista de Componentes y Conectores

Un estilo de componentes y conectores incluye un conjunto específico de tipos de componentes y conectores y reglas específicas de cómo sus elementos pueden ser combinados. Sus vistas entregan los aspectos de tiempo de ejecución de un sistema, un estilo de componentes y conectores generalmente es asociado a una topología que establece como los datos y los flujos de control a través del sistema son diseñados con el estilo.
Una forma de imponer un orden conceptual en el espacio de estilos de componentes y conectores es considerar varias categorías amplias de estilos, diferenciadas principalmente por su topología. Por ejemplo:

- **Estilos de Flujo de Datos:** estilos en los cuales los cálculos son controlados por el flujo de datos a través del sistema.
- **Estilos de Llamada-Respuesta:** estilos en los cuales sus componentes interactúan a través de invocaciones síncronas de las capacidades de otros componentes.
- **Estilos Basados en Eventos:** estilos en los cuales sus componentes interactúan a través de eventos o mensajes asíncronos.
- **Estilos de Repositorios:** estilos en los cuales sus componentes interactúan a través de grandes colecciones de datos persistentes compartidos.

La Figura A-36 entrega una interpretación general de parte del espacio de estilos de C&C. En la parte superior está la forma más general y sin restricciones de vistas de C&C, la que utilizaría componentes y conectores genéricos con ninguna restricción particular en su topología, comportamiento, o propiedades de elementos. Bajo esta, están estilos especializados descritos anteriormente. También se debe tener en cuenta que estos estilos especializados pueden especializarse en más de una categoría general como es el caso del estilo de arquitectura orientada a servicios (SOA).

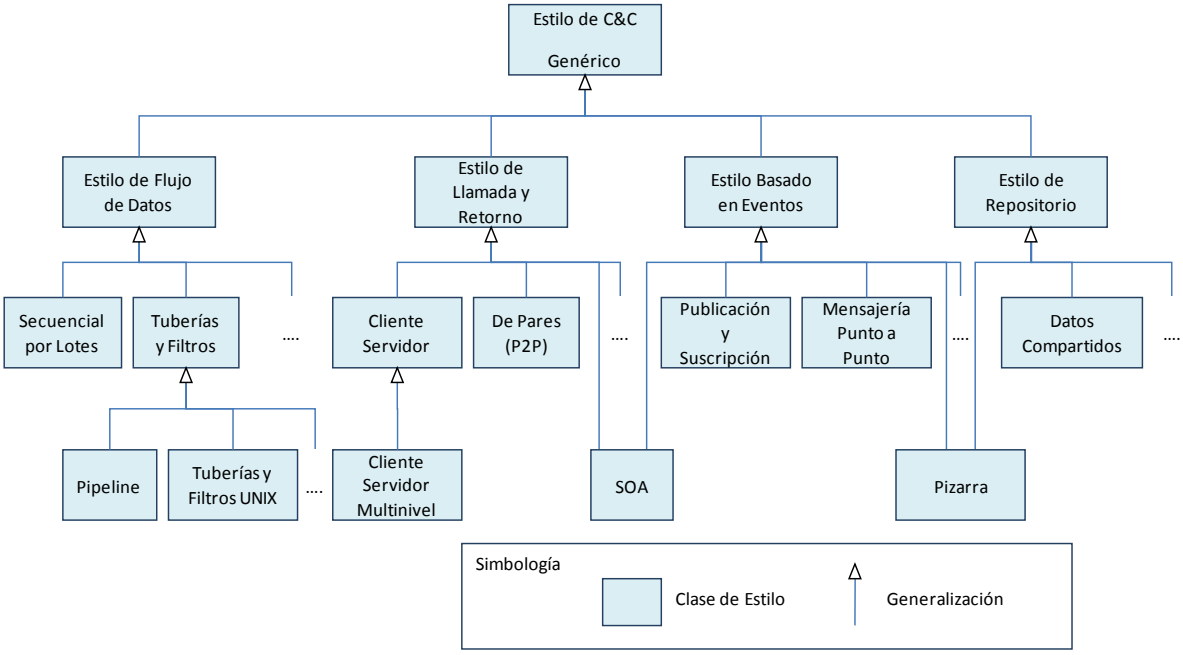


Figura A-36 Una representación parcial del espacio de estilos de C&C

1.2.2.1. Estilos de Flujo de Datos

Esta categoría de estilos enfatiza la reutilización y la modificabilidad. Es apropiada para sistemas que implementan transformaciones de datos en pasos sucesivos. Ejemplos de los estilos de flujos de datos son las arquitecturas de tubería-filtros y las de proceso secuencial en lote.

Estilo Tubería y Filtros (pipe-and-filter style)

Es uno en que el patrón de interacción está caracterizado por transformaciones sucesivas de datos. Los datos llegan a un filtro, son transformados, y son pasados a través de la tubería hacia el próximo filtro en la línea de tuberías. Ejemplos de tales sistemas son sistemas de procesamiento de señales y los UNIX pipes.

Una tubería (pipeline) es una arquitectura popular que conecta componentes de procesamiento (filtros) a través de conectores (pipes), de modo que las computaciones se ejecutan en forma de flujo. Los datos se transportan a través de las tuberías entre los filtros, transformando gradualmente las entradas en salidas. Debido a su simplicidad y su facilidad para captar una funcionalidad, es una arquitectura muy utilizada cada vez que se trata de demostrar ideas sobre la formalización del espacio de diseño arquitectónico.

La aplicación típica del estilo es un procesamiento clásico de datos [11]: el cliente hace un requerimiento; el requerimiento se valida; un Web Service toma el objeto de la base de datos; se lo convierte a HTML; se efectúa la representación en pantalla. El estilo tubería-filtros propiamente dicho enfatiza la transformación incremental de los datos por sucesivos componentes. El caso típico es la familia UNIX de Sistemas operativos, pero hay otros ejemplos como la secuencia de procesos de los compiladores (sobre todo los más antiguos), el tratamiento de documentos XML como ráfaga en SAX, ciertos mecanismos de determinados motores de servidores de bases de datos, algunos procesos de workflow y subconjuntos de piezas encadenadas.

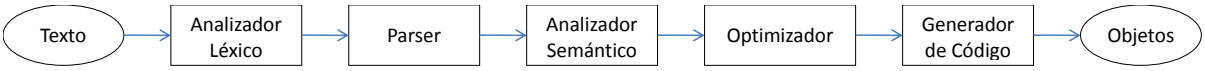


Figura A-37 Compilador en Estilo Tuberías y Filtros

Elementos, Relaciones y Propiedades

La Tabla A-10 muestra un resumen de los elementos básicos del estilo de Tuberías y Filtros. En él, se considera un único tipo de componente –el filtro- y sólo un tipo de conector –la tubería-. Un filtro transforma los datos que se reciben desde una o más tuberías y transmite los resultados a través de una o más tuberías.

Elementos	• Filtro (filter): es un componente que transforma los datos leídos por sus puertos de entrada a datos que son escritos en sus puertos de salida. Generalmente, los filtros se ejecutan en forma concurrente e incremental. Sus propiedades pueden especificar capacidad de procesamiento, formatos de entrada/salida y las transformaciones ejecutadas por el filtro. • Tubería (pipe): es un conductor que traslada datos desde los puertos de salida de un filtro hacia los puertos de entrada de otro filtro. Una tubería tiene un rol de entrada y salida, mantiene el orden de los registros de datos que transitan por ella y no los altera. Sus propiedades pueden indicar tamaño del buffer, protocolo de interacción, y el formato de datos que pasan a través de la tubería.
Relaciones	• Uniones: asocian los puertos de salida de un filtro a los roles de entrada en una tubería, y los puertos de entrada de un filtro a los roles de salida de la misma.
Topología	• Los datos son transformados desde los puertos de entrada del sistema hacia sus puertos de salida a través de una serie de transformaciones realizadas por sus filtros.
Restricciones	• Las tuberías se conectan a los puertos de salida de los filtros a los puertos de entrada de los filtros. • Los filtros conectados deben acordar los tipos de datos que serán pasados por la tubería. • Las especializaciones de este estilo pueden restringir la asociación de componentes a un grafo acíclico o una secuencia lineal –a veces llamada pipeline-. • Otras especificaciones pueden indicar que los componentes tienen puertos con nombres, como los filtros UNIX stdin, stdout, y stderr.
Para qué utilizarlo	• Mejorar el reúso producto de la independencia de los filtros. • Mejorar el rendimiento producto de la paralelización del procesamiento de datos. • Simplificar el análisis sobre el comportamiento general.

Tabla A-10 Resumen del Estilo de Tuberías y Filtros

Los filtros generalmente se ejecutan concurrente e incrementalmente. Principalmente, un filtro no necesita saber la identidad de los filtros rio arriba o rio abajo. Por esta razón, los sistemas de

tuberías y filtros tienen la propiedad de composición funcional de los filtros, permitiendo una simplificación del análisis del comportamiento del sistema a través del comportamiento de cada una de sus partes.

¿Para qué es este estilo?

El estilo de tuberías y filtros es utilizado generalmente para sistemas de transformación de datos, donde el procesamiento completo puede ser dividido en un conjunto de pasos independientes, cada uno responsable de una transformación incremental de sus datos entrada. La independencia del proceso realizado por cada paso permite el reúso, paralelización, y la simplificación del análisis del comportamiento completo del sistema. Por lo tanto, se pueden considerar las siguientes condiciones para utilizar este estilo:

- Se puede especificar en una secuencia de un número conocido de pasos.
- No se requiere esperar la respuesta asincrónica de cada paso.
- Se busca que todos los componentes situados aguas abajo sean capaces de inspeccionar y actuar sobre los datos que vienen de aguas arriba (pero no viceversa).

La Tabla A-11 indica las ventajas y desventajas de utilizar el estilo de tuberías y filtros

Ventajas	Desventajas
• Es simple de entender e implementar. Es posible implementar procesos complejos con editores gráficos de líneas de tuberías o con comandos de línea. • Fuerza un procesamiento secuencial. • Es fácil de envolver (wrap) en una transacción atómica. • Los filtros se pueden empaquetar, y hacer paralelos o distribuidos.	• El patrón puede resultar demasiado simplista, especialmente para orquestación de servicios que podrían ramificar la ejecución de la lógica de negocios de formas complicadas. • No controla muy eficientemente construcciones condicionales, bucles y otras lógicas de control de flujo. Agregar un paso suplementario afecta la performance de cada ejecución de la tubería. • Eventualmente pueden llegar a requerirse buffers de tamaño indefinido, por ejemplo en las tuberías de clasificación de datos. • El estilo no es apto para manejar situaciones interactivas, sobre todo cuando se requieren actualizaciones incrementales de la representación en pantalla. • La independencia de los filtros implica que es muy posible la duplicación de funciones de preparación que son efectuadas por otros filtros (por ejemplo, el control de corrección de un objeto de fecha).

Tabla A-11 Ventajas y Desventajas del Estilo de Tuberías y Filtros

Relación con otros estilos y modelos

Una vista de un sistema de tuberías y filtros no es lo mismo que un modelo de flujo de datos. En el estilo de tuberías y filtros, las líneas entre componentes representan conectores, que representan un proceso específico como es el transmitir corrientes de datos desde un filtro a otro. En los modelos de flujos de datos, las líneas representan relaciones, indicando que existe una comunicación de datos entre componentes. Los flujos en un modelo de flujo de datos tienen un procesamiento menor donde se indica que existe un flujo de datos desde un elemento a otro. Este flujo puede ser realizado por un conector, como una llamada a procedimiento, el ruteo de un evento entre un publicador y sus suscriptores, o datos transmitidos a través de una tubería. La razón del porqué estas vistas pueden ser confundidas es que el modelo de flujo de datos de un estilo de tuberías y filtros luce casi idéntico a la vista original de tuberías y filtros.

Los estilos de flujo de datos son generalmente combinados con otros estilos usándolos para caracterizar un subsistema particular.

Estilo Secuencial por Lotes

En el estilo secuencial por lotes (batch sequential) los componentes son programas independientes; el supuesto es que cada paso se ejecuta hasta completarse antes que se inicie el paso siguiente. Este estilo se reconoce como una variante del estilo de tuberías y filtros por lo que se aplican los mismos elementos, relaciones y propiedades considerando que los componentes son programas independientes.

1.2.2.2. Estilos de Llamada y Retorno (Call-Return style)

Los estilos de llamada y retorno representan un modelo de procesamiento en el cual los componentes entregan un conjunto de servicios u operaciones que pueden ser invocados por otros componentes. Un componente que invoca un servicio se detiene (o es bloqueado) hasta que el servicio haya completado su procesamiento. Por lo tanto, es análogo a una llamada a procedimientos en muchos lenguajes de programación. Los conectores son los responsables de convertir las solicitudes de servicio desde el solicitante hasta el proveedor y de entregar sus resultados. Esta categoría de estilos enfatiza la **modificabilidad** y la **escalabilidad**, son los estilos más utilizados en sistemas a gran escala.

Los estilos de llamada y retorno se diferencian de otros por muchas razones. Algunas de estas diferencias corresponden al comportamiento de sus conectores, por ejemplo, los conectores en algunos estilos de llamada y retorno pueden permitir control de errores (como cuando el proveedor de servicios no está disponible). Otras diferencias se relacionan con restricciones de su topología, por ejemplo, algunas arquitecturas de llamada y retorno están organizadas en niveles. También existen los que agrupan componentes en conjuntos que pueden hacer solicitudes y los que pueden responderlas.

Algunos ejemplos de estilos de llamada y retorno son los estilos cliente-servidor, peer-to-peer, y rest.

Estilo Cliente Servidor (client-server style)

Muestra componentes interactuando a través de la solicitud de servicios de otros componentes. La esencia de este estilo es que la comunicación es generalmente pareada. Una solicitud de servicio de un cliente es pareada con la provisión de ese servicio. Los servidores en este estilo entregan un conjunto de servicios a través de una o más interfaces, y los clientes usan cero o más servicios provistos por otros servidores en el sistema. En este estilo puede haber un servidor central o muchos servidores distribuidos. Algunos ejemplos de sistemas clientes-servidor son:

- Sistemas de ventanas, que dividen el sistema de acuerdo a una aplicación cliente y servidor de pantalla.
- Servicios DNS, que se dividen de acuerdo al resolutor de nombres y el servidor de nombres,
- Sistemas de Bases de Datos de dos niveles que dividen el sistema de acuerdo a los clientes y los datos.
- Sistemas Distribuidos basados en web, que dividen el sistema de acuerdo a sus responsabilidades como aplicaciones clientes, lógica de negocio, y servicios de administración de datos.
- Sistemas de información que se encuentran en redes locales, donde los clientes son aplicaciones con interfaces de usuario gráficas (como las de visual basic) y el servidor es un sistema de administración de bases de datos (como oracle).
- Aplicaciones web donde los clientes corren en un web browser y los servidores son componentes ejecutados en un servidor web (como apache tomcat).

Elementos, Relaciones y Propiedades

En el estilo cliente servidor los tipos de componentes pueden ser clientes o servidores. El tipo de conector principal es el conector consulta-respuesta utilizado para invocar servicios. Cuando se puede invocar más de un servicio en el mismo conector, generalmente se utiliza una especificación de protocolo para documentar relaciones de orden a través de los servicios invocables por él. Los servidores tienen puertos que describen los servicios que pueden entregar. Los clientes tienen puertos que describen los servicios que necesitan. Los servidores pueden convertirse en clientes que requieren servicios de otros servidores. Un componente que tiene puertos de solicitud y de entrega de servicios puede funcionar como cliente y servidor.

La Tabla A-12 resume las características del estilo cliente servidor:

Elementos	• **Cliente**: es un componente que invoca servicios de un componente servidor. • **Servidor**: es un componente que entrega servicios a componentes clientes. Sus propiedades variarán de acuerdo a sus responsabilidades en la arquitectura pero generalmente incluyen información acerca de la naturaleza de los puertos servidores (por ejemplo cuantos clientes se pueden conectar) y características de rendimiento (como la tasa máxima de invocación de servicios). • **Conector Consulta-Respuesta**: es utilizado por un cliente para invocar los servicios en un servidor. Los conectores consulta-respuesta tienen dos roles: un role de solicitud y un rol de respuesta. Las propiedades de un conector pueden incluir si las solicitudes son locales o remotas y si los datos son encriptados.
Relaciones	• **Uniones**: la relación de unión asocia los puertos de solicitud de servicio del cliente con el rol de solicitud del conector y a los puertos de servicio de solicitud del servidor con el rol de respuestas del conector.
Topología	• Los clientes inician interacciones, invocando servicios de los servidores cuando se necesiten y esperando los resultados de las solicitudes.

Restricciones	• Los clientes están conectados a los servidores a través de conectores consulta-respuesta
	• Los componentes servidores pueden ser clientes de otros servidores.
	• Existen especializaciones que pueden imponer otras restricciones:
	  ○ Número de adjuntos a un puerto específico
	  ○ Relaciones permitidas entre servidores
	• Los componentes pueden ser organizados en niveles.

Para qué utilizarlo	• Para promover la escalabilidad y el reúso a través de la factorización de servicios comunes. • Mejorar la escalabilidad y la disponibilidad en caso de que el servidor esté siendo replicado. • Analizar la dependencia ante fallas de alguno de los componentes, la seguridad para prevenir accesos no autorizados, y el rendimiento para dar servicios a un número determinado de clientes a tasas de respuesta determinadas.

Tabla A-12 Resumen del estilo cliente servidor

¿Para qué es este estilo?

El estilo cliente servidor presenta una vista de sistema que separa las aplicaciones clientes de los servicios que utiliza. Este estilo ayuda al entendimiento del sistema y el reúso debido a la factorización de servicios comunes. Debido a que los servidores pueden dar respuesta a un número indeterminado de clientes, es relativamente fácil agregar nuevos clientes al sistema. Asimismo, los servidores pueden ser replicados para permitir escalabilidad y disponibilidad.

Relación con otros estilos

Como en muchos estilos de componentes y conectores, el estilo cliente servidor desacopla productores de servicios y datos de sus consumidores. Otros estilos como el estilo de pares, utilizan un esquema más flexible en que los productores pueden actuar como productores y consumidores al mismo tiempo.

Los clientes y servidores generalmente son agrupados y distribuidos en diferentes máquinas en ambientes distribuidos para formar jerarquías de varios niveles.

Especializaciones del estilo

En el estilo cliente servidor generalmente se encuentran las siguientes especializaciones:

- Modelo Vista Controlador
- Arquitectura Cliente-Servidor Multinivel
- Arquitectura Orientada a Objetos
- Arquitectura Basada en Componentes

Estilo de Arquitectura Modelo Vista Controlador

El Modelo Vista Controlador (MVC) ha sido propio de las aplicaciones en Smalltalk por lo menos desde 1992, antes que se generalizaran las arquitecturas en múltiples niveles. En ocasiones se lo define más bien como un patrón de diseño o como práctica recurrente.

Un propósito común en numerosos sistemas es el de tomar datos de un repositorio y mostrarlos al usuario. Luego que el usuario introduce modificaciones, estas son aplicadas en el repositorio. Dado que el flujo de información ocurre entre el repositorio y la interfaz, un anti-patrón es unir ambas piezas para reducir la cantidad de código y optimizar la performance. Sin embargo, esta idea es desechada debido a que la interfaz suele cambiar, o acostumbra depender de distintas clases de dispositivos (clientes ricos, browsers, PDAs); la programación de interfaces de HTML, además, 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.

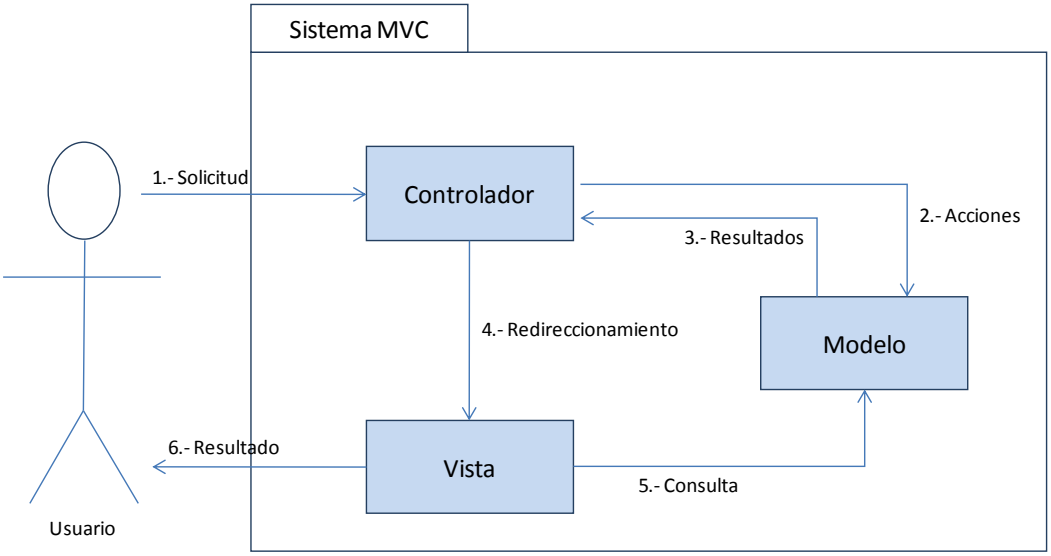


Figura A-38 Modelo Vista Controlador

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

- **Modelo**: El 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**: Interpreta las acciones del ratón y el teclado, informando al modelo y/o a la vista para que cambien según resulte apropiado.

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. La separación entre vista y controlador puede ser secundaria en aplicaciones de clientes ricos y, de hecho, muchos frameworks de interfaz implementan ambos roles en un solo objeto.
Por otra parte, en aplicaciones de Web la separación entre la vista (el browser) y el controlador (los componentes del lado del servidor que manejan los requerimientos de HTTP) está mucho más taxativamente definida.
La Tabla A-13 indica las ventajas y desventajas de utilizar un Modelo Vista Controlador.

Ventajas	Desventajas
• **Soporte de vistas múltiples**, debido a 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 de Web pueden utilizar el mismo modelo de objetos, mostrado de maneras diferentes. • **Adaptación al cambio**, debido a que 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. Este patrón sentó las bases para especializaciones ulteriores, tales como Page Controller y Front Controller.	• **Complejidad**, ya que el patrón introduce nuevos niveles de indirección y, por lo tanto, aumenta ligeramente la complejidad de la solución. También se profundiza la orientación a eventos del código de la interfaz de usuario, lo que puede llegar a ser difícil de depurar. En rigor, la configuración basada en eventos de la interfaz corresponde a un estilo particular (arquitectura basada en eventos) que aquí se examina por separado. • **Costo de actualizaciones frecuentes**, donde el desacoplar el modelo de la vista no significa que los desarrolladores del modelo puedan ignorar la naturaleza de las vistas. Si el modelo experimenta cambios frecuentes, podrían desbordar las vistas con una lluvia de requerimientos de actualización. Hace pocos años sucedía que en algunas vistas, tales como las pantallas gráficas, tomaba más tiempo el plasmar el dibujo que el que demandaban los nuevos requerimientos de actualización.

Tabla A-13 Ventajas y Desventajas del Modelo Vista Controlador

Estilo de Arquitectura Cliente-Servicdor Multinivel

El estilo de arquitectura cliente servidor multinivel es una generalización del modelo de tres niveles, donde se añaden más niveles para separar incumbencias. Aquí, la lógica de aplicación se reparte en diferentes niveles ubicados entre el cliente y los datos. Cada nivel se comunica sólo con los niveles contiguos a través de interfaces bien definidas y entre niveles inferiores entregan servicios a los superiores. Esta es una estructura muy utilizada en sistemas basados en componentes distribuidos (objetos distribuidos).

Generalmente, los niveles son confundidos con capas, sin embargo, las capas muestran agrupaciones de unidades de implementación y corresponden a un estilo de módulos, en cambio, los niveles agrupan componentes en tiempo de ejecución por ser un estilo de componentes y conectores.

La Tabla A-14 indica las ventajas y desventajas de utilizar una Arquitectura Cliente Servidor Multinivel.

Ventajas	Desventajas

• Elementos críticos de la lógica de negocio ubicados en nivel medio
  ○ Más cercanos al nivel de datos para brindar eficiencia de acceso
  ○ Sólo los datos realmente necesarios acaban llegando al cliente
• Mayor flexibilidad y modularidad
• Escalabilidad: facilita añadir recursos para soportar mayor número de clientes
• Extensibilidad: facilita añadir nuevas funcionalidades al sistema sin afectar a los clientes existentes
• Seguridad: facilidad para propagar autenticación y permisos a través de los distintos niveles.
• Facilidades de desarrollo y administración:
  ○ Reusabilidad de componentes.
  ○ Aislamiento frente a cambios en otros niveles.
  ○ Independencia frente a cambios en base de datos.
	• Complejidad: mayor número de elementos de hardware y software a definir, gestionar y mantener
	  ○ Interacciones complejas entre componentes
	  ○ Dificultad para detectar, asilar y corregir fallos
	• Costo de comunicaciones: mayor latencia y consumo de ancho de banda al atravesar niveles distribuidos por la red.
	• Costos de Mantenimiento: al crecer la cantidad de niveles aumenta el costo y la dificultad de instalación y mantenimiento.

Tabla A-14 Ventajas y Desventajas de la Arquitectura Cliente Servidor Multinivel

En la Figura A-39 se muestra un ejemplo de la Arquitectura Cliente-Servidor Multinivel donde se divide a una aplicación en múltiples niveles.

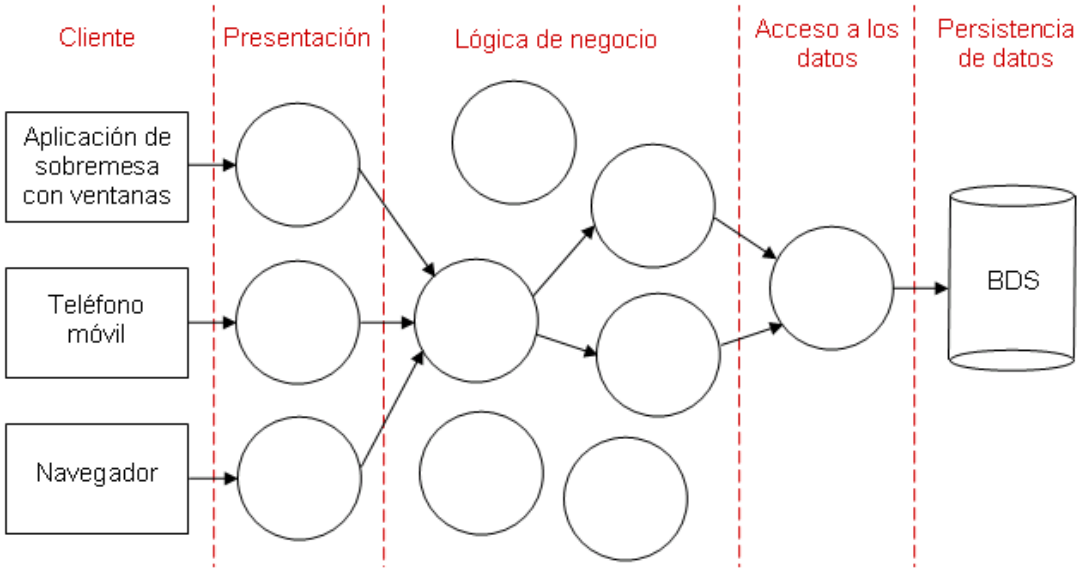


Figura A-39 Arquitectura cliente servidor multinivel

A continuación se describe cada uno de los niveles:

- **Cliente:** Este nivel contiene componentes de fachada, controlador y validación de datos. El objetivo de este nivel es crear un entorno de desarrollo de Diseño Gráfico que no tenga visibilidad con la lógica de negocio ni con la persistencia de datos.
- **Presentación:** Esta nivel contiene componentes de fachada, controlador y validación de datos. El objetivo de este nivel es crear un punto de unión entre el Cliente y la Lógica de negocio.
- **Lógica de negocio**: Este nivel es el corazón de la Aplicación. Su objetivo es que toda la lógica de negocio de la aplicación esté bien localizada y no mezclada con los objetos de otros niveles.
- **Acceso a los Datos**: Este nivel contiene componentes que llevan a cabo conversaciones de bajo nivel con el nivel de Persistencia de datos. Su objetivo es que todas las conversaciones de bajo nivel que pueda tener la aplicación con la Base de datos se realicen a través de él.

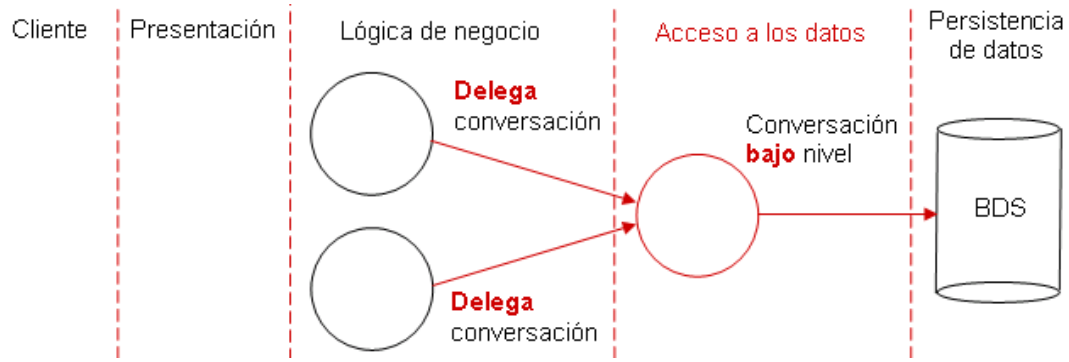


Figura A-40 Nivel de Acceso a Datos en una Arquitectura Cliente Servidor Multinivel

- **Persistencia de Datos**: Es en este nivel donde los datos de la aplicación están almacenados.

Estilo de Arquitectura Orientada a Objetos

Los componentes de este estilo son los objetos, o más bien instancias de tipos de datos abstractos. Los objetos representan una clase de componentes que son llamados gestores (ver Figura A-41), debido a que son responsables de preservar la integridad de su representación. Un rasgo importante de este aspecto es que la representación interna de un objeto no es accesible desde otros objetos. No se establece como cuestión definitoria el principio de herencia, a pesar de que la relación de herencia es un mecanismo organizador importante para definir los tipos de objeto en un sistema concreto, ella no posee una función arquitectónica directa. En particular, en esta concepción la relación de herencia no puede concebirse como un conector, puesto que no define la interacción entre los componentes de un sistema. Además, en un escenario arquitectónico la herencia de propiedades no se restringe a los tipos de objeto, sino que puede incluir conectores e incluso estilos arquitectónicos enteros.

Nombres alternativos para este estilo han sido Arquitecturas Basadas en Objetos, Abstracción de Datos y Organización Orientada a Objetos.

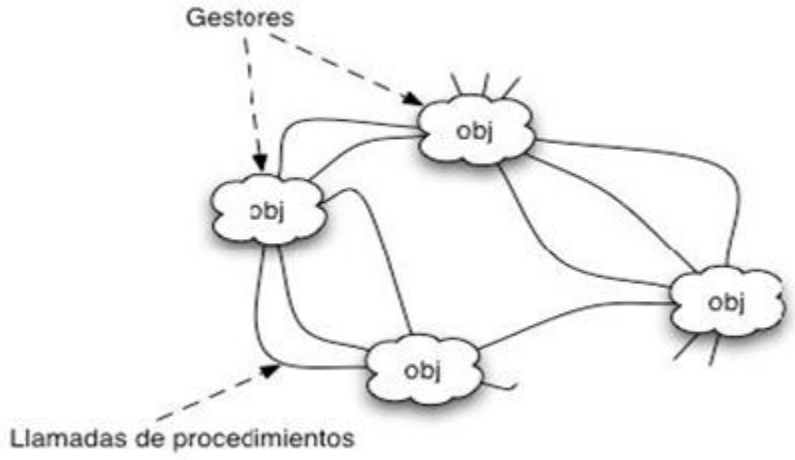


Figura A-41 Ejemplo de Estilo de Arquitectura Orientada a Objetos

Características

Si hubiera que resumir las características de las arquitecturas OO, se podría decir que:

- Los componentes del estilo se basan en principios OO: encapsulamiento, herencia y polimorfismo. Son asimismo las unidades de modelado, diseño e implementación, y los objetos y sus interacciones son 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 (lo mismo que para los componentes en el sentido de CBSE) 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.

- En cuanto a las restricciones, puede admitirse o no que una interfaz pueda ser implementada por múltiples clases. En tantos componentes, los objetos interactúan a través de invocaciones de funciones y procedimientos. Hay muchas variantes del estilo; algunos sistemas, por ejemplo, admiten que los objetos sean tareas concurrentes; otros permiten que los objetos posean múltiples interfaces.

La Tabla A-15 indica las ventajas y desventajas de utilizar una Arquitectura Orientada a Objetos.

Ventajas	Desventajas
• Se puede modificar la implementación de un objeto sin afectar a sus clientes (**modificabilidad**). • Es posible descomponer problemas en colecciones de agentes en interacción. • Un objeto es ante todo una entidad reutilizable en el entorno de desarrollo.	• Para que un objeto interactúe con otro objeto a través de una invocación de procedimiento, se debe conocer su identidad. Esta situación contrasta con lo que es el caso en estilos tubería-filtros, donde los filtros no necesitan poseer información sobre los otros filtros que constituyen el sistema. 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. • se presentan problemas de efectos colaterales en cascada: si A usa B y C también lo usa, el efecto de C sobre B puede afectar a A.

Tabla A-15 Ventajas y Desventajas de la Arquitectura Orientada a Objetos

Estilo de Arquitectura Basada en Componentes

Una arquitectura basada en componentes describe una aproximación de ingeniería de software al diseño y desarrollo de un sistema. Esta arquitectura se enfoca en la descomposición del diseño en componentes funcionales o lógicos que expongan interfaces de comunicación bien definidas. Esto provee un nivel de abstracción mayor que los principios de orientación por objetos y no se enfoca en asuntos específicos de los objetos como los protocolos de comunicación y la forma como se comparte el estado.

El estilo de arquitectura basado en componentes tiene las siguientes características:

- Es un estilo de diseño para aplicaciones compuestas de componentes individuales.
- Pone énfasis en la descomposición del sistema en componentes lógicos o funcionales que tienen interfaces bien definidas.
- Define una aproximación de diseño que usa componentes discretos, los que se comunican a través de interfaces que contienen métodos, eventos y propiedades.

Principios Fundamentales

Un componente es un objeto de software específicamente diseñado para cumplir con cierto propósito. Los principios fundamentales cuando se diseña un componente es que estos deben ser:

- **Reusable**. Los componentes son usualmente diseñados para ser utilizados en escenarios diferentes por diferentes aplicaciones, sin embargo, algunos componentes pueden ser diseñados para tareas específicas.
- **Sin contexto especifico**. Los componentes son diseñados para operar en diferentes ambientes y contextos. Información específica como el estado de los datos deben ser pasadas al componente en vez de incluirlos o permitir al componente acceder a ellos.
- **Extensible**. Un componente puede ser extendido desde un componente existente para crear un nuevo comportamiento.
- **Encapsulado**. Los componentes exponen interfaces que permiten al programa usar su funcionalidad. Sin revelar detalles internos, detalles del proceso o estado.
- **Independiente**. Los Componentes están diseñados para tener una dependencia mínima de otros componentes. Por lo tanto los componentes pueden ser instalados en el ambiente adecuado sin afectar otros componentes o sistemas.

Beneficios

Los siguientes son los principales beneficios del estilo de arquitectura basado en componentes:

- **Facilidad de Instalación**. Cuando una nueva versión esté disponible, usted podrá reemplazar la versión existente sin impacto en otros componentes o el sistema como un todo.
- **Costos reducidos**. El uso de componentes de terceros permite distribuir el costo del desarrollo y del mantenimiento.
- **Facilidad de desarrollo**. Los componentes implementan un interface bien definida para proveer la funcionalidad definida permitiendo el desarrollo sin impactar otras partes del sistema.
- **Reusable**. El uso de componentes reutilizables significa que ellos pueden ser usados para distribuir el desarrollo y el mantenimiento entre múltiples aplicaciones y sistemas.
- **Mitigación de complejidad técnica**. Los componentes mitigan la complejidad por medio del uso de contenedores de componentes y sus servicios. Ejemplos de servicios de componentes incluyen activación de componentes, gestión de la vida de los componentes, gestión de colas de mensajes para métodos del componente y transacciones.

Ejemplos

Tipos comunes de componentes usados en aplicaciones incluyen:

- Componentes de interfaz de usuario, como grillas, botones, etc., generalmente conocidos como “controles”.
- Componentes de ayuda que exponen un conjunto específico de funciones usados por otros componentes.

- Componentes que se no se usan con mucha frecuencia o son intensivos en recursos y deben ser actividades usando una aproximación de solo en el momento justo (Just in Time (JIT)). Estos son comunes en escenarios de componentes distribuidos o en componentes remotos.
- Componentes encolados, aquellos cuyos métodos pueden ser ejecutados de forma asíncrona usando colas de mensajes del tipo almacenamiento, entrega.

Estilo de Pares (peer-to-peer style)

Está caracterizado por la interacción directa entre componentes de servicios de interacción de pares. La comunicación de pares es un tipo de interacción llamada y retorno sin la asimetría encontrada en el estilo cliente servidor. Esto significa que cualquier componente puede, en principio, interactuar con cualquier otro componente solicitando sus servicios. Así, los conectores en este estilo pueden invocar protocolos bidireccionales de interacción complejos, reflejando la comunicación en dos direcciones que puede existir entre dos o más componentes del estilo. Algunos ejemplos de sistemas de pares incluyen arquitecturas que están basadas en infraestructuras de objetos distribuidos como CORBA, COM+, y Java RMI (remote method invocation). En general, son vistas de arquitectura de sistemas de objetos en tiempo de ejecución, como los mostrados en diagramas de colaboración.

Elementos, Relaciones y Propiedades

La Tabla A-16 resume el estilo de pares. Aquí los componentes son pares que generalmente corresponden a programas independientes corriendo en nodos separados. Un conector típico es el conector de Llamada-Retorno. A diferencia del estilo cliente servidor, la interacción puede ser realizada por cualquier par; cada componente par puede ser cliente y servidor. Los pares tienen interfaces que describen los servicios que necesitan de otros pares y los servicios que proveen. La topología de los sistemas de este tipo es simétrica: los pares primero se conectan a la red de pares y luego inician acciones para realizar sus tareas cooperando con sus pares y solicitando servicios de otros.

Usualmente, una búsqueda de un par que busca a otros se propaga desde un par a sus pares conectados hasta un número limitado de saltos. Una arquitectura de pares puede tener nodos especiales (llamados ultrapares, ultranodos, o supernodos) que tienen capacidades de indexar o rutear y permiten búsquedas regulares de pares para llegar a un gran número de pares.

Las restricciones de uso de este estilo pueden limitar el número de pares que pueden estar conectados a un par dado o imponer restricciones sobre qué sabe un par de otro.

Elementos	• **Componente Par** • **Conector Llamada-Retorno**: es usado para conectar la red de pares, buscar otros pares, e invocar servicios de otros pares
Relaciones	La relación de **unión** asocia los pares con conectores Llamada-Retorno
Topología	El modelo se basa en la cooperación de los pares que solicitan servicios a otros pares.
Propiedades	Las mismas que las vistas de C&C, enfatizando en la interacción de protocolos y las propiedades asociadas al rendimiento. Los adjuntos pueden cambiar en tiempo de ejecución.
Restricciones	• Las restricciones pueden estar por el número de attachments de un puerto o rol dado. • Existen componentes especiales que pueden entregar capacidades de enrutamiento, indexación, y búsqueda de pares. • Existen especializaciones que pueden imponer restricciones de visibilidad en lo que algunos componentes pueden saber de otros
Para qué utilizarlo	• Entregar **disponibilidad** mejorada • Entregar **escalabilidad** mejorada • Permitir sistemas altamente distribuidos, como compartición de archivos, mensajería instantánea, y desktop grid computing [12].

Tabla A-16 Resumen del Estilo de Pares

¿Para qué es este estilo?

Los pares interactúan directamente entre ellos y pueden tener roles de consumidor y proveedor de servicios, asumiendo cualquier tarea que se necesite para realizar su trabajo. Esta división proporciona flexibilidad para la implementación de plataformas de sistemas altamente distribuidos. Los pares se pueden agregar y eliminar de la red de pares sin impactos significativos, permitiendo que el sistema tenga gran **escalabilidad**.

Usualmente existen sistemas con capacidades que se solapan, como la capacidad de acceso al mismo repositorio de datos. Por lo tanto, un par actuando como cliente puede colaborar con múltiples pares actuando como servidores para completar ciertas tareas. Si uno de estos pares llega a estar no disponible, los otros pueden seguir entregando el servicio para completar la tarea. El resultado de esto es un mejoramiento global de la disponibilidad. La carga en cualquier componente par actuando como un servidor se reduce, y las responsabilidades que pudiera haber requerido, más capacidad de servidor y la infraestructura para apoyarla, se distribuye. Esto puede reducir la necesidad para otro tipo de comunicación de actualización de datos y de almacenamiento en un servidor central, pero todo a expensas de almacenar los datos localmente.

El estilo de pares generalmente es usado para aplicaciones distribuidas, como para compartir archivos, mensajería instantánea, etc. Con el uso de un despliegue adecuado, las aplicaciones pueden hacer un uso más eficiente de CPU y recursos de disco a través de la distribución de tareas intensivas en cálculo a través de la red de computadores y tomando la ventaja de los recursos locales disponibles para los clientes. Los resultados pueden ser distribuidos entre los pares participantes.

Relación con otros estilos

La inexistencia de una jerarquía hace que los sistemas que están diseñados con el estilo de pares tengan una topología más general que los sistemas con una arquitectura cliente servidor.

Especializaciones del estilo

En el estilo de pares generalmente se encuentran las siguientes especializaciones:

- Arquitectura Orientada a Servicios
- Arquitectura Basada en Eventos
- Arquitectura Basada en Recursos

Estilo de Arquitectura Orientada a Servicios

Las arquitecturas orientadas a servicios (SOA) consisten en una colección de componentes distribuidos que entregan y/o consumen servicios. En SOA, los componentes proveedores y consumidores de servicios pueden utilizar diferentes lenguajes de implementación y plataformas. Los servicios son proveedores de servicios independientes y los consumidores están desplegados de forma independiente, y, generalmente, pertenecen a diferentes sistemas o incluso a diferentes organizaciones.

Elementos, Relaciones y Propiedades

La Tabla A-17 resume el estilo SOA. En él, los tipos de componentes básicos son los consumidores y los proveedores de servicios, los que en la práctica pueden tomar diferentes formas, desde Javascript ejecutados en un browser hasta transacciones CICS que corren en un mainframe.

Además de los consumidores y los proveedores de servicios, una SOA puede utilizar componentes especializados que actúan como intermediarios y proveen servicios de infraestructura como el Bus de Servicios Empresariales, el Registro de Servicios y el Servidor de Orquestación.

Elementos	• **Proveedores de Servicios**: proveen uno o más servicios a través de interfaces públicas. Sus propiedades varían con la implementación tecnológica (como EJB o ASP.NET) pero pueden incluir restricciones de performance, autorización, disponibilidad y costo. En algunos casos estas propiedades son especificadas en los niveles de servicios (SLA). • **Consumidores de Servicios**: invocan servicios directamente o a través de un intermediario. • **Bus de Servicios Empresarial (ESB)**: es un elemento intermediario que puede rutear y transformar mensajes entre proveedores y consumidores de servicios. • **Registro de Servicios**: puede ser usado por los proveedores de servicios para registrar sus servicios y por los consumidores para consultar y descubrir servicios en tiempo de ejecución. • **Servidor de Orquestación**: coordina las interacciones entre los consumidores y proveedores de servicios basado en scripts que definen workflows de negocio. Generalmente se utiliza un servidor BPEL. • **Conector SOAP**: usa el protocolo SOAP para la comunicación sincrónica entre servicios web, generalmente utilizando HTTP. Los puertos de los componentes que usan SOAP generalmente son descritos en WSDL. • **Conector REST**: se basa en las operaciones básicas de llamada y retorno del protocolo HTTP (GET, PUT, POST y DELETE). • **Conector de Mensajes**: usa un sistema de mensajería para ofrecer intercambios de mensajes asíncronos (punto a punto o publicación y suscripción).
Relaciones	Las uniones de los diferentes tipos de puertos disponibles a sus respectivos conectores.
Topología	Los cálculos son realizados por un conjunto de componentes cooperativos que entregan o consumen servicios sobre la red. Algunas veces los cálculos son descritos como un tipo de modelo de workflow.
Restricciones	• Los consumidores se conectan a los proveedores de servicios, aunque se pueden usar componentes que cumplen la función de intermediarios (como un ESB, Registro de Servicios, o un servidor BPEL). • Cuando se utiliza un ESB el modelo se convierte en una topología Hub and Spoke. • Los proveedores de servicios también pueden ser consumidores de servicios.
Para qué utilizarlo	• Una SOA permite la interoperatividad de componentes distribuidos ejecutados en diferentes plataformas o a través de internet. • La integración de sistemas legados. • La reconfiguración dinámica.

Tabla A-17 Resumen del Estilo de Arquitectura Orientado a Servicios

¿Para qué es este estilo?

El principal beneficio y el mayor uso de SOA es la interoperabilidad, debido a que los proveedores y los consumidores de servicios pueden ser ejecutados en plataformas diferentes, generalmente las arquitecturas orientadas a servicios ayudan a integrar diferentes sistemas e incluso sistemas legacy. Además, las arquitecturas orientadas a servicios ofrecen los elementos necesarios para interactuar con servicios disponibles en internet. También, el uso de componentes especiales de SOA como el registro de servicios o el ESB que permiten reconfiguraciones dinámicas, que son útiles cuando existe la necesidad de reemplazar o agregar versiones de componentes sin interrumpir el uso de los sistemas.

1.2.2.3. Estilos de Basados en Eventos

Los estilos basados en eventos permiten comunicar componentes a través de mensajes asíncronos. Tales sistemas generalmente se organizan como una distribución pobremente acoplada de componentes que generan comportamientos en otros componentes a través de eventos.

Existe una variedad de estilos basados en eventos. En algunos casos utilizan conectores punto-a-punto para comunicar mensajes de forma similar a los conectores de consulta-respuesta, pero permitiendo mayor concurrencia, debido a que el emisor del evento no necesita bloquearse mientras los receptores procesan el evento. En otros casos, los conectores tienen múltiples destinos, permitiendo que un evento sea procesado por múltiples componentes, a estos sistemas se les llama sistemas de publicación-suscripción, donde el que anuncia la ocurrencia del evento es visto como el publicador del evento al que están suscritos los receptores.

Estilo Publicación y Suscripción (publish-subscribe style)

Está caracterizado por componentes que interactúan anunciando eventos a componentes que previamente se suscriben a ellos. Generalmente, este estilo es utilizado para desacoplar productores y consumidores de mensajes. Este desacoplamiento aplaza la unión de productores y consumidores de mensajes hasta el tiempo de ejecución y, por lo tanto, soporta la modificación de esos productores y consumidores.

Elementos, Relaciones y Propiedades

La Tabla A-18 muestra un resumen del estilo de publicación y suscripción. Este estilo puede implementarse de múltiples formas. Una de las más comunes es la llamada invocaciones implícitas, donde los componentes tienen interfaces procedurales, y un componente registra un evento asociando uno de sus procedimientos a cada tipo de evento suscrito. Cuando un evento es anunciado, los procedimientos asociados de los componentes suscritos son invocados en un orden generalmente determinado por la infraestructura de hardware. Los frameworks de interfaces gráficas de usuario, como en Visual Basic generalmente son controlados por invocaciones implícitas, donde los fragmentos de código son asociados a eventos predefinidos como por ejemplos los clicks del mouse.

En otras formas del estilo de publicación y suscripción, los eventos son ruteados al componente apropiado. Es tarea del componente saber cómo controlar el evento. Tales sistemas ponen más carga en los componentes individuales para controlar los flujos de eventos, pero además permite una mezcla más heterogénea de componentes que el estilo de invocaciones implícitas.

En algunos sistemas de publicación y suscripción, un anunciador de eventos puede bloquearse hasta que el evento sea procesado completamente por el sistema. Por ejemplo, algunas interfaces de usuario requieren que todas las vistan se actualicen cuando cambian los datos que representan.

Esto es realizado forzando que el componente que anuncia el evento “cambio de datos” se bloquee hasta que todas las vistas suscritas hayan sido notificadas.

Elementos	• Cualquier componente C&C con al menos un puerto de publicación o suscripción. Sus propiedades varían, pero no debería incluirse que eventos son anunciados y/o suscritos a él, y las condiciones sobre las cuales un anunciante es bloqueado. • **Conector publicación-suscripción**, que anunciará y escuchará funciones de los componentes que desea publicar y/o suscribir eventos.
Relaciones	La relación de **unión** asocia componentes con el conector publicación-suscripción indicando que componentes anuncian eventos y qué componentes han sido registrados para recibir eventos.
Topología	Los componentes se suscriben a eventos. Cuando un evento es anunciado por un componente, el conector entrega el evento a todos sus suscriptores.
Restricciones	• Todos los componentes son conectados a un distribuidor de eventos que puede ser visto como un bus –por ejemplo, un conector –o un componente
Para qué utilizarlo	• Para enviar eventos a un número indeterminado de destinatarios, aislando los productores de eventos de sus consumidores. • Para entregar las funcionalidades principales para frameworks de interfaz gráfica, lista de envío de emails, pizarras de publicación de noticias, y redes sociales.

Tabla A-18 Resumen del Estilo de Publicación y Suscripción

¿Para qué es este estilo?

El estilo de publicación y suscripción es utilizado para enviar eventos y mensajes a un número indeterminado de destinatarios. Debido a que el conjunto de receptores de eventos es desconocido para el productor de eventos, la correctitud del productor de eventos no depende de sus destinatarios. Por lo tanto, es posible agregar nuevos destinatarios sin modificar al productor de eventos.

El estilo de publicación y suscripción generalmente es usado para desacoplar las interfaces de usuario de las aplicaciones. También puede ser usado para integrar herramientas en ambientes de desarrollo de aplicaciones: anunciando eventos que permiten invocar a otras herramientas. Otras aplicaciones incluyen sistemas como pizarras de noticias, redes sociales, y listas de mensajes, donde los cambios dinámicos de conjuntos de usuarios son notificados cuando el contenido que están utilizando es modificado.

Relación con otros estilos

El estilo publicación y suscripción es similar al estilo repositorio pizarra, debido a que en ambos estilos los componentes son notificados de cambios en otros componentes. Aunque, en un sistema pizarra, la base de datos es el único componente que genera tales eventos; en un sistema de publicación y suscripción, cualquier componente puede generar eventos.

La invocación implícita generalmente es combinada con llamada-retorno en sistemas cuyos componentes pueden interactuar tanto sincrónicamente por una invocación de servicio o asincrónicamente a través del anuncio de eventos. Por ejemplo, muchas arquitecturas orientadas a

servicios y sistemas de objetos distribuidos (como CORVA y Java EE) permiten comunicación síncrona y asíncrona. En otros sistemas basados en objetos, se utilizan llamadas a procedimientos síncronos para lograr interacciones asincrónicas usando el patrón MVC o el patrón Observador.

1.2.2.4. Estilos de Repositorio o Centrados en Datos

Esta categoría de estilos enfatiza la integrabilidad de los datos. Se estima apropiada para sistemas que se fundan en acceso y actualización de datos en estructuras de almacenamiento. Sub-estilos característicos de la familia serían los repositorios, las bases de datos, las arquitecturas basadas en hipertextos y las arquitecturas de pizarra.

Estilo de Datos Compartidos (shared data style)

El estilo de datos compartidos también es conocido como estilo de pizarra. En este estilo hay dos componentes principales: una estructura de datos que representa el estado actual y una colección de componentes independientes que operan sobre él. En base a esta distinción se han definidos dos sub-categorías principales del estilo:

1. Si los tipos de transacciones en el flujo de entrada definen los procesos a ejecutar, el repositorio puede ser una base de datos tradicional (implícitamente no cliente-servidor).
2. Si el estado actual de la estructura de datos dispara los procesos a ejecutar, el repositorio es lo que se llama una pizarra pura o un tablero de control.

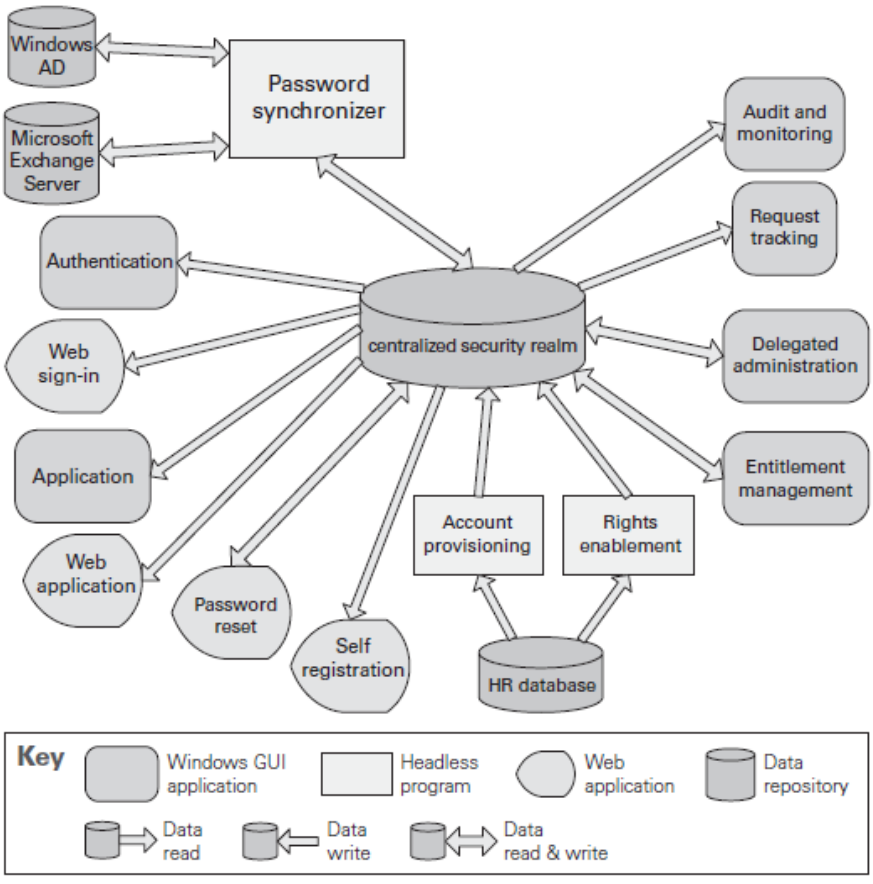


Figura A-42 Diagrama de datos compartidos de un sistema de administración de acceso

Los sistemas de bases de datos y sistemas basados en conocimiento son ejemplos de estilos de datos compartidos. Una característica de un estilo de datos compartidos es cómo el consumidor de datos puede descubrir que los datos de interés están disponibles. Estos sistemas se usan en aplicaciones que requieren procesos de interpretación de señales complejas (como reconocimiento de patrones, reconocimiento del habla, etc.), o en sistemas que involucran acceso compartido a datos con agentes débilmente acoplados. También se han implementado estilos de este tipo en procesos batch de base de datos y ambientes de programación organizados como colecciones de herramientas en torno a un repositorio común. Algunas arquitecturas de compiladores que suelen presentarse como representativas del estilo tuberías y filtros, se podrían representar mejor como propias del estilo de datos compartidos, dado que muchos compiladores contemporáneos operan en base a información compartida tal como tablas de símbolos, árboles sintácticos abstractos (AST), etc.

Elementos, Relaciones y Propiedades

El estilo de datos compartidos está organizado alrededor de uno o más almacenes de datos compartidos, las cuales almacenan los datos que otros componentes podrán leer o escribir. Estos tipos de componentes pueden ser almacenes de datos compartidos y de acceso a datos. El tipo de conector es el lector y escritor de datos. El modelo de proceso asociado a los sistemas de datos compartidos es que los componentes de acceso a datos realizan operaciones que requieren datos del almacén de datos y escriben resultados a uno o más almacenes de datos. Los datos pueden ser vistos y actualizados por otros componentes de acceso a datos. En un sistema de datos compartidos puro, los componentes de acceso a datos interactúan sólo a través de uno o más almacenes de datos compartidos. Aunque en la práctica los sistemas de datos compartidos también permiten interacciones directas entre componentes de acceso a datos. Los componentes de los almacenes de datos de un sistema de datos compartido permiten el acceso compartido de datos, la persistencia de datos, controlan el acceso concurrente a los datos a través de la administración de transacciones, entregan tolerancia a fallas, ayudan al control de acceso, y manejan la distribución y el cache de datos.

Las especializaciones del estilo de datos compartido se diferencian en relación a la naturaleza de los datos almacenados: los enfoques existentes incluye estructuras relacionales, de objetos, de capas, y jerárquicas.

La Tabla A-19 resume las características del estilo de datos compartidos.

Elementos	• Componente Repositorio, sus propiedades incluyen tipos de datos almacenados, propiedades orientadas al rendimiento de los datos, distribución de datos, número de componentes de acceso a datos permitidos. • Componente de acceso a datos. • Conector de lectura y escritura de datos, una propiedad importante es cuando el conector es transaccional o no.
Relaciones	La relación de unión determina que componentes de acceso a datos están conectados a cada repositorio.
Topología	La comunicación entre los componentes de acceso a datos es mediada por un almacén de datos compartido. El control puede se iniciado por los componentes de acceso a datos o el almacén de datos. Los datos se hacen persistentes a través del almacén de datos
Restricciones	Los componentes de acceso a datos interactúan a través del almacén de datos.
Para qué utilizarlo	• Permitir que múltiples componentes accedan a datos persistentes • Entregar la modificabilidad mejorada desacoplando productores de consumidores.

Tabla A-19 Resumen del estilo de datos compartido

¿Para qué es este estilo?

El estilo de datos compartidos es útil cuando varios ítems de datos tienen múltiples consumidores y persistencia. El uso de este estilo desacopla productores de datos de sus consumidores, por lo tanto, este estilo permite la modificabilidad debido a que los productores no tienen conocimiento directo de los consumidores.

Los análisis asociados al uso de este estilo generalmente se centran en las cualidades de rendimiento, seguridad, privacidad, disponibilidad, escalabilidad, y compatibilidad, por ejemplo, de los repositorios existentes y sus datos. Particularmente, cuando un sistema tiene más de un repositorio de datos, la principal preocupación de arquitectura es el mapeo de los datos y su procesamiento. El uso de múltiples almacenes de datos puede ocurrir debido a que naturalmente, o históricamente, los datos se particionan en almacenes separados. En otros casos los datos pueden ser replicados en muchos almacenes para mejorar el rendimiento y/o la disponibilidad a través de la redundancia. Esas elecciones pueden afectar fuertemente las cualidades indicadas anteriormente.

Relación con otros estilos

Este estilo tiene aspectos comunes con el estilo cliente-servidor, especialmente con el cliente-servidor multinivel. En aplicaciones de gestión de información que utilizan este estilo, el repositorio generalmente es una base de datos relacional, entregando consultas relacionales y actualizaciones utilizando interacciones cliente-servidor. Los clientes de la base de datos relacional (ósea los consumidores) se conectan al DBMS (de sus iníciales en inglés, Database Management System) utilizando un puerto de red y un protocolo especificado por el DBMS. Un módulo puente, incluido en los componentes clientes, entrega las operaciones de base de datos.

El estilo de datos compartidos está estrechamente relacionado al estilo de modelo de datos. Donde una vista de datos compartidos de un sistema representa los repositorios de datos y sus consumidores, el modelo de datos muestra cómo se estructuran los datos en los repositorios, en términos de entidades de datos y sus relaciones.
Es parecido a otros estilos de C&C, el estilo de datos compartidos está relacionado al estilo de distribución. Muy frecuentemente los sistemas que tienen repositorios de datos compartidos son aplicaciones distribuidas donde existe uno o más servidores que contienen repositorios de datos. Una vista de despliegue de un sistema muestra la ubicación de los repositorios y otros componentes en los nodos de la infraestructura de hardware.

1.2.2.5. Estilos Heterogéneos de C&C

Estilo de Comunicación de Procesos (communicating-processes style)
Se distingue por la interacción de componentes ejecutándose concurrentemente a través de varios mecanismos de conexión. Algunos ejemplos de mecanismos de conexión son: sincronización, paso de mensajes, intercambio de datos, partir, detener, and así sucesivamente. La comunicación entre procesos existe comúnmente en grandes sistemas y es necesaria en todos los sistemas distribuidos. Por lo tanto, para la mayoría de sistemas, el estilo de comunicación de procesos es apropiado para entender cualquier comportamiento asociado con concurrencia.

Especializaciones del estilo
En el estilo de comunicación de procesos generalmente se encuentran las siguientes especializaciones:

- Sistemas de Control de Procesos
- Arquitecturas Basadas en Atributos

1.2.3. ¿Para qué es el Tipo de Vista de Componentes y Conectores?

Las vistas de componentes y conectores son utilizadas para mostrar a los desarrolladores y a otros stakeholders como funciona un sistema. Las vistas de C&C (con su documentación de comportamiento asociada) especifican las estructuras y comportamientos de los elementos en tiempo de ejecución. En particular esas vistas permiten responder preguntas como las siguientes:

- ¿Cuáles son los componentes en ejecución principales del sistema, y cómo interactúan?
- ¿Cuáles son los principales almacenes de datos compartidos?
- ¿Qué partes del sistema están replicadas, y cuantas veces?
- ¿Cómo progresan los datos mientras se ejecuta el sistema?
- ¿Qué protocolos de interacción son utilizados para comunicar entidades?
- ¿Qué partes del sistema funcionan en paralelo?
- ¿Cómo cambia la estructura del sistema mientras este se ejecuta?

Las vistas de componentes y conectores también son usadas para analizar los atributos de calidad del sistema, como el **performance**, **confiabilidad**, y **disponibilidad**. En particular, una vista bien

documentada permite a los arquitectos predecir las propiedades generales del sistema, dando estimaciones o mediciones de propiedades de los elementos individuales e interacciones. Por ejemplo, para determinar si un sistema puede satisfacer los requerimientos de tiempo-real planificados, generalmente se debe conocer el tiempo de ejecución de cada componente de proceso (entre otras cosas). El comportamiento en el tiempo de cómo este representaría propiedades de elementos similares. También, el documentar la **confiabilidad** de elementos individuales y sus canales de comunicación, ayuda a un arquitecto cuando estima o calcula la **confiabilidad total** de un sistema. En algunos casos, tales análisis son facilitados por modelos formales y herramientas de análisis. En otros, se logra utilizando juicio experto basado en reglas generales y la experiencia.

1.2.4. Elección de la Abstracción de los Componentes

Si se elige un estilo particular de C&C, lo más probable es que los conectores que se deben utilizar para documentarlo ya estén definidos. Sin embargo, existen casos en que un arquitecto tiene la libertad para determinar los tipos de conectores a utilizar y cómo representarlos en la documentación. Esta opción generalmente se resuelve dependiendo de cuantas estructuras de implementación se deben exponer. Por un lado, un conector puede ser usado para encapsular una interacción compleja como una abstracción individual, y, por otro lado, se puede representar una forma de interacción compleja como un conjunto de conectores que se deben implementar.
Como ilustración, se pueden considerar dos formas de documentar un sistema de publicación y suscripción como el mostrado en la Figura A-43. La primera versión muestra cinco componentes comunicándose a través de un bus de eventos, el que describe una interacción que asegura que cada evento publicado es entregado a todos los suscriptores de ese evento. La segunda versión muestra los mismos cinco componentes comunicándose con la asistencia de un componente despachador responsable de la distribución de eventos que llaman a procedimientos en los otros componentes.
Hay varias ventajas de utilizar la primera versión:

- Simplifica la descripción dado que hay menos elementos en la vista.
- Claramente distingue las partes de la arquitectura que son utilizadas para la interacción (los conectores) y las partes que son usadas para entregar las funciones del sistema (los componentes).
- Permite utilizar una variedad de implementaciones que ayuden a ejecutar interacciones basadas en eventos. Por ejemplo, en lugar de un despachador simple, pueden ser varios, o alternativamente cada componente puede ser responsable de enviar sus eventos a los listeners solicitados.
- Entrega una forma natural para descomponer la documentación en múltiples vistas, donde la implementación específica será representada en su vista propia como un refinamiento del conector del bus de eventos.

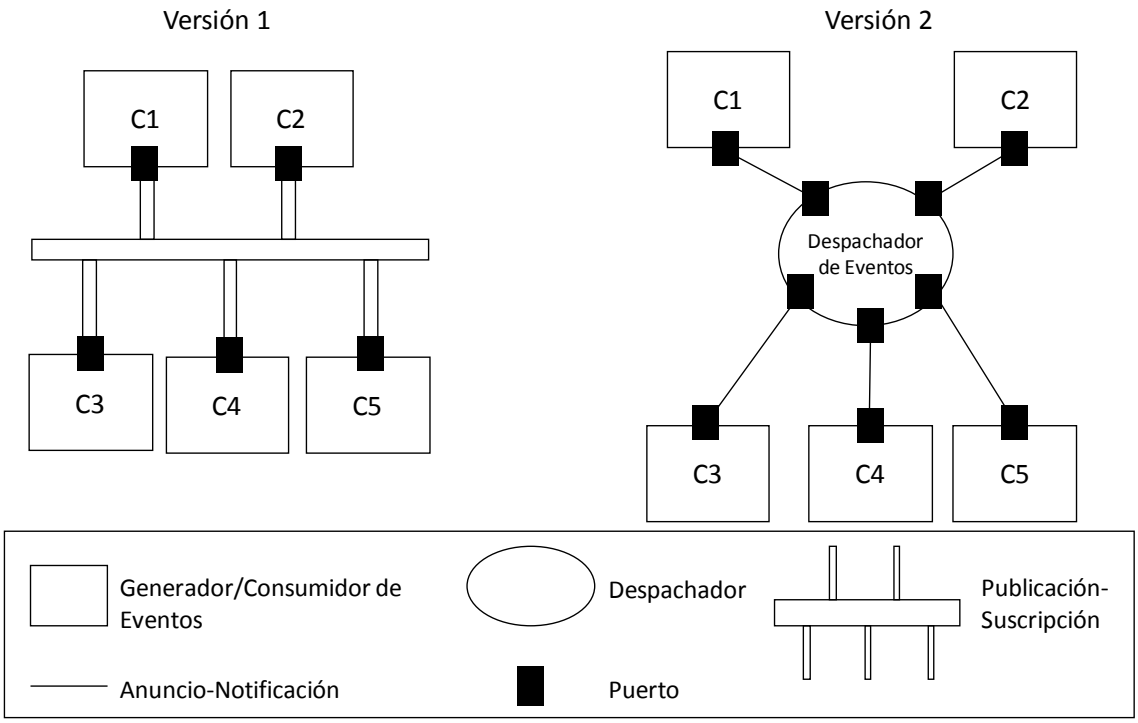


Figura A-43 Dos versiones potenciales de un sistema de publicación-suscripción

Por otro lado, la segunda representación tiene algunas ventajas:

- Claramente indica qué tipo de mecanismos están siendo utilizados para realizar la distribución de eventos.
- Es mejor para ayudar al análisis de sus propiedades en tiempo de ejecución, como retrasos, garantizar el orden, y cosas así, donde es necesario el conocimiento de los mecanismos específicos para la distribución.
- Encaja con lo que permite la notación elegida: por ejemplo, dado que UML no permite una forma de representar conectores enriquecidos, se fuerza a utilizar el segundo enfoque.

Por lo tanto la elección de la abstracción de un conector dependerá del gusto, necesidades de análisis, y la cantidad de detalle de implementación conocida del arquitecto cuando se documenta la arquitectura. En la práctica, sin embargo, la documentación generalmente se equivoca al poner demasiado detalle, utilizando mecanismos de comunicación de bajo nivel y componentes adicionales en lugar de definir abstracciones que representan interacciones de alto nivel.

1.2.5. Notaciones para el Tipo de Vista de C&C

1.2.5.1. Notaciones Informales

Como siempre, están disponibles las cajas y líneas para representar las vistas de C&C. La Figura A-34 es un ejemplo de un diagrama de C&C que utiliza una notación informal (explicado en la parte inferior del diagrama). A pesar de que las notaciones informales pueden transmitir una semántica

limitada, siguiendo algunas pautas pueden aportar el rigor y la profundidad de las descripciones. La directriz principal es asignar a cada tipo de componente y cada tipo de conector una forma visual (símbolo), y listar cada uno de los tipos con una clave.
Más allá de nombrar tipos, sin embargo, se debe especificar su significado. Por ejemplo, la Figura A-34 muestra un conector del tipo publicar-suscribir, pero el diagrama no muestra la capacidad del conector, el tipo de datos que puede transmitir, o no se garantiza la entrega, o una serie de otras importantes consideraciones. Esos detalles pueden ser documentados en la guía de estilos en la que se define el tipo, o como propiedades en el catálogo de elementos de la vista de C&C.
Se debe tener un cuidado especial con los conectores. Un origen común de ambigüedad en la mayoría de documentos de arquitectura existentes es el significado de conectores, especialmente aquellos que utilizan flechas en su símbolo visual. Se debe asegurar de especificar qué significa la dirección de la flecha.

1.2.5.2. Notaciones Formales

La mayoría, por no decir todos, de los lenguajes de descripción de arquitecturas (ADLs, de sus siglas en inglés) pueden ser usados para describir tipos de componentes y conectores, restricciones sobre las topologías de los gráficos de componentes y conectores, y propiedades que pueden ser asociadas con los elementos del gráfico. Las herramientas permiten procesar una descripción de arquitectura al referirse a los significados de los tipos, restricciones, y las propiedades. Por ejemplo, algunas herramientas asociadas a ADL pueden decir si un conjunto de procesos pueden programarse para que, dados los recursos de CPU, todos ellos cumplan los plazos de procesamiento.

1.2.5.3. Notaciones Semi-Formales: UML

A continuación se presentan algunas nociones básicas para representar componentes y conectores utilizando UML.

Componentes en UML

Los componentes de UML poseen una buena semántica para representar componentes de C&C ya que permiten una documentación intuitiva de la información importante como interfaces, propiedades, y descripción de comportamiento. Los componentes de UML además distinguen entre tipos de componentes e instancias de componentes, los que son útiles cuando se definen tipos de componentes de una vista específica.
Debido a que los componentes de C&C que aparecen en una vista son instancias, deben ser representados utilizando instancias de componentes UML como se muestra en la Figura A-44. La distinción visual entre los tipos de componentes UML y las instancias se encuentran en las convenciones de nombres. Los nombres que no incluyen dos puntos (:) son tipos; los nombres que incluyen dos puntos son instancias, con el nombre de la instancia a la izquierda de los dos puntos. Las instancias anónimas como las instancias de la base de datos Account en la Figura A-44, son mostradas con un punto y coma iniciando el nombre con dos puntos.

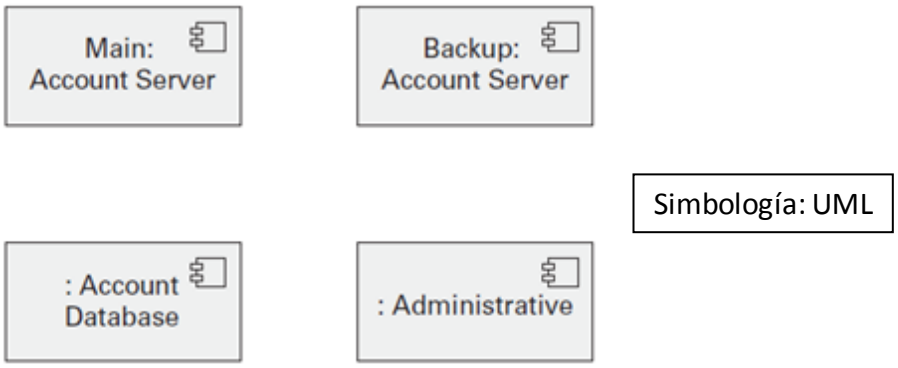


Figura A-44 Una representación UML de una porción de vista C&C

Se puede definir un tipo de componente en un diagrama UML en su guía de estilos o en un catálogo de elementos de la vista para un tipo específico de la vista. Se debe especificar atributos comunes a todas las instancias en el tipo de componente. Si se crea un tipo específico de la vista, se debe enlazar la definición de tipo a un tipo definido en la guía de estilo, como en la definición de un estereotipo en la definición de tipo, como se muestra en la Figura A-45 Una representación UML de un tipo de componente C&C.

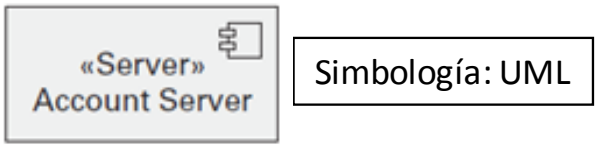


Figura A-45 Una representación UML de un tipo de componente C&C

Los puertos UML son una buena combinación semántica de los puertos C&C. Un puerto UML puede ser decorado con una multiplicidad, como se muestra en el sector izquierdo de la Figura A-46 Representación UML de puertos en un tipo de componente C&C (izquierda) y una instancia de componente (derecha), aunque esto suele hacerse en los tipos de componentes. El número de puertos en instancias de componentes, como muestra el sector derecho de la Figura A-46, se suele enlazar a un número específico. Los componentes que crean y administran dinámicamente un conjunto de puertos deben mantener un descriptor de la multiplicidad en las descripciones de instancias.

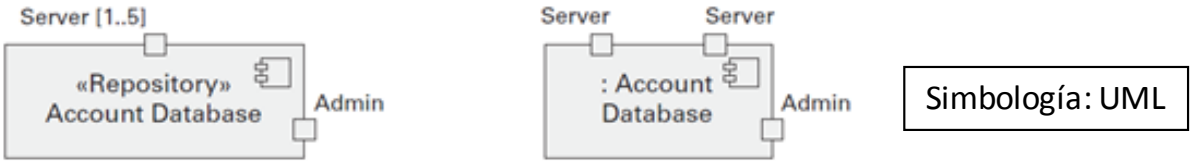


Figura A-46 Representación UML de puertos en un tipo de componente C&C (izquierda) y una instancia de componente (derecha)

UML proporciona una notación lollipop/socket para mostrar las interfaces provistas y/o requeridas adjuntas a puertos. Cada puerto puede tener un número arbitrario de interfaces provistas y requeridas. La Figura A-47 muestra los mismos componentes de la Figura A-46, pero ahora cada puerto del tipo de base de datos Account incluye una interfaz provista (lollipop) que puede seguir

desarrollándose en UML mediante la entrega de información adicional, como métodos y atributos. La instancia de la base de datos Account en el lado derecho tiene exactamente dos puertos servidores, y las interfaces son omitidas.

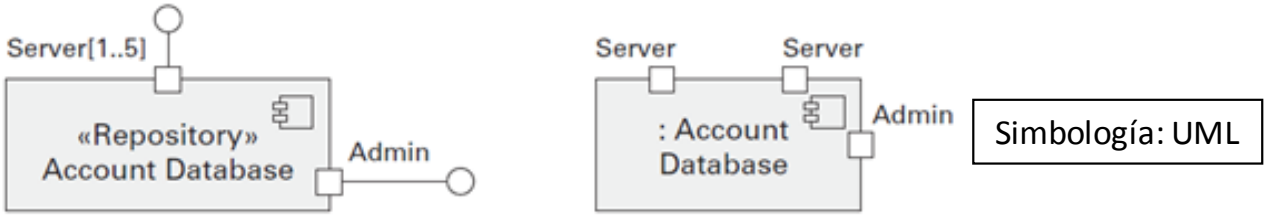


Figura A-47 Cada puerto en el tipo de base de datos Account ahora incluye una interfaz provista (lollipop)

La notación lollipop/socket de UML puede ser confundida si no se utiliza cuidadosamente. Si el estilo de interacción de conector es una forma de llamada-respuesta, entonces el lollipop y el socket corresponderá a las llamadas que son ofrecidas y requeridas, respectivamente. En un conector cliente-servidor, un simple puerto puede ofrecer y requerir algo al mismo tiempo, en cuyo caso se debería adornar el mismo puerto con ambos un lollipop y un socket. Pero en otros casos, donde ofrecer y requerir pueden malentenderse, la notación debe ser evitada. En un sistema de tubos y filtros, por ejemplo, que hace una interfaz de filtro “ofrece” y que es lo que “requiere”. En ese caso, el puerto se documenta por sí mismo.

Aun cuando sea aprobado, normalmente se omiten los lollipop/socket desde la vista de C&C (la que muestra instancias) y sólo se usan en las definiciones de tipo de componente. Generalmente, los detalles completos de las interfaces se entregan en las definiciones de tipo, y en la presentación principal sólo se mostrarán los puertos. Esto reduce el desorden visual sin perder las definiciones de interfaces de las instancias.

Conectores en UML

Mientras que los conectores de C&C son ricos semánticamente como los componentes de C&C, esto no es cierto para los conectores en UML. Los conectores en UML no pueden tener sub-estructuras, atributos, o descripciones de comportamientos. Esto hace que la elección de cómo representar los C&C sea más difícil, debido a que los conectores UML no son lo suficientemente ricos.

Se debe representar un conector C&C simple usando un conector UML –una línea recta. Los conectores C&C más comúnmente utilizados son bien conocidos, semánticas e implementaciones independientes de aplicaciones, tales como llamadas a funciones u operaciones de lectura de datos. Si la única información que se debe entregar es el tipo de conector, entonces un conector UML es el adecuado. Los conectores llamada-respuesta (call-return) pueden ser representados por un conector de ensamblado UML, el cual une la interfaz requerida de un componente (socket) a otra interfaz provista por un componente (lollipop). Se puede utilizar un estereotipo para indicar el tipo de conector. Si todos los conectores en la presentación principal son del mismo tipo, se puede indicar una vez con un comentario en lugar de hacerlo explícitamente en cada conector, para reducir el desorden visual. Un adjunto se muestra al conectar los extremos del conector a los puertos de componentes. La Figura A-48 Una representación UML de un simple conector C&C entre dos componentes ilustra algunos de estos puntos.

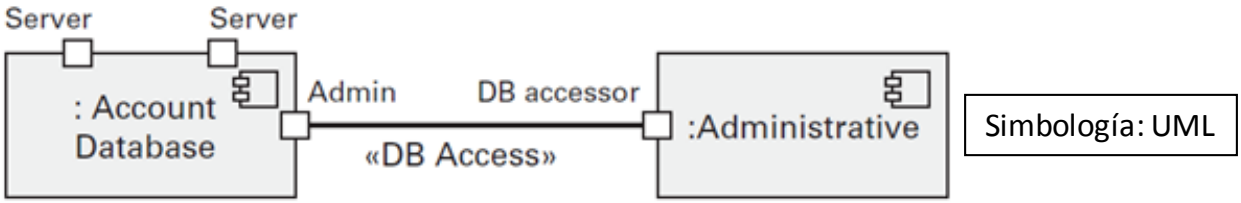


Figura A-48 Una representación UML de un simple conector C&C entre dos componentes

Los roles de un conector no pueden ser representados explícitamente con un conector UML debido a que el elemento conector UML no permite la inclusión de interfaces (a diferencia de los puertos UML, que sí permiten interfaces). La mejor aproximación es etiquetar los extremos del conector y utilizar esas etiquetas para identificar las descripciones de funciones que deben ser documentadas en alguna parte.

Si se necesita además entregar información descriptiva simple, como pares atributo-valor, se debe adjuntar a un conector UML mediante el uso de valores etiquetados o un comentario.

Para representar un conector de C&C enriquecido en UML se puede utilizar un componente o también utilizando una anotación con un conector UML con un tag u otra documentación auxiliar que explique el significado del conector complejo.

La Figura A-49 muestra un ejemplo de representación de un conector C&C utilizando un componente UML. En este enfoque, las funciones son representadas utilizando puertos UML. Las relaciones adjuntas son representadas a través de la unión de los puertos UML de los componentes y el conector utilizando un conector UML. Aunque no es ideal el utilizar la misma convención gráfica que utiliza un componente C&C, a veces es necesario realizarlo con UML.

A veces es mejor utilizar una línea recta (posiblemente estereotipada) con un tag que explique el conector complejo. Por ejemplo, si existen diez clientes, cada uno de los cuales está hablando a algún servidor con el mismo protocolo asíncrono no-trivial. El introducir diez componentes extra provocaría mucho desorden, en cambio, el utilizar un conector de línea recta estereotipada ayudaría a la claridad del diagrama.

Una presentación principal de C&C en UML

La presentación principal de C&C mostrada en la Figura A-49 es un ejemplo de una vista combinada que combina estilos cliente-servidor, publish-subscribe, y shared-data. Las Figura A-49 y la Figura A-50 muestran como representar la misma información con UML.

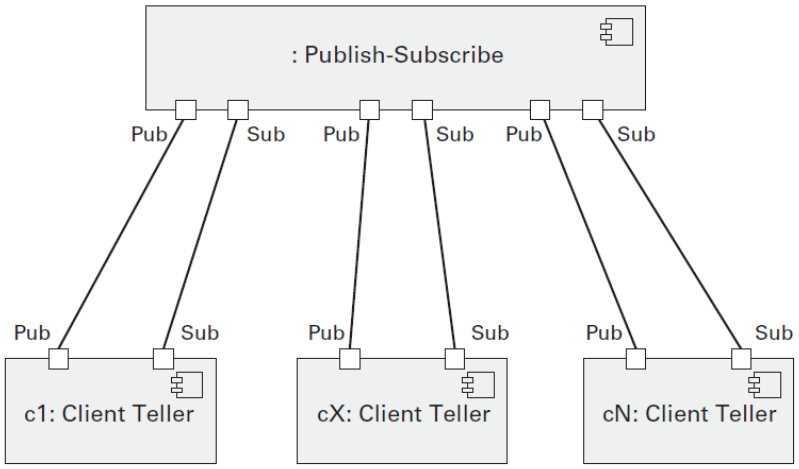


Figura A-49 Representación de un Conector UML enriquecido utilizado para conectar tres componentes

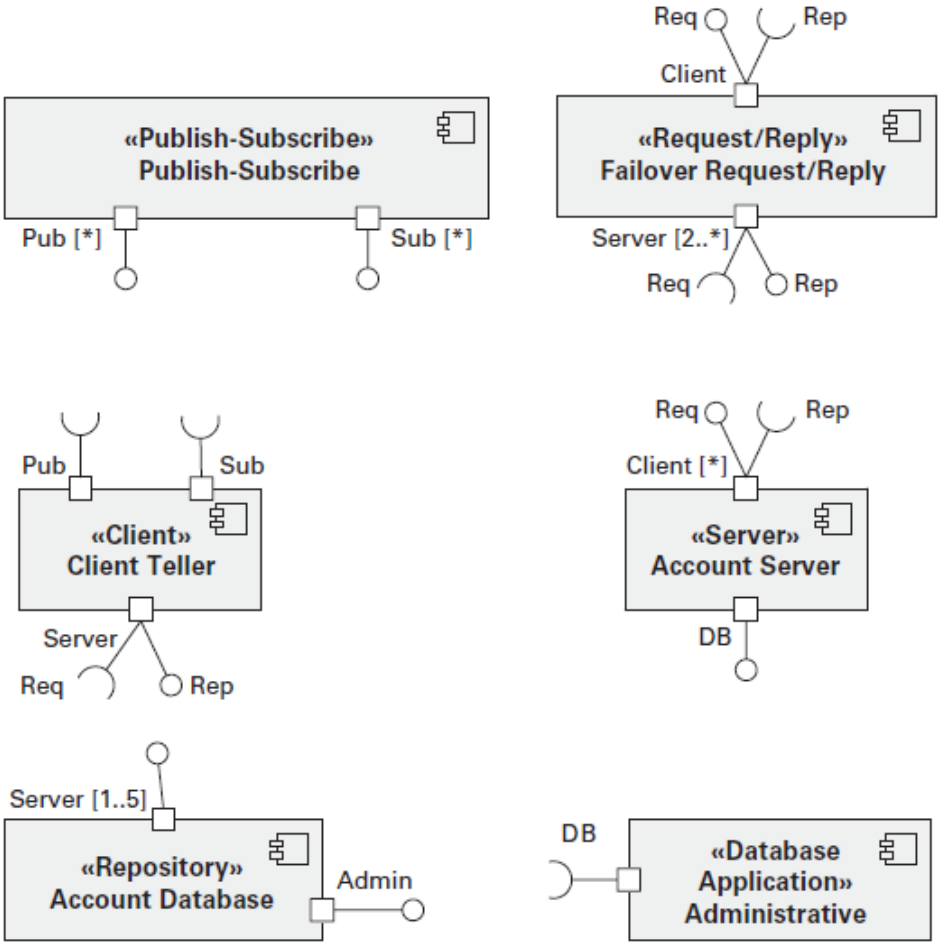


Figura A-50 Una representación de tipos de componentes y conectores en UML

La Figura A-49 define los subtipos de componentes y conectores que hay en una vista específica. Cada tipo utiliza un estereotipo UML para identificar los tipos de componentes y conectores definidos correspondientes en una de las tres guías de estilos citados. Las multiplicidades son adjuntadas a alguno de los puertos para indicar donde están permitidas múltiples conexiones y para establecer límites en el número de conexiones. Esta información debe estar en el catálogo de elementos de la vista.

La Figura A-51 muestra la presentación principal de una vista, como se representa en UML. Como el conector de publicación-suscripción, el conector Failover Request/Reply está representado usando un componente UML; esto permite que los detalles de la semántica del failover sean formalmente documentados, y facilita la representación de un conector n-ario.

Además de los avances presentados en la representación de conceptos básicos en UML de C&C, se debe decidir cómo representar la variabilidad de la Figura A-49. Esa figura da la impresión de un número variable de componentes clientes teller, cualquiera de los cuales puede estar conectado a uno o ambos componentes Account Server al mismo tiempo.

En UML, el representar un número variable de componentes no es fácil utilizando un diagrama de instancias. Por lo cual, es más cómodo utilizar la convención de nombres Client teller component c1, cX y cN para expresar un número arbitrario de clientes (1..N). El significado de esta convención tendría que ser documentado en la vista, como si no fuera una convención estándar en UML.

UML contiene muchos de los elementos para documentar correctamente e intuitivamente componentes de C&C pero sufre de suavidad visual. En las notaciones informales de C&C se pueden usar diferentes figuras para diferentes tipos de componentes hasta resaltar distinciones importantes, todos los tipos de componentes UML son representados gráficamente utilizando el mismo cuadro rectangular. En teoría, UML permite ese tipo de adaptación visual, pero muchas herramientas carecen de ella. Al igual que los tipos de conectores no pueden ser diferenciados rápidamente, por ejemplo, no existen convenciones para líneas diferentes; a cambio, el lector debe distinguir entre descripciones textuales en las líneas o las cajas, las cuales además introducen un desorden visual.

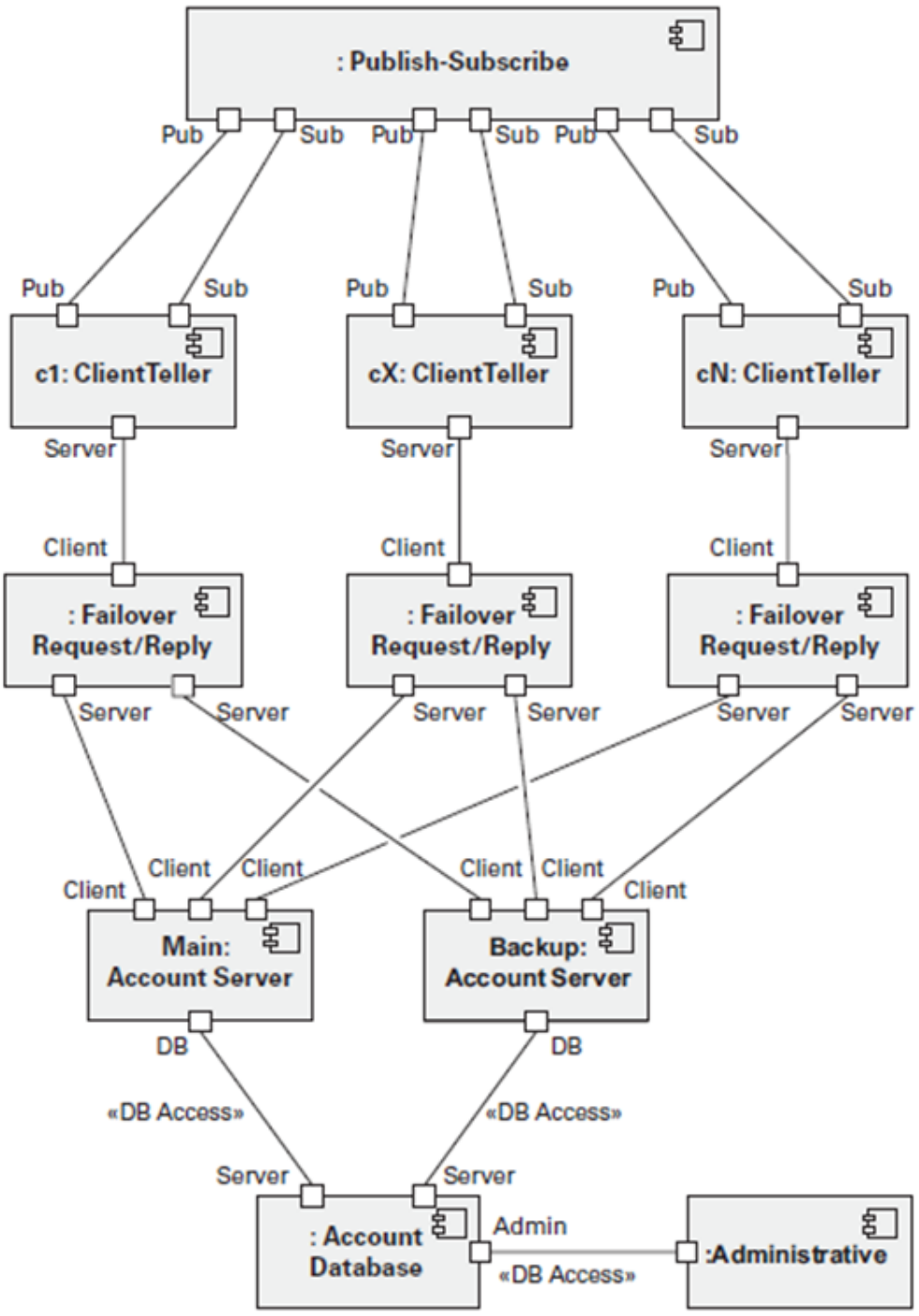


Figura A-51 Una representación la presentación principal encontrada en la Figura A-49

1.3. Tipo de Vista de Asignación

Cada estilo de asignación describe la asignación de unidades de software a elementos del ambiente (el hardware, el sistema de archivos, o el equipo de desarrollo).

La arquitectura de software interactúa con:

- Hardware: Para analizar el performance de los sistemas
- File System: Para administrar el desarrollo de sistemas
- Equipos de Trabajo: Para desarrollar las actividades de administración de proyectos

En el mapeo de la arquitectura de software en sus ambientes se pueden identificar tres estilos comunes (Patrones Arquitectónicos):

1. **Estilo de Distribución (Deployment):** que describe el mapeo de los componentes y conectores sobre el hardware en el cual se ejecuta
2. **Estilo de Aplicación (Implementation):** que describe el mapeo de los módulos en el file system que contiene esos módulos
3. **Estilo de Asignación de Trabajo:** que describe el mapeo de módulos en las personas, grupos, o equipos de tareas con el desarrollo de módulos

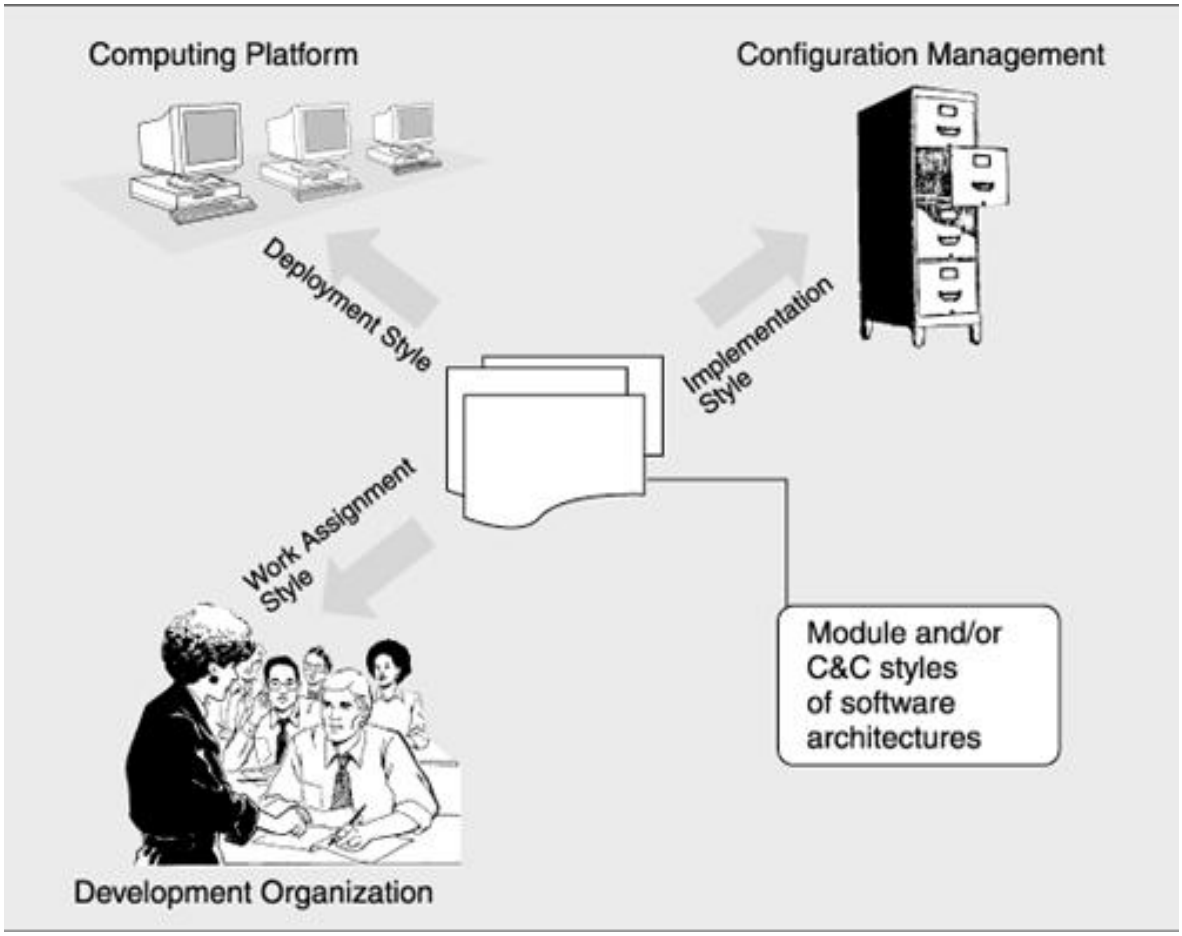


Figura A-52 Tipos de Vistas de Asignación

1.3.1. Elementos, Relaciones y Propiedades de los Tipos de Vistas

1.3.1.1. Elementos

Los Elementos del tipo de vista de asignación son elementos de software y elementos de ambiente. Un elemento de ambiente representa una pieza de una estructura del ambiente como puede ser un procesador, una granja de discos, un ítem de configuración, o un grupo de desarrollo. Los elementos de software en un estilo del tipo de vista de asignación provienen desde un estilo ya sea tipo de vista de módulo o componente y conector o ambos. Los elementos (no las relaciones) desde los estilos de módulos o componentes y conectores son enfatizados en un estilo de asignación.

1.3.1.2. Relaciones

La relación en un tipo de vista de asignación es la relación “asignado-a” (allocated-to), con la dirección desde el elemento de software al elemento del ambiente. Un elemento de software único puede ser asignado a múltiples elementos del ambiente, y múltiples elementos de software pueden ser asignados a un único elemento de ambiente. Esas asignaciones pueden cambiar en el tiempo (durante el desarrollo y ejecución del sistema), en tal caso se pueden aplicar las técnicas de especificación de arquitecturas dinámicas.

1.3.1.3. Propiedades

Los elementos de software y los elementos de ambiente tienen propiedades. Las propiedades específicas dependen del propósito de la asignación. Asignar un software a un elemento de ambiente implica coincidir las propiedades requeridas del elemento de software con las propiedades provistas del elemento de ambiente. Si esta coincidencia no puede realizarse, una relación “Asignado-a” no sería válida. Por ejemplo, para asegurar el tiempo de respuesta requerido de un componente, debe ejecutarse en un procesador suficientemente rápido. Esto podría ser una simple comparación: Una multiplicación de un IEEE 754 single-precision floating-point debe ejecutarse en 50 microsegundos. Es posible mirar las especificaciones de los tiempos de respuestas del procesador a utilizar para verificar que el requerimiento será cumplido. Estas comparaciones pueden ser más complicadas: La tarea no puede usar más de 10 kilobytes de memoria virtual. En este caso, debe realizarse un modelo de ejecución del elemento de software, y luego debe determinarse el uso de memoria virtual.

Los usos y notaciones específicos para los estilos del tipo de vista de asignación son estilos específicos y son cubiertos en sus respectivas secciones. A pesar de que los componentes de notación son diferentes para cada estilo, los estilos son dominados por los elementos de ambiente; los elementos de software toman un rol secundario.

1.3.2. Estilos del Tipo de Vista de Asignación

A continuación se describirán algunos de los estilos más comunes del tipo de vista de asignación. Cada estilo restringe el tipo de vista de asignación básico, quizás agregan versiones especializadas de tipos de elementos y relaciones.

- **El Estilo de Despliegue (deployment style)** asigna procesos a elementos de hardware: nodos de procesamiento, canales de comunicación, almacenes en memoria, y almacenes de datos. Los elementos de software en este estilo son generalmente procesos. Este estilo,

usado para describir como los procesos son asignados al hardware y el tráfico de mensajes resultantes, es usado para análisis de performance, seguridad y confiabilidad, y entrega una base para estimar el costo de distribución de un nodo único.

- **El Estilo de Aplicación (Implementation style)** asigna los módulos de un tipo de vista de módulo a la infraestructura de desarrollo. Los elementos del estilo de aplicación son módulos y entidades de configuración. Este estilo es usado para describir cómo los módulos son asignados a entidades dentro de un sistema de administración de configuración, así como para administrar versiones y revisiones, y para coordinar el desarrollo de múltiples equipos.
- **El Estilo de Asignación de Trabajo (Work assignment style)** asigna módulos de un tipo de vista de módulo a equipos de desarrollo humano. Los elementos del estilo de asignación de trabajo son módulos y equipos de desarrollo. Este estilo es usado para describir qué equipos son responsables de qué elementos del desglose de la estructura de trabajo, así como de informar la programación y el presupuesto estimado.

También existen otros estilos útiles en el tipo de vista de asignación que deben ser considerados. Como el estilo de implementación que describe cómo se organiza el ambiente de desarrollo en árboles de directorios y archivos, y cómo se relacionan con los módulos de una vista de módulos. También se puede encontrar el estilo de almacenes de datos que describe las relaciones entre entidades de datos y el hardware de los servidores de datos donde reside el sistema. Además, se pueden describir una variedad de estilos que describan relaciones con el ambiente del sistema y que apoyen necesidades específicas de los stakeholders.

1.3.2.1. Estilo de Despliegue (Deployment Style)

En el estilo despliegue, los elementos de un estilo de C&C –usualmente el estilo de comunicación de procesos– son asignados a plataformas de ejecución. Las restricciones para una asignación particular son los requerimientos expresados por los elementos de software y cómo esos requerimientos son cumplidos por las características de los elementos relevantes de hardware.

Elementos, Relaciones y Propiedades

La Tabla A-20 resume la discusión de las características del estilo de despliegue. Los elementos de ambiente en este estilo son entidades que corresponden a unidades físicas que almacenan, transmiten o procesan datos. Las unidades físicas incluyen nodos de procesamiento (CPUs), canales de comunicación, memoria volátiles y almacenes de datos.

Elementos	• Elementos de Software; usualmente un proceso desde el tipo de vista de C&C. • Elementos de Ambiente; hardware –procesador, memoria, disco, red y similares.
Relaciones	• **Asignado-a** (Allocated-to), muestra en qué unidades físicas reside el software • **Migra-a** (Migrates-to), una relación de un elemento de software en un procesador a el mismo elemento de software a un procesador diferente, esta relación indica que ese elemento de software puede moverse de un procesador a otro • **Copia-migra-a** (copy-migrates-to), esta relación es similar a la relación "migra-a" excepto que el elemento de software envía una copia de sí mismo al nuevo elemento de procesamiento mientras se mantiene la copia del elemento de procesamiento original • **Ejecución-migra-a** (execution-migrates-to), si la asignación es dinámica, similar a las dos anteriores, esta relación indica que la ejecución se mueve de procesador en procesador pero la residencia el código no cambia
Propiedades de los elementos	• Propiedades requeridas de un elemento de software; aspectos significantes del hardware, como procesamiento, memoria, requerimientos de capacidad, y tolerancia a fallas • Propiedades provistas por un elemento de ambiente: los aspectos relevantes de hardware que influencian las decisiones de asignación
Propiedades de relaciones	• Relación Asignado-a; ambas estática y dinámica
Topología	• Sin restricciones

Tabla A-20 Resumen del Estilo de Despliegue

Los elementos de software en este estilo son generalmente derivados de elementos en una vista de C&C relacionando los procesos. Cuando se representa en el estilo de despliegue, se asume que los elementos de software corren en un computador con un sistema operativo soportado. Aun cuando, los elementos de software en este estilo son parecidos a procesos de sistema operativo.
La relación generalmente representada en el estilo de despliegue es una forma especial de asignado-a que muestra en qué unidad física residen los elementos de software. La relación puede ser dinámica; esto significa que la asignación puede cambiar mientras el sistema se ejecuta. En este caso, pueden existir relaciones adicionales como las siguientes:

- **Asignado-a:** Una relación desde un elemento de software en un procesador al mismo elemento de software en un procesador diferente, esta relación indica que el elemento de software puede moverse de procesador a procesador pero no puede existir simultáneamente en ambos procesadores
- **Copia-migra-a:** Esta relación es similar a la relación migra-a, excepto que el elemento de software envía una copia de sí mismo al nuevo elemento de proceso mientras permanece la copia en el elemento de procesamiento original
- **Ejecución-migra-a:** similar a las dos anteriores, esta relación indica que la ejecución se mueve de procesador en procesador pero la residencia el código no cambia. Una copia de

un proceso existe en más de un procesador, pero sólo una está activa en un momento en particular. La ejecución del proceso "migra" cuando se cambia el proceso activo.

También es posible que la asignación cambie a través del tiempo como resultado de una reconfiguración manual. En este caso, las posibilidades representan variación de puntos.
Las propiedades importantes de los elementos, de software y físicos, del estilo de distribución son aquellas que afectan la asignación del software a los elementos físicos. Como un elemento físico satisface requerimientos de un elemento de software es determinado por las propiedades de ambos. Por ejemplo, si un elemento de software FOO requiere una capacidad mínima de almacenamiento, cualquier elemento de ambiente que tiene al menos esa capacidad es candidatos para la asignación exitosa de FOO.
Los tipos de análisis realizados basados en un estilo de distribución además determinan las propiedades particulares de los elementos que deben procesar. Por ejemplo, si es necesario un análisis de capacidad de memoria, las propiedades necesarias de los elementos de software deben describir aspectos de consumo de memoria, y las propiedades relevantes de los elementos de ambiente deben representar capacidades de memoria de varias entidades de hardware.
A continuación hay algunas propiedades de elementos de ambiente relevantes a unidades físicas.

- **Propiedades de CPU:** Se debe especificar un conjunto de propiedades relevantes a varios elementos de procesamiento. Estas propiedades incluyen velocidad del reloj de procesamiento, número de procesadores, capacidad de memoria, velocidad del bus, y velocidad de ejecución de instrucciones.
- **Propiedades de Memoria:** Se debe especificar un conjunto de propiedades relevantes al almacenamiento en memoria. Estas propiedades incluyen tamaño de memoria y características de velocidad.
- **Capacidad de Disco u otra unidad de almacenamiento:** Esta propiedad especifica la capacidad de almacenamiento y velocidad de acceso a unidades de disco: unidades de discos individuales, almacenes de discos, y unidades RAID (de sus iníciales en inglés, Redundant Array of Inexperience Disks).
- **Ancho de Banda:** Esta propiedad indica la capacidad de los canales de comunicación para transferir datos
- **Tolerancia a Fallas:** Múltiples unidades de hardware pueden realizar la misma función, y esas unidades deben tener un mecanismo de control de falla.

Propiedades que son relevantes a elementos de software son:

- **Consumo de Recursos:** Por ejemplo, el cálculo toma 32.123 instrucciones.
- **Requerimientos de recursos y restricciones que deben ser cumplidos:** Por ejemplo, un elemento de software debe ejecutarse en 0.1 segundos.
- **Safety Critical:** Por ejemplo, un elemento de software debe estar siempre ejecutándose

La siguiente propiedad es relevante para la asignación:

- **Migration Trigger:** si la asignación puede cambiar mientras se ejecuta el sistema, esta propiedad especifica qué debe ocurrir cuando se migra un elemento de software desde un elemento de procesamiento a otro.

¿Para qué es este estilo?

El estilo de despliegue es utilizado para análisis de **performance**, **confiabilidad**, y **seguridad**. Los testers utilizan esta vista para comprender las dependencias de tiempo de ejecución y los integradores lo utilizan para los planes de integración y pruebas. También se puede utilizar para apoyar en la estimación de costos en la evaluación de opciones de compra de hardware.

El performance se transforma cambiando la ubicación del software en el hardware. Las decisiones de asignaciones óptimas o mejores son aquellas que eliminan cuellos de botella en procesadores o que distribuyen el trabajo más uniformemente de forma que la utilización del procesador es aproximadamente uniforme a través del sistema. Generalmente, las mejoras de performance son enfrentadas colocando unidades de distribución que tienen comunicación frecuente y/o un alto ancho de banda con otras. El volumen y la frecuencia de comunicación entre diferentes unidades de distribución en diferentes elementos de procesamiento, los que toman un lugar en los canales de comunicación entre estos elementos, es el foco para la ingeniería de performance de sistemas. El arquitecto puede utilizar hardware adicional o reemplazar los elementos de hardware con versiones más potentes cuando las necesidades no se pueden satisfacer.

La disponibilidad y confiabilidad son afectadas directamente por el comportamiento del sistema cuando hay una degradación o falla en elementos de procesamiento o canales de comunicación. Si asumimos que un procesador o un canal de comunicación fallan sin una advertencia, las copias de unidades distribuibles son ubicadas en procesadores separados. Si se considera que una advertencia precederá a una falla, las unidades distribuibles pueden ser migradas en tiempo de ejecución cuando la falla es inminente. Los usuarios típicos de esta estructura son ingenieros de performance, ellos usan esta estructura para diseñar y para predecir el performance del sistema; los testers usan esta estructura para entender las dependencias en tiempo de ejecución, y los integradores utilizan esta vista para planear la integración y realizar las pruebas de integración.

El costo de implementación de un sistema depende de los elementos de hardware de ese sistema. Por lo tanto, la vista de distribución es utilizada para mostrar elementos de hardware de una configuración particular y sus propósitos. Los propósitos están dados por los procesos implementados en varios elementos de hardware.

Las arquitecturas de software modernas buscan hacer las decisiones de asignación transparente y, por lo tanto, fáciles de cambiar. Por ejemplo un objetivo es llevar a cabo la comunicación entre procesos exactamente de la misma manera que si los procesos residen en el mismo o en diferentes procesadores. Así el estilo de distribución puede contener información que los ejecutores no les deberían tener permitido utilizar. Cuando las decisiones de asignación son dinámicas se debe usar la relación migra-a en sus varias formas para indicar dinamismo.

Adicionalmente, las asignaciones pueden capturar implementaciones empresariales, por las cuales la asignación de subsistemas depende de una organización en la empresa. Cada organización puede usar sus propias políticas y estándares para controlar la implementación.

Un uso incorrecto del estilo de distribución es tratarlo como la arquitectura completa de un sistema. Una vista única de este estilo, aislada, no es una descripción completa de una arquitectura de software. A pesar de que esta observación se aplica a todos los estilos, los estilos de asignación son especialmente susceptibles. Cuando se pregunta por la arquitectura de un software, los arquitectos

a veces presentan un diagrama impresionante que muestra una red de ordenadores con todas sus propiedades y los protocolos utilizados, y el software que está corriendo en esos computadores. A pesar que estos diagramas juegan un rol importante, ayudando a organizar el trabajo y entender el software, no son representativos de la arquitectura de software.
No se debe forzar una relación entre los módulos y unidades de hardware. Por ejemplo, suele ser un error de diseño forzar a cada capa de un sistema de capas en su procesador propio ya que las capas no son niveles.

Notaciones para el Estilo de Despliegue

Notaciones Informales

Las notaciones informales gráficas contienen cajas, círculos, líneas, flechas y cosas así. Las Cajas y los Círculos son usados para representar el software y elementos de ambiente, y las líneas y flechas son usadas para representar la relación “asignado-a”. En muchos casos, se utilizan símbolos o iconos para representar los elementos de ambiente. Los símbolos son frecuentemente imágenes de los dispositivos de hardware en cuestión. Además, se utilizan sombras, colores, tipos de bordes y patrones de relleno para indicar el tipo de elemento. Los elementos de software se pueden mostrar en su interior o al lado del hardware en el que están ubicados. Si la estructura de distribución es simple, puede ser adecuada una tabla que liste las unidades de software y los elementos de hardware.

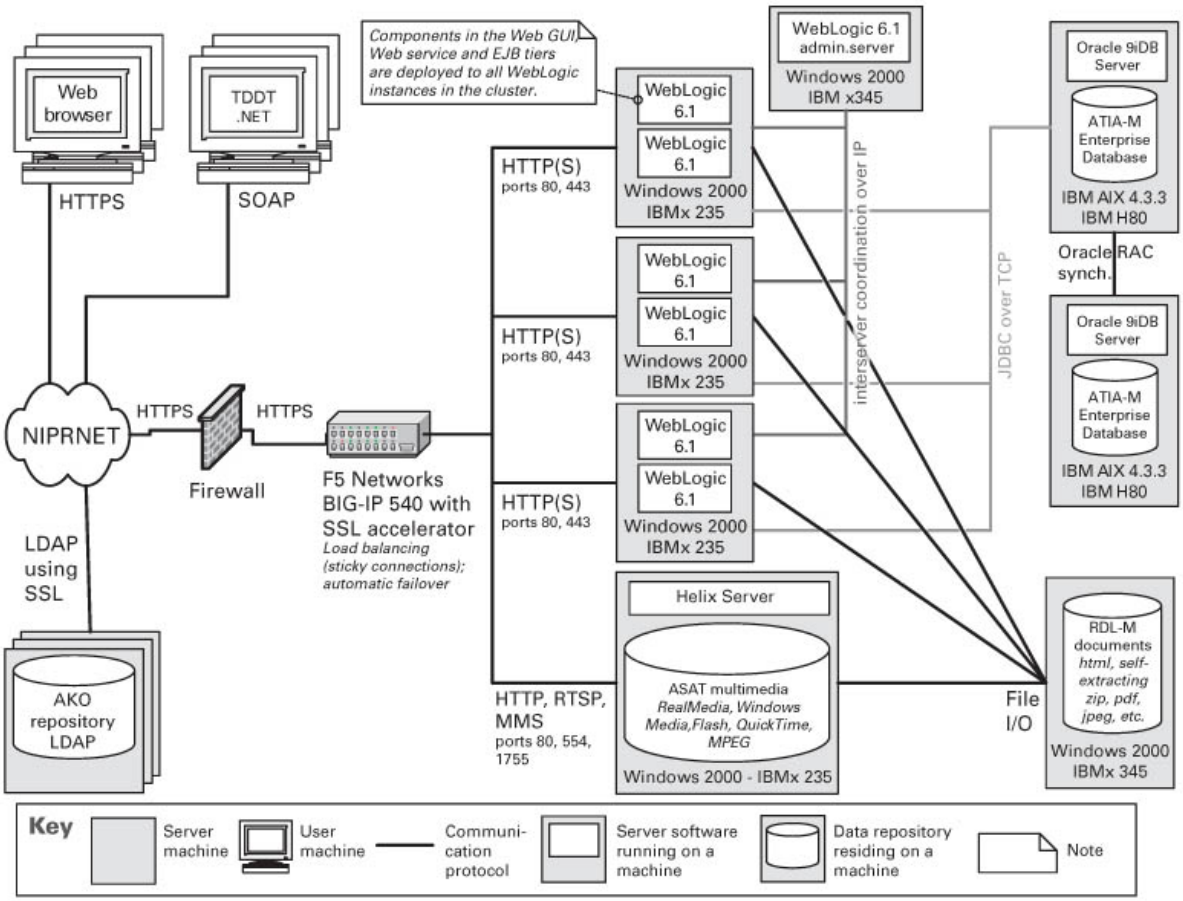


Figura A-53 Notación Informal para el Estilo de Despliegue

Notaciones Formales

El Architecture Analysis and Design Language (AADL) y SysML son ejemplos de lenguajes de descripción de la arquitectura que proporcionan notaciones formales para describir vistas de distribución. AADL proporciona un vocabulario para representar el hardware y enlazar el software a elementos de hardware como procesadores, memoria, y conexiones. El lenguaje permite el análisis de rendimiento, confiabilidad, seguridad-críticos, y los requisitos de seguridad. En SysML, la representación gráfica es compatible con una versión modificada de los diagramas de bloque de UML. Además, proporciona una forma de tabla para representar la implementación y otras formas de asignación.

UML

En UML, un diagrama de distribución es un gráfico de nodos conectados por asociaciones de comunicación. Los nodos corresponden a elementos de procesamiento, por lo general tienen una memoria y una capacidad de procesamiento. Los nodos pueden contener instancias de componentes, lo que indica que el componente se encuentra en el nodo. Los componentes pueden estar conectados entre sí por flechas de dependencia. En un diagrama de distribución de UML, los componentes pueden contener objetos, lo que significa que los objetos son parte de esos componentes. La migración de componentes de un nodo a otro (u objetos de componente a componente) está indicada con la dependencia "pertenece-a". Un nodo se muestra con un símbolo que parece una caja de tres dimensiones, con un nombre opcional en el interior. Los nodos están conectados por asociaciones que representan vías de comunicación. La naturaleza precisa de la trayectoria de comunicación puede ser indicada por un estereotipo de asociación (por ejemplo, <<Ethernet 10-T>>, <<RS-232>>). Las propiedades se representan como pares atributo nombre-valor (por ejemplo, velocidadProcesador = 300 MHz, memoria = 128 MB). Una especificación de distribución especifica los parámetros que guían la distribución de un componente, como el modo de concurrencia (por ejemplo: thread, proceso, ninguno).

La Figura A-54 muestra un ejemplo de la notación UML para la vista de despliegue.

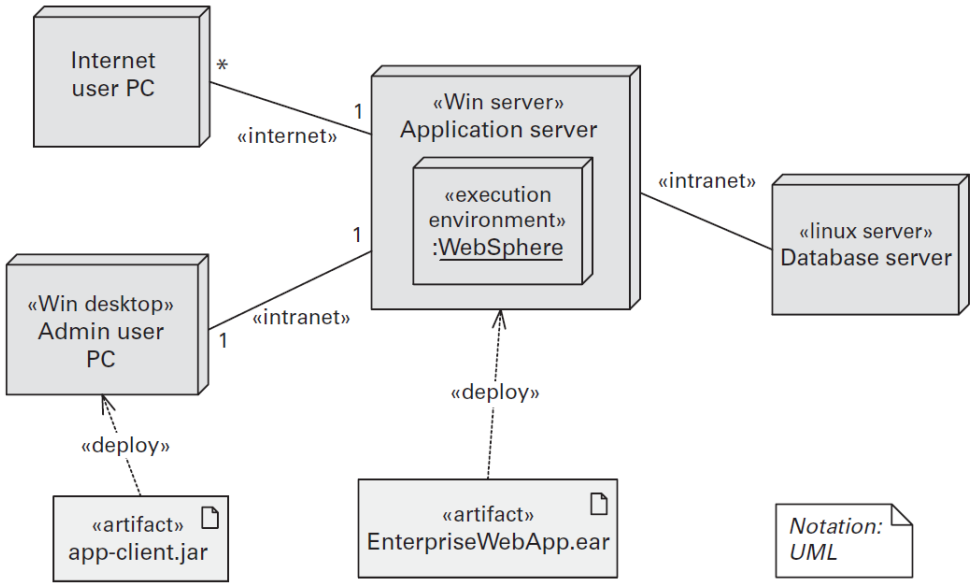


Figura A-54 Ejemplo de Notación UML para una Vista de Despliegue

Relación con otros estilos

El estilo de distribución está relacionado con el estilo de Componentes y Conectores (C&C) que proporciona elementos de software que son asignados al hardware de la plataforma informática. Asimismo, está estrechamente relacionado con el estilo de instalación, que muestra el contenido de los archivos implementados en los nodos de hardware.

1.3.2.2. Estilo de Aplicación (Implementation Style)

El estilo de aplicación mapea módulos de un tipo de vista de módulo a infraestructura de desarrollo. Implementar un módulo generalmente resulta en muchos archivos separados, como aquellos que contienen código fuente, archivos que deben ser incluidos y usualmente contienen definiciones, archivos que describen como construir un ejecutable, y archivos que son el resultado de traducciones o compilaciones del módulo. Esos archivos necesitan ser organizados para no perder el control y la integridad del sistema. Las técnicas de administración de la configuración usualmente realizan este trabajo: en el caso más simple, en una jerarquía de directorios en un sistema de archivos.

Una vista del estilo de aplicación puede ser mucho más elaborada, quizás mostrando ítems de configuración especiales para varios tipos de módulos, como los ítems que necesita un procedimiento específico de pruebas. Esto generalmente requiere sistemas de administración de la configuración más sofisticados que permitan la definición de nuevos ítems de configuración y procesos complejos necesarios para crear nuevos ítems en la estructura.

Elementos, Relaciones y Propiedades

La Tabla A-21 resume la discusión de las características de un estilo de aplicación. Los Elementos de ambiente en el estilo son ítems de configuración: archivos en un sistema de archivos o reglas administradas por el sistema de administración de la configuración. En el caso más simple, esos ítems de configuración son directorios para propósitos organizacionales y archivos como contenedores de información. Los elementos de software son módulos de cualquier tipo de vista del estilo de módulo, como funciones o clases.

Las relaciones del estilo de aplicación son:

- **Asignado-a (Allocated-to):** Una relación entre módulos e ítems de configuración. Esta relación conecta un módulo con el ítem de configuración que implementa dicho módulo y es principalmente una relación uno-a-uno. Aun cuando, el ítem de configuración puede componerse de múltiples ítems.
- **Contenedor (Containment):** Una relación entre ítems de configuración. Esta relación indica que un ítem de configuración contiene otros ítems de configuración. Un ítem de configuración puede estar contenido en múltiples otros ítems de configuración. Ejemplos de ellos son las estructuras de directorios y agrupaciones de versiones

La Tabla A-21 resume el estilo de Aplicación:

Elementos	• Elementos de Software; un módulo • Elementos de Ambiente; un ítem de configuración, como un archivo o directorio
Relaciones	• **Contiene (Containment),** especificando que un ítem de configuración es contenido en otro

	• **Asignado-a (Allocated-to),** describiendo la asignación de un módulo a un ítem de configuración
Propiedades de los elementos	• Propiedades requeridas de un elemento de software, si existe alguna; usualmente, requerimientos en los ambientes de desarrollo, como java o una base de datos • Propiedades provistas de un elemento de ambiente; indicadores de las características entregadas por los ambientes de desarrollo
Propiedades de relaciones	• Ninguna
Topología	• Ítems de Configuración Jerárquicos; está contenido en (is-contained-in)

Tabla A-21 Resumen del Estilo de Aplicación

Así como en el estilo de implementación, las propiedades importantes del software y los elementos de ambiente del estilo de aplicación son aquellos que afectan la asignación del software a los ítems de configuración. Por ejemplo, como un sistema de administración de la configuración se comporta con la historia y agrupaciones es una propiedad del ítem de configuración; una versión específica del compilador de java a utilizar podría ser una propiedad del módulo de software. Una dependencia que un módulo requiere antes de ser compilado otro, es una propiedad del módulo de software.

¿Para qué es este estilo?

El estilo de aplicación es usado durante el desarrollo y al momento de la construcción para manejar y mantener los archivos que corresponden a elementos de software. Los desarrolladores usan este estilo para identificar archivos que se pueden validar para actualizaciones, pruebas, o construcción del sistema; y para volver atrás en nuevas versiones.

El estilo de aplicación puede ser usado para especificar las diferencias de versión de un sistema en particular. Este estilo puede ser usado para identificar aquellos elementos que son usados para propósitos especiales, como en las pruebas o para analizar el sistema de administración de configuración.

Notaciones para el estilo de aplicación

Cualquier notación para el estilo de aplicación debe tener módulos, ítems de configuración, y el mapeo entre ellos. Para distinguir los ítems de configuración de los módulos se usan idealmente iconos. Además, se debe mostrar la descomposición de los ítems de configuración.

La Figura A-55 muestra una pequeña vista de aplicación, representada en una notación informal especial para el estilo.

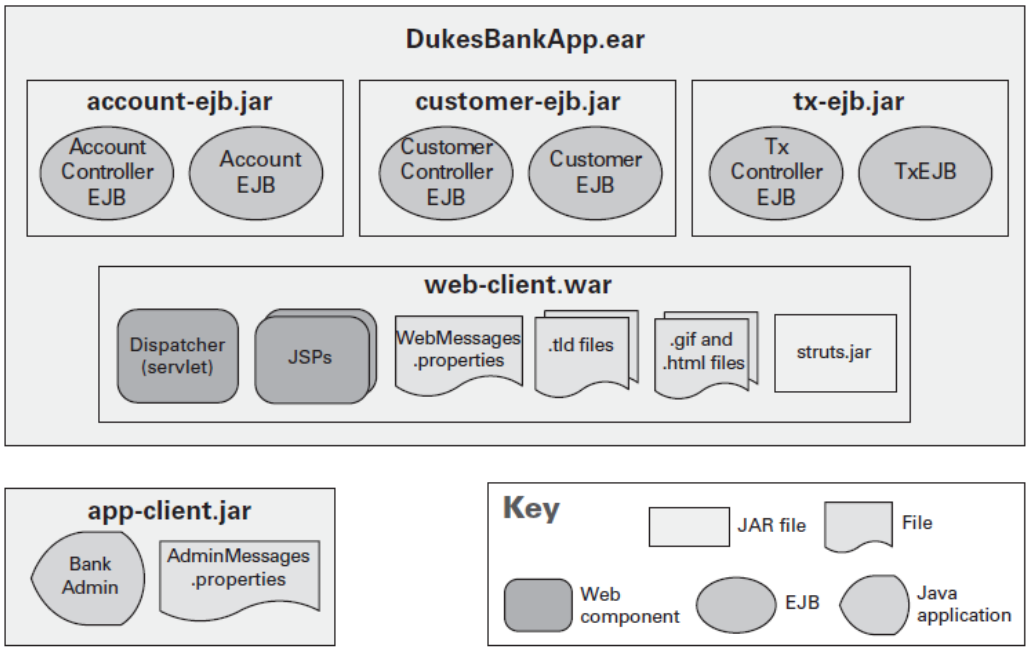


Figura A-55 Ejemplo de Notación Informal para una Vista de Aplicación

La Figura A-56 muestra el mismo diagrama anterior en notación UML.

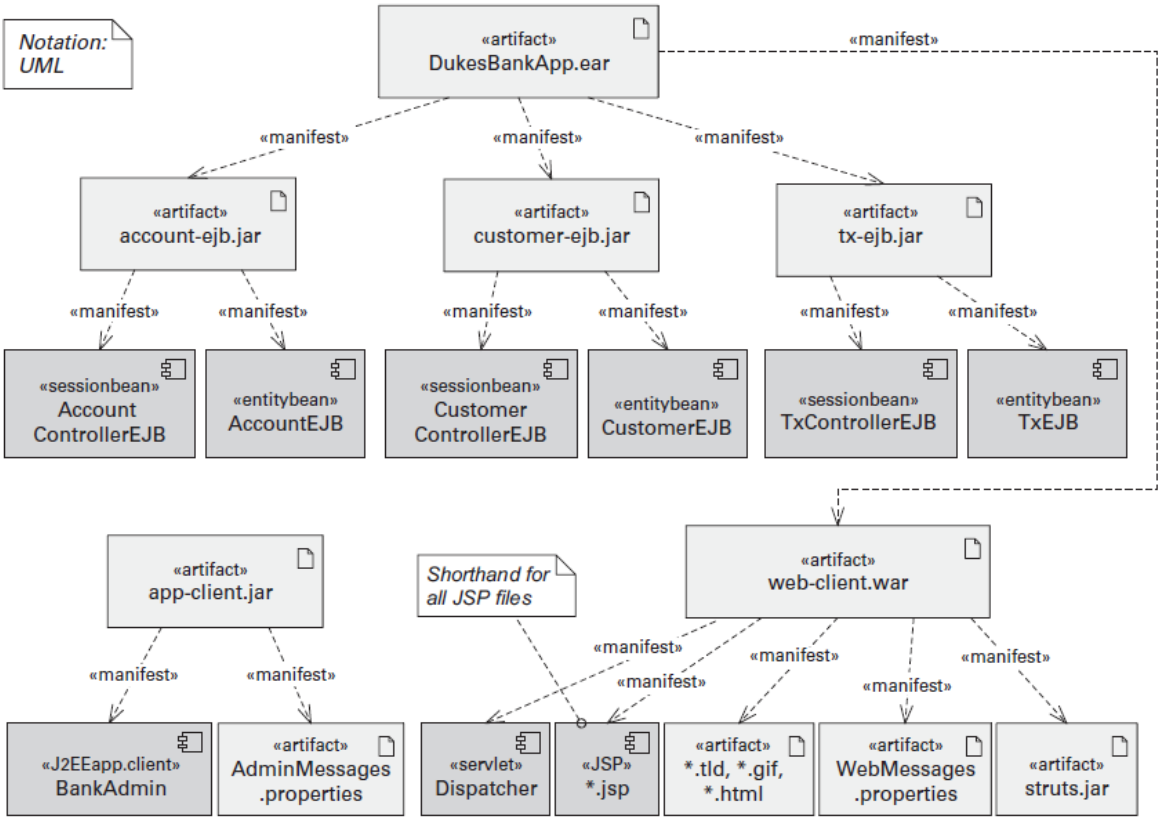


Figura A-56 Ejemplo de Notación Formal para una Vista de Aplicación

Relación con otros estilos

El estilo de aplicación es naturalmente el más relacionado a los estilos de módulos que muestran elementos de software para su ubicación.

1.3.2.3. Estilo de Asignación de Trabajo (Work Assignment Style)

En los años 60s, Conway [1968] formuló una ley que indica que las estructuras arquitectónicas se asemejan a las estructuras organizacionales. El basó su ley en casos de comunicación entre a través de grupos opuestos. Esta ley es una articulación organizacional de acoplamiento y coherencia. Los trabajos de software a través de estructuras generalmente se han basado en la descomposición de sistemas que son construidos por partes: una arquitectura de software. Recientemente, Paulish [2001] ha observado que el tiempo necesario y la estimación de presupuesto dependen de su base arquitectónica. Las observaciones de Paulish tienen principalmente una base intuitiva, como la estructura, la cual depende de la arquitectura de software.

Debido a que las asignaciones de trabajo representan una relación de la arquitectura de software en los grupos humanos, es un estilo importante. Los equipos –y sus asignaciones de trabajo- no están asociados simplemente con la construcción del software que correrá en el sistema final. Aun cuando si el software es adquirido enteramente como un producto comercial sin la necesidad de ningún trabajo de implementación, alguien debe hacerse responsable de impulsarlo, testearlo, y entender cómo trabaja, y alguien tiene que hablar por esto durante los test de integración y de sistema. El equipo responsable de eso tiene un lugar en la vista de asignación de trabajo. Además, el software escrito para soportar el desarrollo del sistema –herramientas, ambientes, casos de pruebas, y similares- y el equipo responsable tiene un lugar de primera clase en el estilo de asignación de trabajo.

Elementos, Relaciones, y Propiedades

La Tabla A-22 resume la discusión de las características del estilo de asignación de trabajo. Los elementos de este estilo son módulos de software y elementos de personas.

Elementos	• Elementos de Software; un módulo • Elementos de Ambiente; una unidad organizacional, como una persona, un equipo, un departamento, un sub-contratista, y cosas así.
Relaciones	• Asignado-a (Allocated-to)
Propiedades de los elementos	• Conjunto de Habilidades: requeridos y provistos
Propiedades de relaciones	• Ninguna
Topología	• En general, no restringida; en la práctica, usualmente restringida debido a que un módulo es asignado a una unidad organizacional

Tabla A-22 Resumen del Estilo de Asignación de Trabajo

La producción, adquisición, testing, y/o integración de módulos de software son responsabilidad de un individuo o un equipo de personas. La documentación del estilo de asignación de trabajo debe incluir información de cada módulo y responsabilidades; en esencia, para dar a cada equipo su capítulo

Los elementos de personas denotan unidades organizacionales o equipos específicos de personas o roles del personal. Las personas dentro de un elemento de personas pueden pertenecer a múltiples equipos para múltiples propósitos. Por ejemplo, una práctica común es para una persona que tiene una relación superior para asuntos de personal, como son las revisiones, y otras relaciones que reportan el desarrollo. Dentro del estilo de asignación de trabajo, consideraremos que su alcance es sólo lo relacionado con aquellos equipos que tienen responsabilidades de desarrollo.
En este estilo, la relación “asignado-a” mapea desde los elementos de software a los elementos de personas.
Una relación de asignación de trabajo bien formada tiene la propiedad de completitud –para ello se considera todo el trabajo- y no de solapamiento –ningún trabajo es asignado a dos lugares. Las propiedades de los elementos pueden incluir una descripción del conjunto de habilidades requeridas, así como las propiedades de los elementos de personas pueden incluir conjuntos de habilidades previstos.

¿Para qué es este estilo?

El estilo de asignación de trabajo muestra las principales unidades de software que deben ser presentadas para formar un sistema que funcione y quien las producirá, así como las herramientas y los ambientes en los cuales el software será construido. Este estilo está bien acotado para administrar asignaciones de equipos de recursos y para explicar la estructura de un proyecto –para una nueva línea, por ejemplo. Este estilo es lo básico para estructuras de trabajo y para detallar el presupuesto y estimaciones de tiempo.
El estilo de asignación de trabajo no muestra las relaciones de tiempo de ejecución, como llamadas o transmisión de datos. Ni tampoco muestra relaciones de dependencia entre módulos.

Notaciones para el estilo de Asignación de Trabajo

No existen notaciones especiales para asignaciones de trabajo arquitecturales.

Relaciones con otros estilos

El estilo de asignación de trabajo está estrechamente relacionado y utiliza el estilo de descomposición de módulos como la base para su planificación de asignación. Este estilo puede extender la descomposición de módulos que corresponden a las herramientas de desarrollo, herramientas de testing, sistemas de administración de la configuración, y relacionados con los anteriores, aquellas operaciones rutinarias y del día a día deben ser asignadas a un individuo y un equipo.
El estilo de asignación de trabajo es generalmente combinado con otros estilos. Por ejemplo, las asignaciones de equipos de trabajo pueden ser módulos en un estilo de descomposición de módulos, las capas en un diagrama de capas, el software asociado con niveles en una arquitectura multicapa, o las tareas o procesos en un sistema multiproceso. Además, este estilo es una confluencia de diferentes vistas conceptuales, a veces trabaja lo suficientemente bien. La creación de un estilo de asignación de trabajo como un estilo separado –donde se mantiene separado o cuidadosamente incorporado en otro- posibilita al arquitecto y al jefe de proyecto poner atención a la mejor manera de dividir el trabajo en partes manejables y mantener explícitamente la necesidad de asignar la responsabilidad del software, como un ambiente de desarrollo, que no será parte del ambiente de implementación. El peligro de combinar asignaciones de trabajo en otros estilos es que las asignaciones de trabajo asociadas con herramientas de construcción pueden perderse.

Se debe tener cuidado al combinar el estilo de asignación de trabajo con otros estilos. Recuerde que la descomposición de las asignaciones de trabajo pertenece al mismo tipo de elementos. Lo mismo no es verdad si se aplica a procesos, niveles, capas, y muchos otros elementos arquitectónicos. Si está basado en la descomposición, la estructura de asignación de trabajo no mapeará bien aquellos tipos de elementos. Por otra parte, mapear un módulo la descomposición obtenida bajo el principio de ocultamiento o encapsulamiento de información es un tema natural. El compartir información dentro de módulos es gratamente deseado para compartir información entre equipos.

Ejemplo de un Estilo de Asignación de Trabajo

La Figura A-57 muestra una porción de la vista de asignación de trabajo para el sistema ECS.

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
. . .	. . .	. . .

Figura A-57 Ejemplo de una Vista de Asignación de Trabajo

1.3.3. ¿Para qué es el Tipo de Vista de Asignación?

Las vistas de asignación se usan para describir las relaciones con el ambiente externo de un sistema, donde se pueden encontrar elementos como el hardware que utiliza, los equipos de trabajo que participaron en la implementación, ingenieros de performance, unidades organizacionales que son responsables de la administración, etc. Cada elemento tiene propiedades que son útiles para los intereses de los stakeholders que pueden ser tan variados como para responder preguntas como:

- ¿cómo el sistema se mantiene operativo ante un desastre natural?
- ¿cuanto se demorará un proceso para poder iniciar las operaciones diarias?

- ¿el sistema tiene mecanismos de seguridad que impidan accesos no autorizados?
- ¿Cuáles deben ser los conocimientos técnicos que debe tener el personal para integrarlo a otros sistemas?

Estas preguntas tienen relación con los atributos de calidad de funcionalidad, confiabilidad, eficiencia y mantenibilidad del sistema.

ANEXO B. MODELO DE VISTAS 4+1

El Modelo de Vistas 4+1 [2] está orientado a describir sistemas orientados a objetos [7] y es representado gráficamente como muestra la Figura B-1.

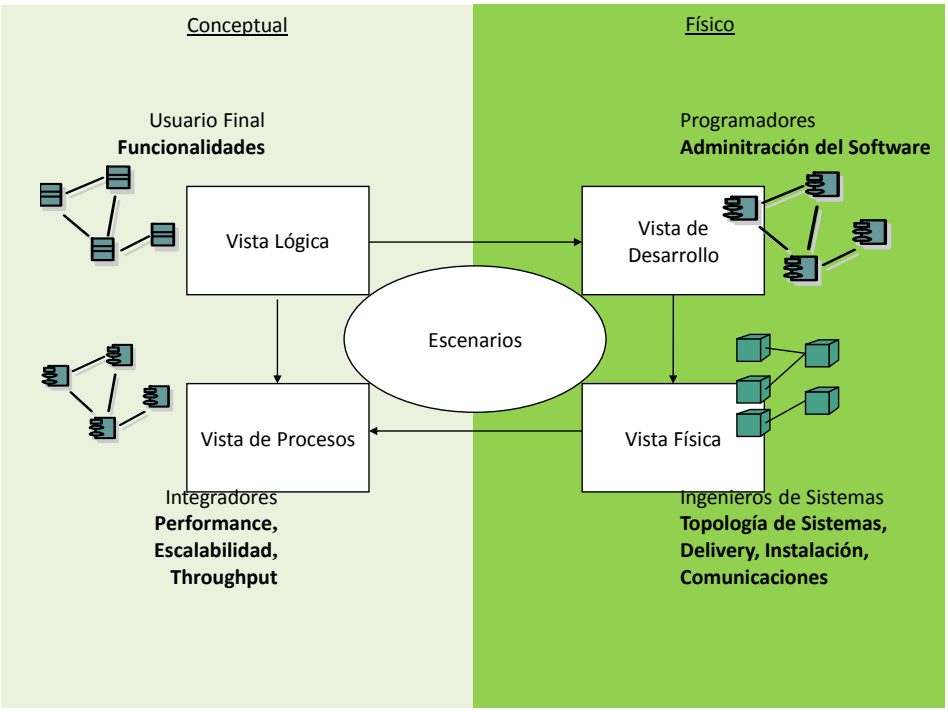


Figura B-1 El modelo de vistas 4+1 [2]

Este modelo fue propuesto por Philippe Kruchten en 1995 [2] y se divide en:

- **Vista Lógica**: describe el modelo de objetos del diseño, cuando se usa un método de diseño orientado a objetos.
- **Vista de Procesos**: describe los aspectos de concurrencia y sincronización del diseño.
- **Vista de Física**: describe cómo está organizado el software sobre el hardware y refleja los aspectos de distribución.
- **Vista de Desarrollo**: describe la organización estática del software en sus ambientes de desarrollo.

Con cuatro vistas se muestran las decisiones de arquitectura, y se suma una quinta vista llamada **Escenarios** o **Casos de Usos,** que indica las necesidades funcionales del sistema descritas por las otras vistas en un conjunto reducido de casos de usos. La descripción gráfica de este modelo lo muestra la Figura B-1 donde la Vista Lógica y la Vista de Procesos muestra la parte conceptual de la arquitectura, y la Vista de Desarrollo y la Vista Física se encargan de mostrar la parte física.
Este modelo tiene una relación directa con UML como se ve en la Figura B-2.

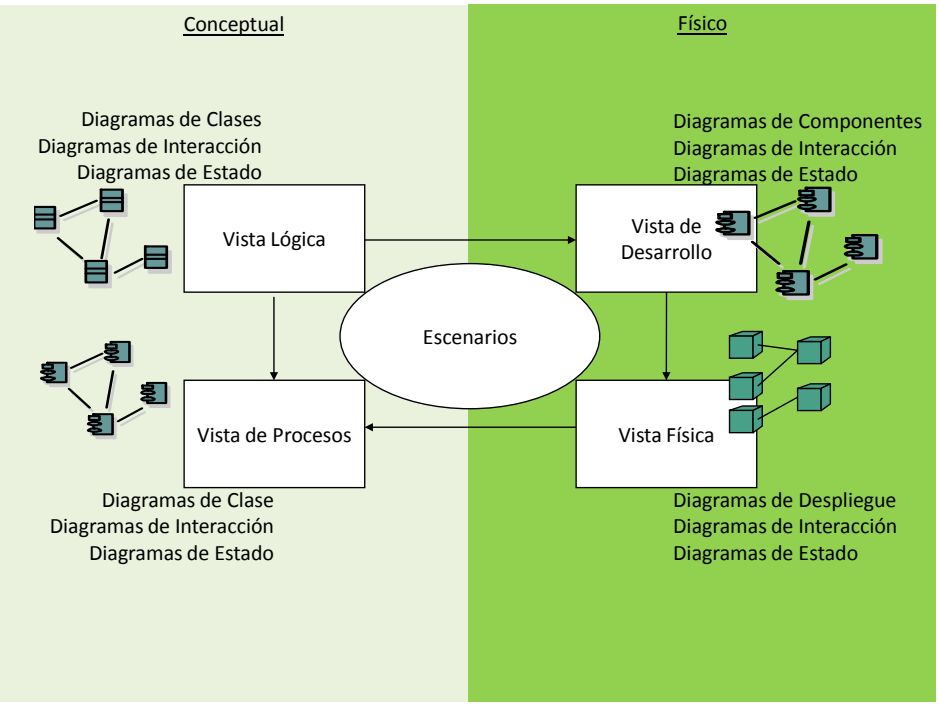


Figura B-2 El Modelo de Vistas 4+1 y su relación con UML

A continuación se revisará en más detalle cada una de las vistas del Modelo de Vistas 4+1.

2.1. Vista Lógica

La Vista Lógica está orientada a mostrar los requerimientos funcionales del sistema al Usuario Final. El sistema es descompuesto en un conjunto de abstracciones, obtenidas principalmente del dominio del problema, en forma de objetos o clases de objetos. Esta descomposición no está orientada sólo al análisis funcional, sino que sirve para identificar mecanismos comunes y elementos de diseño a través de varias partes del sistema.

2.1.1. Notación

Para representar esta vista, generalmente se utilizan Diagramas de Clases que representan mecanismos comunes y elementos de diseño de varias partes del sistema, y, si es necesario, se utilizan Diagramas de Transición de Estados para mostrar el estado interno de un elemento.
La Vista Lógica deriva de la notación de Booch [2] simplificándola considerablemente para mostrar los elementos que son arquitectónicamente significativos. Para representarla, se pueden utilizar elementos de notación de Booch [17] mostrados en la Figura B-3.

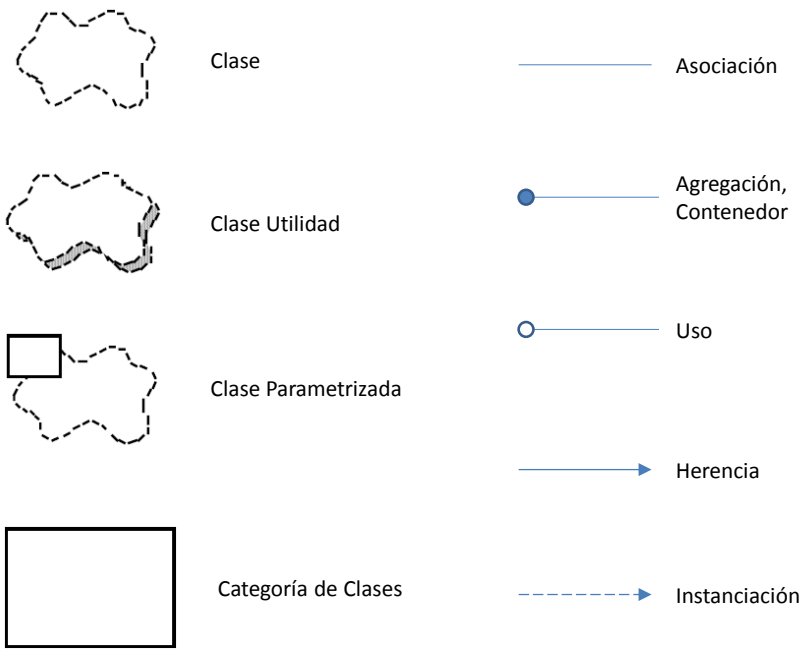


Figura B-3 Elemento de Notación para la Vista Lógica [2]

Un ejemplo de representación de una Vista Lógica es el que muestra la Figura B-4. Una PABX[3] establece la comunicación entre terminales. Un *Terminal* puede ser un teléfono, una línea troncal, una línea de interconexión, una línea de datos, una línea ISDN, etc. Cada tipo de línea es soportada por una tarjeta de interfaz diferente. La responsabilidad del objeto *Controlador* de línea es decodificar e inyectar todas las señales en la tarjeta de interfaz de línea, traduciendo las señales específicas de una tarjeta para un pequeño y uniforme conjunto de eventos: iniciar, parar, digitar, etc. El controlador también tiene todas las restricciones de tiempo real. Esta clase tiene muchas subclases para hacer frente a diferentes tipos de interfaces. La responsabilidad del objeto *Terminal* es mantener el estado del terminal y negociar los servicios a nombre de esa línea. Por ejemplo, usar los servicios de plan de enumeración para interpretar el marcado en la fase de selección. La *Conversación* representa un conjunto de terminales involucrados en una conversación. La conversación utiliza *Servicios de Traducción* (directorios, correspondencia entre direcciones físicas y lógicas, ruteos), y los *Servicios de Conexión* para establecer una ruta de acceso entre los terminales.

[3] Un PBX o PABX (siglas en inglés de Private Branch Exchange y Private Automatic Branch Exchange para PABX) cuya traducción al español sería Ramal privado de conmutación automática, o más bien Central Secundaria Privada Automática; es en realidad cualquier central telefónica conectada directamente a la red pública de telefonía por medio de líneas troncales para gestionar además de las llamadas internas, las entrantes y salientes con autonomía sobre cualquier otra central telefónica. Este dispositivo generalmente pertenece a la empresa que lo tiene instalado y no a la compañía telefónica, de aquí el adjetivo Privado a su denominación.

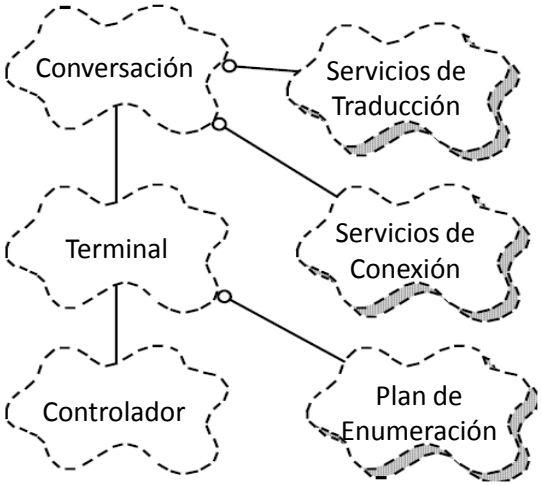


Figura B-4 Vista Lógica de PABX Telix [2].

Para sistemas mucho más grandes, que contienen algunas docenas de clases de significado arquitectónico, un ejemplo sería un diagrama de clases de alto nivel como el de la Figura B-5 que contiene 8 categorías de clases.

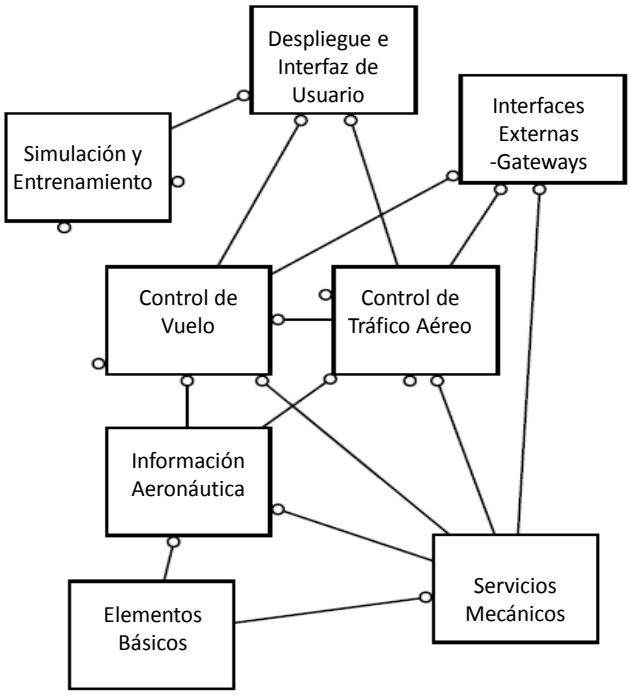


Figura B-5 Vista Lógica de un Sistema de Control de Tráfico Aéreo [2]

2.2. Vista de Procesos

La arquitectura de procesos toma en cuenta requerimientos no funcionales como el performance y la disponibilidad. Enfrenta problemas de concurrencia y distribución, de la integridad de sistemas, de la

tolerancia a fallos, y cómo las principales abstracciones de la vista lógica se ajustan dentro de la arquitectura de procesos –en qué hilo de control se está ejecutando una operación de un objeto.
La arquitectura de procesos puede ser descrita en diferentes niveles de abstracción, cada nivel enfrenta diferentes preocupaciones. En el nivel más alto, la arquitectura de procesos puede ser vista como un conjunto independiente de redes lógicas de programas que se comunican (llamados procesos), distribuidos a través de un conjunto de recursos de hardware conectados por una LAN o una WAN. Pueden existir múltiples redes lógicas simultáneamente, compartir los mismos recursos físicos. Por ejemplo, se pueden utilizar múltiples redes lógicas para apoyar la separación de la operación del sistema en línea del sistema fuera de línea, así como apoyar la coexistencia de versiones de software de simulación o prueba.
Un proceso es un agrupamiento de tareas que forman una unidad ejecutable. Los procesos representan el nivel en el cual la arquitectura de procesos puede ser controlada tácticamente (por ejemplo: iniciar, recuperar, reconfigurar, y detener). Además, los procesos pueden ser replicados para aumentar la distribución de la carga, o para mejorar la disponibilidad.
El software se divide en un conjunto de tareas independientes. Una tarea es un hilo de control separado, que puede ser planificado individualmente en un nodo de procesamiento. Entonces podemos distinguir:

- **Tareas principales:** que son elementos de arquitectura que pueden ser manejados en forma única. Estas se comunican a través de mecanismos de comunicación entre-tareas bien definido: servicios de comunicación basados en mensajes síncronos y asíncronos, llamadas a procedimientos remotos, difusión de eventos, etc. Las tareas principales no deben tener suposiciones relacionadas a su ubicación con otras tareas en un mismo proceso o en un nodo de procesamiento.
- **Tareas secundarias:** que son tareas adicionales que son introducidas localmente por razones de implementación (actividades cíclicas, almacenamiento en buffers, time-outs, etc.). Pueden comunicarse a través de memoria compartida.

El flujo de mensajes y la carga de procesos pueden estimarse en base al diagrama de procesos. También es posible implementar una vista de procesos vacía con cargas dummy para los procesos, y medir su performance en el sistema objetivo.

2.2.1. Notación

La notación propuesta para representar la vista de procesos puede ser la indicada en la Figura B-6.

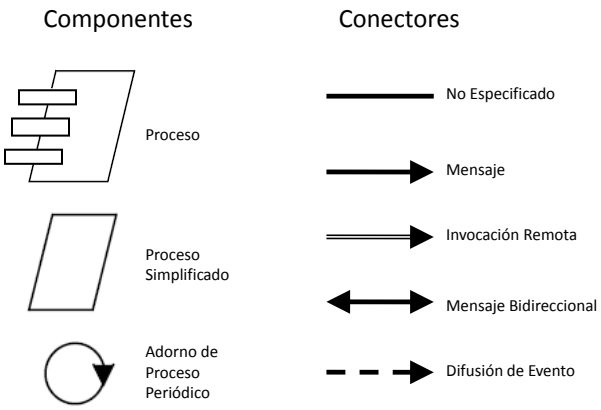


Figura B-6 Notación para Diagramas de Procesos

2.2.2. Estilos

Para representar la vista de procesos se pueden utilizar muchos estilos. Por ejemplo, tomando la taxonomía de Garlan, et al [8] tenemos: Tuberías y Filtros, o Cliente-Servidor, con variantes de varios clientes y un único servidor o múltiples clientes y múltiples servidores.

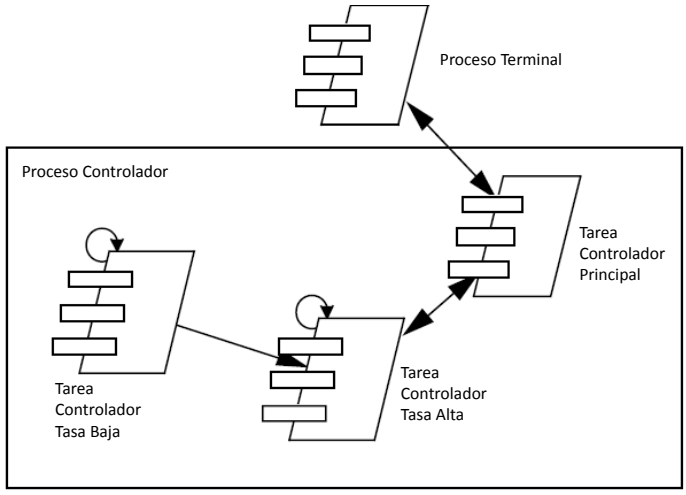


Figura B-7 Diagrama (Parcial) de procesos para Telic (PABX) [2]

La Figura B-7 muestra una vista parcial de procesos para el sistema PABX. Todos los terminales son administrados por un único proceso terminal, el que es controlado a través de mensajes en sus colas de entrada. Los objetos controladores son ejecutados en alguna de las tres tareas que componen el proceso controlador: una tarea cíclica de tasa baja que chequea todos los terminales (200ms), pone todo terminal que se activa en la lista de búsqueda de la tarea cíclica de tasa alta (10ms), la que detecta cualquier cambio de estado significativo, y lo pasa a la tarea controladora

principal que interpreta el cambio y lo comunica mediante un mensaje al terminal correspondiente. Aquí el paso del mensaje dentro del proceso controlador es hecho utilizando memoria compartida.

2.3. Vista Física

La Vista Física toma en cuenta principalmente los requisitos no funcionales como la disponibilidad, confiabilidad (tolerancia a fallas), performance (throughput), y escalabilidad. Considerando que el software se ejecuta sobre una red de computadores o nodos de procesamiento, la variedad de elementos identificados –redes, procesos, tareas y objetos- requieren ser mapeados sobre estos nodos. Se espera que puedan usarse diferentes configuraciones: algunas para desarrollo y pruebas, otras para emplazar el sistema en varios sitios para distintos usuarios. Por lo tanto, el mapeo del software en los nodos requiere ser muy flexible y tener un impacto mínimo sobre el código fuente.

2.3.1. Notación

Los diagramas físicos pueden tornarse muy confusos cuando se representan grandes sistemas, y por lo tanto tomas diversas formas, con o sin el mapeo de la vista de procesos.

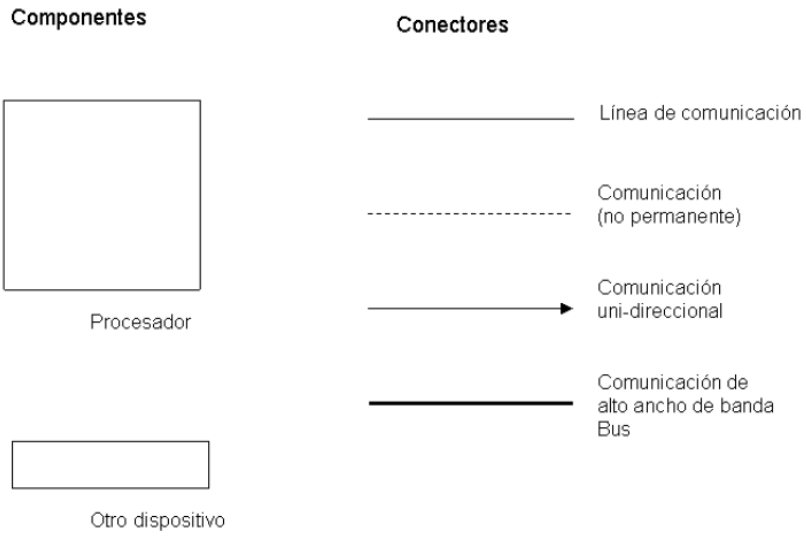


Figura B-8 Notación para el diagrama físico [2]

En base a la notación indicada en la Figura B-8 se representa un diagrama físico como el mostrado en la Figura B-9 que muestra la configuración de hardware posible para una gran PABX.

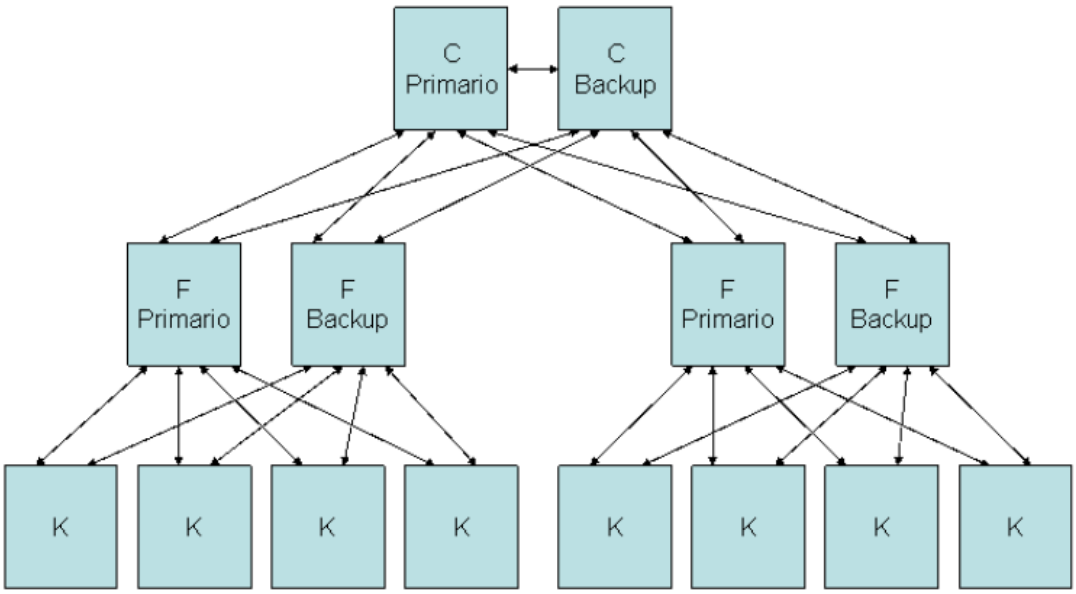


Figura B-9 Diagrama Físico de PABX [2]

2.4. Vista de Desarrollo

La Vista de Desarrollo que permite a los programadores la administración de los módulos. El software se empaqueta en partes pequeñas –bibliotecas de programas o subsistemas- que pueden ser desarrollados por uno o un grupo pequeño de desarrolladores. Los subsistemas se organizan en una jerarquía de capas, cada una de las cuales brinda una interfaz estrecha y bien definida hacia las capas superiores.

La vista de desarrollo tiene en cuenta los requisitos internos relativos a la facilidad de desarrollo, administración del software, reutilización y elementos comunes, y restricciones impuestas por las herramientas o el lenguaje de programación que se utilice. La vista de desarrollo apoya la asignación de requisitos y trabajo al equipo de desarrollo, y apoya la evaluación de costos, la planificación, el monitoreo del progreso del proyecto, y también sirve de base para analizar el reúso, portabilidad y seguridad. Es la base para establecer una línea de productos.

La vista de desarrollo de un sistema se representa a través de diagramas de módulos o subsistemas que muestran las relaciones exporta e importa. La arquitectura de desarrollo completa sólo puede describirse completamente cuando todos los elementos del software han sido identificados. Sin embargo es posible listar las reglas que rigen la arquitectura de desarrollo –partición, agrupamiento, visibilidad- antes de conocer todos los elementos.

2.4.1. Notación

Como se muestra en la Figura B-10, se utiliza una variante de la notación Booch [17] limitándose a aquellos items relevantes para la arquitectura.

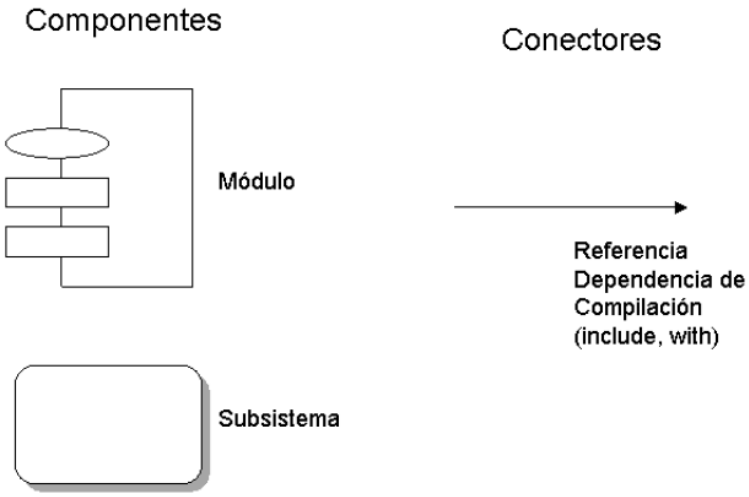


Figura B-10 Notación para el diagrama de desarrollo [2]

2.4.2. Estilos

El estilo recomendado para vistas de despliegue es el estilo de capas, definiendo entre 4 a 6 niveles de subsistemas. Cada capa tiene una responsabilidad bien definida. La regla de diseño es que un subsistema en una cierta capa sólo puede depender de subsistemas que estén en la misma capa o en capas inferiores, de este modo se minimiza el desarrollo de redes de dependencia complejas entre módulos y se permite el uso de estrategias de desarrollo por capas.

2.4.3. Ejemplo de Arquitectura de Desarrollo

La Figura B-11 representa la organización del desarrollo en 5 capas de la línea de productos de un sistema de tráfico aéreo desarrollado por Hughes Aircraft de Canadá. Esta es la arquitectura de desarrollo correspondiente a la arquitectura lógica que se muestra en la Figura B-5.

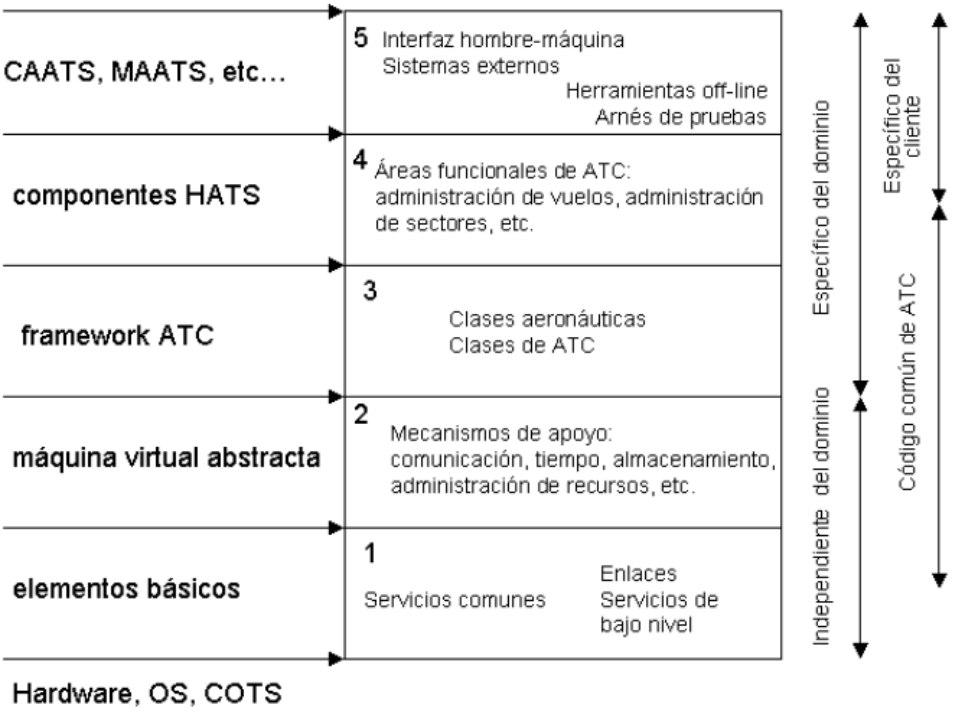


Figura B-11 Las 5 capas de un sistema de tráfico aéreo de Hughes (HATS) [2]

Las capas 1 y 2 constituyen la infraestructura distribuida independiente del dominio que es común a toda la línea de productos y la independiza de las variaciones de la plataforma de hardware, sistema operativo, o productos comerciales tales como administradores de bases de datos. La capa 3 agrega a esta infraestructura un framework ATC para formar una arquitectura de software dependiente del dominio. Usando este framework, en la capa 4 se construye una paleta de funcionalidad. La capa 5 es dependiente del cliente y del producto, y contiene la mayor parte de las interfaces con el usuario y con sistemas externos. Tantos como 72 subsistemas forman parte de la capa 5, cada uno de los cuales contienen entre 10 y 50 módulos, y puede representarse en diagramas adicionales.

2.5. Escenarios

Los elementos de las cuatro vistas trabajan conjuntamente en forma natural mediante la representación de un pequeño número de escenarios relevantes –instancias de casos de uso más generales- para los cuales se deben describir los scripts correspondientes (secuencias de interacciones entre objetos y procesos). Los escenarios son de alguna forma una abstracción de los requisitos más importantes. Su diseño se expresa mediante el uso de diagrama de escenarios y diagramas de interacción de objetos.
Esta vista es redundante con las otras (y por lo tanto “+1”), pero sirve a dos propósitos principales:

- Como una guía para descubrir elementos arquitectónicos durante el diseño de arquitectura.
- Como un rol de validación e ilustración después de completar el diseño de arquitectura, en el papel y como punto de partida de las pruebas de un prototipo de la arquitectura.

2.5.1. Notación

La notación es muy similar a la vista lógica para los componentes, pero usa los conectores de la vista de procesos para la interacción entre objetos. Las instancias de objeto se denotan con líneas sólidas.
Un ejemplo de escenario es el de la Figura B-12, donde se muestra el inicio para una llamada local para una PABX.

1. El Controlador del teléfono de Joe detecta y valida la transición desde colgado a descolgado y envía el mensaje correspondiente para levantar el objeto terminal correspondiente.
2. El Terminal asigna algunos recursos, e indica al controlador que emita algún tono de discado.
3. El Controlador recibe los dígitos y los transmite al Terminal.
4. El Terminal usa el Plan de Numeración para analizar el flujo de dígitos.
5. Cuando se ingresa una secuencia de dígitos valida, el Terminal abre una conversación.

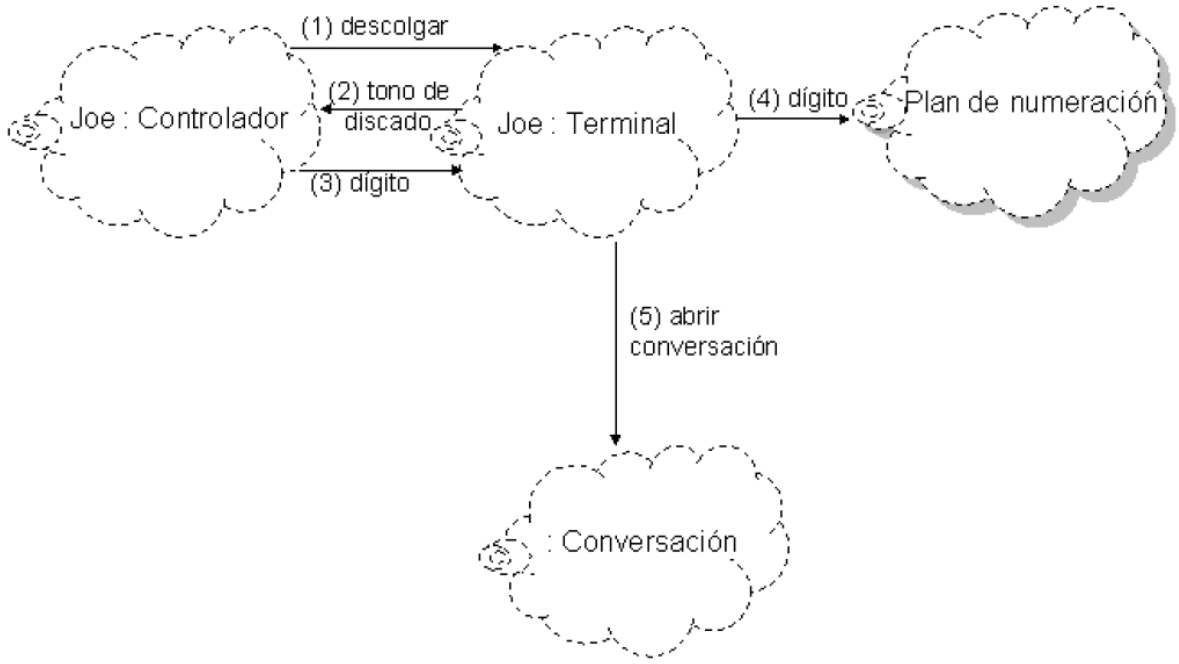


Figura B-12 Escenario de inicio para una llamada local – Fase de Selección [2]

2.6. Correspondencia entre las Vistas

Las distintas vistas no son completamente ortogonales o independientes. Los elementos de una vista están conectados a los elementos de las otras vistas siguiendo ciertas reglas y heurísticas de diseño.

2.6.1. Desde la Vista Lógica a la Vista de Procesos.

Se identifican varias características importantes de las clases de la arquitectura lógica:

- Autonomía: ¿Los objetos son activos, pasivos o protegidos?
    - un objeto activo toma la iniciativa de invocar las operaciones de otros objetos o sus propias operaciones y tiene el control completo sobre la invocación de sus operaciones por parte de otros objetos.
    - un objeto pasivo nunca invoca espontáneamente ninguna operación y no tiene ningún control sobre la invocación de sus operaciones por parte de otros objetos.
    - un objeto protegido nunca invoca espontáneamente ninguna operación pero ejecuta cierto arbitraje sobre la invocación de sus operaciones.
- Persistencia: ¿Los objetos son permanentes o temporales? ¿Qué hacen ante la falla de un proceso o un procesador?
    - Subordinación: ¿La existencia o persistencia de un objeto depende de otro objeto?
    - Distribución: ¿Están el estado y las operaciones de un objeto accesibles desde varios nodos de la arquitectura física, y desde varios procesos de la arquitectura de procesos?

En la vista lógica de la arquitectura se considera que cada objeto está activo y es potencialmente "concurrente". Por ejemplo, estando en paralelo con otros objetos, no se pone atención al grado exacto de concurrencia que se requiere para alcanzar este efecto. Por lo tanto, la arquitectura lógica tiene en cuenta sólo el aspecto funcional de los requisitos.
Sin embargo, cuanto se define la arquitectura de procesos, implementar cada objeto con su propio thread de control (por ejemplo, su propio proceso Unix o tarea Ada) no es muy práctico en el estado actual de la tecnología debido al gran overhead que esto impone. Más aún, si los objetos son concurrentes, debería haber alguna forma de arbitraje para invocar sus operaciones.
Por otra parte, ser requiere múltiples threads de control por varias razones:

- para reaccionar rápidamente a ciertas clases de estímulos externos, incluyendo eventos relativos al tiempo
- para sacar partido de las múltiples CPUs en un nodo, o los múltiples nodos en un sistema operativo
- para aumentar la utilización de la CPU, asignando la CPU a otras actividades mientras algún thread de control está suspendido esperando que otra actividad finalice (por ejemplo, acceso a cierto dispositivo externo, o acceso a otro objeto activo)
- para priorizar actividades (y potencialmente mejorar la respuesta)
- para apoyar la escalabilidad del sistema (con procesos adicionales que compartan la carga)
- para separar intereses entre las diferentes áreas del software
- para alcanzar una mayor disponibilidad del sistema (con procesos de backup)

Se utilizan dos estrategias concurrentemente para determinar la cantidad correcta de concurrencia y definir el conjunto de procesos que se necesitan. Considerando el conjunto de posibles arquitecturas físicas, se puede proceder o bien:

- **De adentro hacia afuera:** A partir de la arquitectura lógica iterar los siguientes pasos hasta reducir los procesos a un número razonablemente pequeño que permita la distribución y uso de los recursos físicos:

    - Definir tareas de agentes que multiplexen un único thread de control entre múltiples objetos activos de una clase;
    - Los objetos cuya persistencia o vida está subordinada a un objeto activo también se ejecutan en el agente anterior
    - Muchas clases que requieren ser ejecutadas con exclusión mutua, o que requieren sólo un pequeño procesamiento deben compartir el mismo agente.

- **De afuera hacia adentro:** A partir de la arquitectura física:

    - Identificar los estímulos externos (requerimientos) al sistema, definir los procesos cliente para manejar los estímulos y procesos servidores que sólo brindan servicios y que no los inician.
    - Usar la integridad de los datos y las restricciones de serialización del problema para definir el conjunto correcto de servidores, y asignar objetos a los agentes cliente y servidor
    - Identificar cuáles objetos deben ser distribuidos.

El resultado es el mapeo de las clases (y sus objetos) en un conjunto de tareas y procesos de la arquitectura de procesos. Generalmente, existe una tarea agente para una clase activa con algunas variaciones: varios agentes para una clase dada para aumentar el throughput, o varias clases mapeadas en un mismo agente porque sus operaciones no se invocan frecuentemente o para garantizar su ejecución secuencial.
La Figura B-13 muestra cómo un pequeño conjunto de clases de un sistema de control de tráfico aéreo hipotético puede mapearse en procesos.

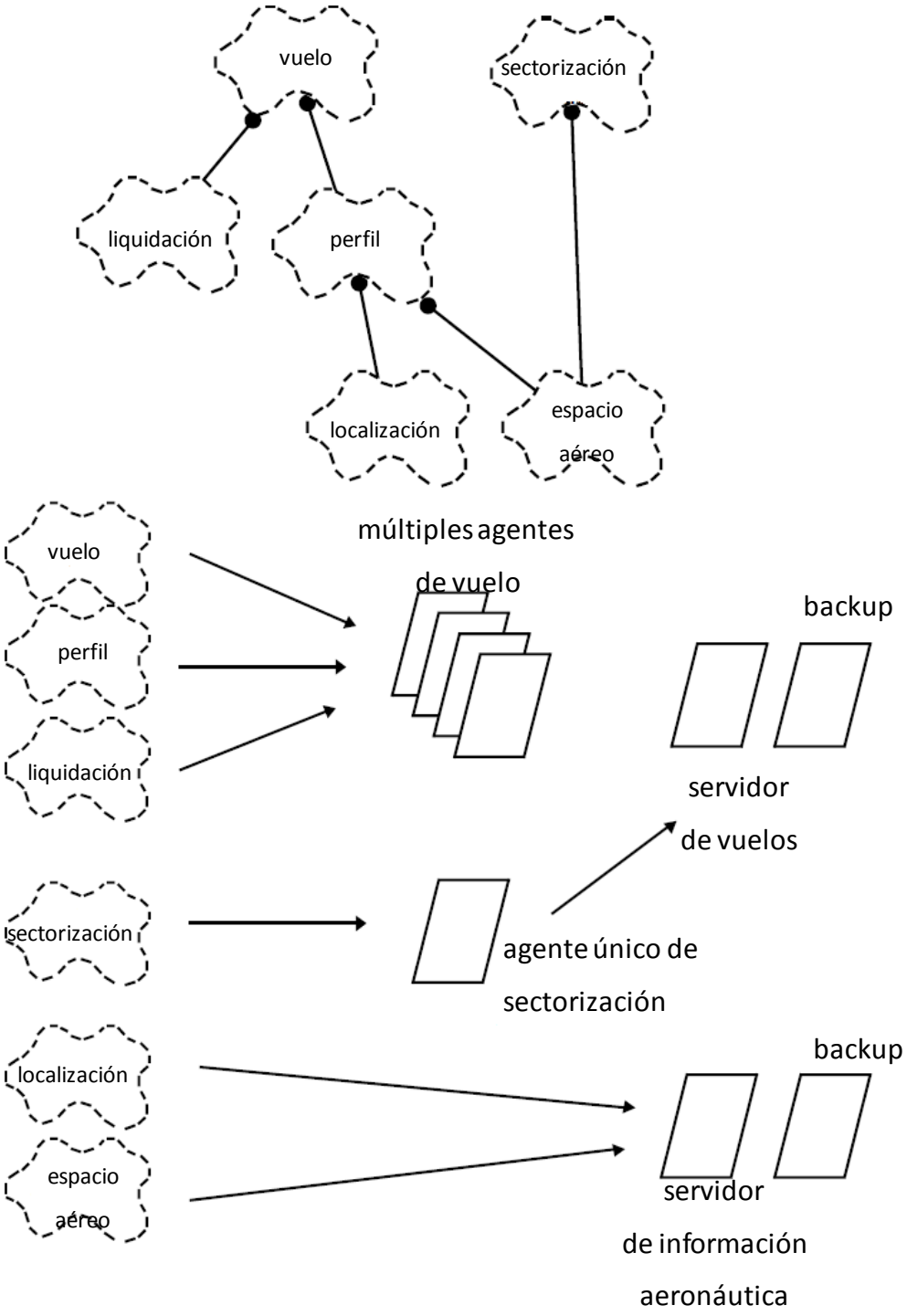


Figura B-13 Ejemplo de Mapeo de la Vista Lógica a la Vista de Procesos [2]

La clase vuelo se mapea a un conjunto de agentes de vuelo: existen muchos vuelos a procesar, una alta tasa de estímulos externos, el tiempo de respuesta es crítico, la carga debe distribuirse entre múltiples CPUs. Más aún, los aspectos de persistencia y distribución del procesamiento aéreo se difieren a un servidor de vuelos, el cual está duplicado por motivos de disponibilidad.

Un perfil de vuelo o una liquidación siempre están subordinadas a un vuelo, y a pesar que son clases complejas, ellas comparten los mismos procesos que la clase vuelo. Los vuelos se distribuyen en varios procesadores, de forma notable para el despliegue y las interfaces externas.
Una clase sectorización, que establece una partición del espacio aéreo para la asignación de jurisdicción de controladores de vuelos, debido a sus restricciones de integridad puede ser manejada solamente por un agente único, pero puede compartir el proceso servidor con el vuelo: las modificaciones son infrecuentes.
Localización y espacio aéreo y otra información aeronáutica estática son objetos protegidos, compartidos entre muchas otras clases, y raramente modificados; se mapean en su propio servidor, y se distribuye a otros procesos.

2.6.2. Desde la Vista Lógica a la Vista de Desarrollo.

Una clase se implementa generalmente como un módulo, por ejemplo un tipo de la parte visible de un paquete Ada. Las clases grandes se descomponen en múltiples paquetes. Colecciones de clases íntimamente relacionadas –categorías de clases– se agrupan en subsistemas. Deben también considerarse otras restricciones para la definición de subsistemas tales como la organización del equipo de desarrollo, el tamaño esperado del código (generalmente 5K a 20K SLOC por subsistema), grado de reuso y comonalidad[4] esperada, principio de distribución en capas (visibilidad), políticas de liberación, y administración de la configuración. Por lo tanto, generalmente se obtiene una vista que no tiene necesariamente una relación uno a uno con la vista lógica.
Las vistas lógica y de desarrollo son muy cercanas, aunque se refieren a distintos asuntos. Se ha encontrado [2] que cuanto mayor es el proyecto, mayor es también la distancia entre estas dos vistas. Al igual que para las vistas de procesos y física: cuanto mayor el proyecto, mayor es la distancia entre estas vistas. Por ejemplo, si se compara la Figura B-5 y la Figura B-11, no existe una correspondencia uno a uno de las categorías de clases y las capas. Si se considera la categoría "Interfaces Externas–Gateway", su implementación se distribuye a lo largo de varias capas: los protocolos de comunicación están en los subsistemas dentro o debajo de la Capa 1, los mecanismos generales de gateways están en los subsistemas de la Capa 2, y los gateways específicos reales están en los subsistemas de la Capa 5.

2.6.3. Desde la Vista de Proceso a la Vista de Física

Los procesos y grupos de procesos se mapean sobre el hardware físico disponible en varias configuraciones para testing o distribución. Birman describe algunos esquemas elaborados para realizar este mapeo dentro del proyecto Isis [2].
Los escenarios se relacionan esencialmente con la vista lógica, en términos de cuáles clases se usan y con la vista de procesos cuando las interacciones entre objetos involucran más de un thread de control.

[4] Se refiere a fragmentos de código comunes que permiten la definición de super-tipos.

2.7. Documentación de Arquitecturas

La documentación producida durante el diseño de la arquitectura se captura en dos documentos:

- Un *Documento de Arquitectura del Software*, cuya organización sigue las "4+1" vistas (ver la Figura B-14 por un punteo típico)
- Un documento de *Guías del Diseño del Software*, que captura (entre otras cosas) las decisiones de diseño más importantes que deben respetarse para mantener la integridad de la arquitectura del sistema.

Página de Título

Historia de cambios

Tabla de contenidos

Lista de figuras

1. Alcance

2. Referencias

3. Arquitectura del software

4. Objetivos y restricciones de la arquitectura

5. Arquitectura lógica

6. Arquitectura de procesos

7. Arquitectura de desarrollo

8. Arquitectura física

9. Escenarios

10. Tamaño y performance

11. Cualidades

Apéndices

A. Siglas y abreviaturas

B. Definiciones

C. Principios de diseño

Figura B-14 Punteo de un documento de Arquitectura de Software

2.8. Resumen del Modelo

La Figura B-15 muestra el resumen presentado en [2] de cada una de las vistas del modelo de vistas 4+1.

Vista	Lógica	Procesos	Desarrollo	Física	Escenarios
Componentes	- Clase	- Tarea	- Módulo -Subsistema	- Nodo	- Paso - Script
Conectores	- Asociación - Herencia - Contención	- Rendez-vous - Mensaje - Broadcast - RPC - etc.	- dependencia de compilación - sentencia "width" - sentencia "include"	- Medio de Comunicación - LAN - WAN - BUS	
Contenedores	- Categoría de Clase	- Proceso	- Subsistema (Biblioteca)	- Subsistema Físico	- Web
Stakeholders	- Usuario Final	- Diseñador - Integrador	- Desarrollador - Administrador	- Diseñador de Sistema	- Usuario - Desarrollador
Intereses	- Funcionalidad	- Performance - Disponibilidad - Tolerancia a Fallas - Integridad	- Organización - Reuso - Portabilidad - Línias de Productos	- Escalabilidad - Performance - Disponibilidad	- Comprensibilidad

Figura B-15 Resumen del Modelo de Vistas 4+1

ANEXO C. MODELO DE CUATRO VISTAS DE SIEMENS

En el trabajo de Robert L., et al [3] después de entrevistar a arquitectos de sistemas y diseñadores de un grupo de sistemas de distintos tipos (sistemas de procesamiento de imágenes y señales, sistema operativo de tiempo real, sistemas de comunicación, y sistemas de instrumentación y control) observaron que en muchos de ellos existía una distinción entre la arquitectura para un producto específico de software y la arquitectura para su plataforma. Para ambas, el producto de software y la plataforma de software, se utilizaban en diferentes etapas del proceso de desarrollo. Las estructuras que se encontraron describían los sistemas fueron clasificadas en:

- **Arquitectura Conceptual:** donde se describe el sistema en términos de sus elementos principales y las relaciones que existen entre ellos.
- **Arquitectura de Interconexión de Módulos:** comprende dos estructuras ortogonales, la descomposición funcional y de capas.
- **Arquitectura de Ejecución:** describe la estructura dinámica de un sistema.
- **Arquitectura de Código:** describe cómo es organizado el código fuente, librerías y binarios en el ambiente de desarrollo.

Se observó que la arquitectura del software juega un rol importante a través del proceso de desarrollo, y el logro de los niveles deseados de las propiedades no funcionales de los sistemas.

3.1. ¿Cómo se Identificaron las 4 Vistas de Arquitectura?

Para justificar la utilización de las cuatro vistas antes mencionadas, se consideró que inicialmente existía sólo código fuente. Luego, se introdujeron interfaces para que los compiladores u otras herramientas hicieran chequeo de tipos. Como el software comenzó a aumentar su tamaño fue útil dividir el código en archivos y el uso de técnicas de ocultamiento de información. Se introdujeron prácticas como la administración de la configuración y técnicas de construcción de sistemas para describir y manipular la organización a nivel de archivos de código fuente. Esta información pertenece a la Arquitectura de Código.

Como los sistemas comenzaron a ser cada vez más grandes, fue necesario que en los proyectos trabajaran muchos desarrolladores, hubiese mecanismos de abstracción, módulos, y la introducción de estructuras de interconexión de módulos.

Cuando los sistemas comenzaron a distribuirse, los programadores necesitaron incluir estructuras dinámicas y la comunicación, coordinación y sincronización. Fue necesario ubicar los componentes funcionales dentro de esas estructuras dinámicas. Aparecieron los lenguajes de interacción que enfrentaban la problemática de ubicar componentes en ambientes distribuidos. Incluyeron por ejemplo: Conic, Durra, Enterprice, Polylith, y UNAS. En las categorías de arquitecturas de software, estos problemas son enfrentados en la Arquitectura de Ejecución.

Luego, los ingenieros de software buscaron componer sistemas utilizando objetos que se comunicaban o conjuntos de componentes y conectores. Este grupo se categorizó con el nombre de Arquitectura Conceptual.

Arquitectura de Software	Ejemplos de Uso	Ejemplos de Factores de Influencia
Arquitectura de Código	• Administración de la Configuración • Técnicas de Construcción de Sistemas • Políticas de Precio de Equipos de Manufactura Original	• Lenguajes de Programación • Herramientas y ambientes de desarrollo • Subsistemas externos (ej. X Windows)
Arquitectura de Módulo	• Módulos de interfaces de administración y control • Análisis de Impacto de Cambios • Verificación de consistencia de restricciones de interfaces • Administración de la Configuración	• Potenciar tecnologías de software • Estructuras organizacionales • Principios de diseño
Arquitectura de Ejecución	• Análisis de Performance y Planificabilidad • Configuración estática y dinámica de sistema • Portabilidad de sistemas a diferentes ambientes de ejecución	• Arquitectura de hardware • Criterios de rendimiento de ambientes de ejecución • Mecanismos de comunicación
Arquitectura Conceptual	• Diseño utilizando componentes y conectores de dominios específicos • Estimación de performance • Análisis de seguridad y fiabilidad • Entendimiento de la configurabilidad estática y dinámica de sistemas	• Dominios de aplicación • Paradigmas de abstracción de software • Métodos de diseño

Tabla C-1 Factores de Uso e Influencia de las Arquitecturas de Software

3.2. Arquitectura Conceptual

La arquitectura conceptual describe un sistema desde los principales elementos de diseño y sus relaciones. Esta corresponde a una estructura del sistema de alto nivel, usando elementos y relaciones de diseño específicas del dominio del problema. Las arquitecturas conceptuales son independientes de las decisiones de implementación y se orientan a la interacción de los protocolos entre elementos de diseño.

3.3. Arquitectura de Módulos

La arquitectura de módulos ayuda a la programación a gran escala. Comprende dos estructuras ortogonales:

- **Descomposición Funcional**: captura la forma en que el sistema es descompuesto lógicamente en subsistemas, módulos y unidades abstractas de programa. Captura las interrelaciones entre componentes en términos de interfaces exportadas e importadas.
- **Descomposición por Capas**: refleja las decisiones de diseño basadas en base a relaciones de importación y exportación permitidas y las restricciones de interfaces. Reducen y aíslan las dependencias externas (por ejemplo: hardware y sistema operativo) e internas, y facilitan la construcción bottom-up y el testing de varios subsistemas.

A diferencia de la arquitectura conceptual, la arquitectura de módulos refleja las decisiones de implementación. Estas decisiones de implementación generalmente son independientes del lenguaje de programación. Por lo tanto esta arquitectura puede ser considerada como la estructura de implementación ideal.

3.4. Arquitectura de Ejecución

La arquitectura de ejecución es usada para describir la estructura dinámica de un sistema en términos de sus elementos (por ejemplo: tareas de sistema operativo, procesos, utilización de espacio), mecanismos de comunicación, asignación de funcionalidades a elementos en tiempo de ejecución, y asignación de recursos. Además describe las decisiones de diseño relacionadas a la ubicación, migración, y replicación de los elementos de sistemas en tiempo de ejecución.
Las fuerzas que apoyan esta categoría son el rendimiento, la distribución de los requerimientos, y el ambiente de ejecución, los que incluyen a la plataforma de hardware y software que la soporta. Por lo tanto, la arquitectura de ejecución es probable que cambie más frecuentemente debido a que la tecnología de hardware y software que apoya los elementos representados en ella. Las arquitecturas de ejecución son usadas para hacer análisis de performance, monitoreo, y puesta a punto y, también, para debugging y mantenimiento de servicios. A veces, la arquitectura de ejecución es trivial, como por ejemplo cuando el sistema corresponde a un único proceso.

3.5. Arquitectura de Código

La arquitectura de código es usada para organizar el código fuente en módulos de niveles de lenguaje, directorios, archivos y librerías. Esta arquitectura está influenciada por la elección del lenguaje de programación, las herramientas de desarrollo y ambiente (por ejemplo: administración de la configuración), y la estructura del proyecto (por ejemplo: entre empresas) y la organización. Es principalmente utilizada para facilitar la construcción del sistema, su instalación, y la administración de la configuración. También se usa para minimizar las dependencias (por ejemplo: de compilación) entre subprefectos y para reforzar las restricciones de importación y exportación especificadas en la arquitectura de módulos.

3.6. Relaciones entre las Arquitecturas

La relación entre la arquitectura conceptual y la arquitectura de ejecución se divide en dos partes:

1. Cada componente conceptual o un conector (o su implementación) puede estar relacionado a un elemento en tiempo de ejecución en la arquitectura de ejecución usando la relación asignado-a
2. Los mecanismos de comunicación seleccionados en la arquitectura de ejecución restringen la implementación de los componentes y conectores conceptuales.

La relación entre la arquitectura de módulos y la arquitectura de ejecución es similar.

1. Los módulos en la arquitectura de módulo son asignados-a elementos en tiempo de ejecución en la arquitectura de ejecución.
2. Cada elemento en la arquitectura de ejecución es implementado-por módulos específicos en la arquitectura de módulos

La relación entre la arquitectura de módulo y la arquitectura de código está caracterizada por la relación implementada-por. Asimismo, cada unidad abstracta de programa en la arquitectura de módulos es implementada-por unidades de programa concretos en la arquitectura de código.
Existe también una relación entre elementos en tiempo de ejecución en la arquitectura de ejecución y elementos como ejecutables y archivos de configuración en la arquitectura de código.
La Figura C-1 resume las relaciones entre las arquitecturas del modelo.

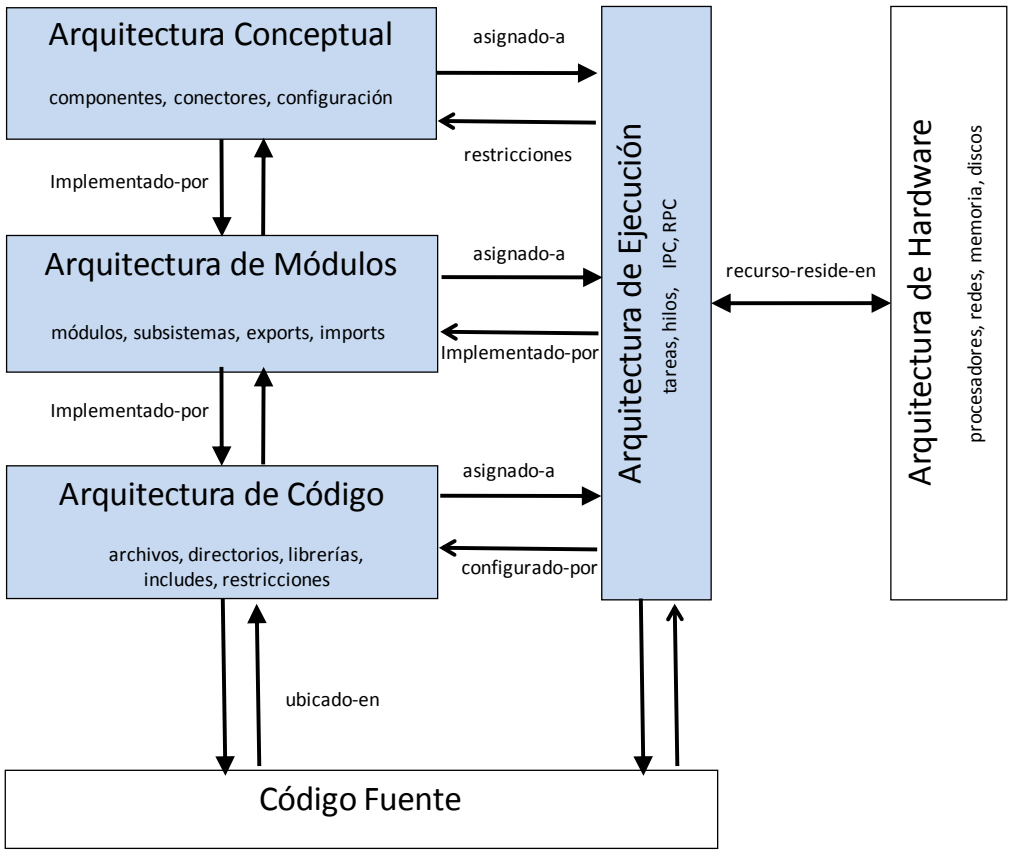


Figura C-1 Relación entre las cuatro vistas de Siemens [3]

ANEXO D. MODELO DE REFERENCIA ISO PARA EL PROCESAMIENTO ABIERTO Y DISTRIBUIDO (RM-ODP)

El RM-ODP (de sus siglas en inglés, Reference Model for Open Distributed Processing) es un modelo propuesto en 1998 por la ISO (International Standards Organization, http://www.iso.ch/) e ITU-T (International Telecommunication Union, conocido antes como CCITT, http://www.itu.int/)
Entre las contribuciones del RM-ODP se pueden destacar:

- El RM-ODP proporciona un marco de trabajo conceptual y una arquitectura que integra aspectos relacionados con la distribución, interoperabilidad y portabilidad de sistemas de software, permitiendo que la heterogeneidad del hardware, sistemas operativos, redes, lenguajes de programación, bases de datos y distintas formas de gestión sean transparentes al usuario.
- El RM-ODP proporciona un marco de coordinación para el procesamiento abierto y distribuido, que no sólo trata de aunar los estándares actuales de estos temas, sino que también dar oportunidad al desarrollo de nuevos estándares conforme vaya surgiendo la necesidad.
- El RM-ODP define de forma clara y precisa aquellos conceptos que aparecen en el desarrollo de plataformas de componentes distribuidos, proporcionando un vocabulario y un marco semántico común a todos los participantes y usuarios de las aplicaciones (desde los distintos departamentos de las empresas hasta los desarrolladores y usuarios finales). RM-ODP propicia, dentro de lo posible, el uso de técnicas de descripción formal para la especificación de estos conceptos y de la propia arquitectura, como pueden ser ESTELLE, LOTOS, SDL, o Z.

4.1. Normas Básicas del RM-ODP

El núcleo central del modelo de referencia de RM-ODP está basado en cuatro normas fundamentales que especifican su estructura y elementos básicos:

- **Visión de conjunto** (ISO/IEC 10746-1; ITU-T X.901). Esta norma contiene una visión de conjunto de las motivaciones de ODP, presentando el alcance, la justificación y la explicación de sus conceptos esenciales, así como una descripción de su arquitectura.
- **Fundamentos** (ISO/IEC 10746-2; ITU-T X.902). Esta norma contiene la definición (en lenguaje natural) de los conceptos básicos de RM-ODP, así como un marco analítico para la descripción normalizada de sistemas de procesamiento abierto y distribuido. Presenta también los principios disconformidad con las normas ODP, y la forma en que éstos deben aplicarse. En sólo 18 páginas, esta norma es capaz de sentar las bases de todo el modelo, de una forma clara, concreta, y precisa.
- **Arquitectura** (ISO/IEC 10746-3; ITU-T X.903). Esta norma contiene la especificación de las características que debe tener un procesamiento distribuido para que pueda ser considerado como abierto, así como las restricciones que deben cumplir aquellas normas que deseen integrarse en este modelo de referencia. Asimismo, define los distintos puntos de vista

(viewpoints) o subdivisiones que pueden hacerse de un sistema desde diferentes perspectivas, y que ayudan a su comprensión y especificación global.

- **Semántica arquitectural** (ISO/IEC 10746-4; ITU-T X.904). Esta norma contiene una formalización de los conceptos del modelo, utilizando para ello diferentes técnicas de descripción formal.

4.2. Conceptos Fundamentales

Las normas anteriores establecen una serie de conceptos fundamentales sobre los que se apoya el modelo para la normalización de sistemas abiertos y distribuidos:

- La especificación de un sistema en términos de especificaciones de diferentes puntos de vista (viewpoints), que están interrelacionados entre sí.
- El uso de un modelo de objetos común para la especificación del sistema desde cada uno de los puntos de vista.
- La definición de una infraestructura que permita ocultar ciertas complejidades inherentes a los sistemas distribuidos, simplificando la especificación y diseño de las aplicaciones (transparencias de distribución).
- La definición de un conjunto de funciones comunes que son de utilidad general para la especificación y construcción de sistemas abiertos y distribuidos.
- Un marco para evaluar la conformidad del sistema respecto a las propias normas que define ODP.

A continuación se examinará en más detalle algunos términos que aparecen involucrados en los conceptos fundamentales descritos anteriormente.

Puntos de Vista

En general, la extensión y complejidad de un sistema de información impiden que una sola persona pueda abarcar todos y cada uno de sus aspectos. Por otro lado, cada persona tiene sus propios intereses, necesidades, y perspectivas distintas desde donde abordar y examinar las especificaciones de un sistema: un ejecutivo realiza preguntas diferentes a las de un administrativo, o uno de los implementadores del sistema. Lo que RM-ODP proporciona es un marco de referencia mediante el cual es posible examinar, describir y especificar un sistema desde distintas perspectivas, denominadas puntos de vista. Cada uno de estos puntos de vista trata de satisfacer una audiencia distinta, cada una interesada en aspectos diferentes del sistema. Asociado a cada uno de estos puntos de vista se define un lenguaje especializado que recoge el vocabulario y la forma de expresarse de la audiencia concreta a la que se dirige.

RM-ODP define cinco puntos de vista genéricos, y que considera básicos a la hora de definir un sistema:

- **El punto de vista de la empresarial,** que describe los requisitos desde la perspectiva del propio negocio, así como la manera en la que pretende satisfacerlos. Se centra pues en la finalidad, alcance, entorno y políticas que rigen las actividades del sistema especificado, dentro de la organización de la que forma parte.

- **El punto de vista de la información,** que escribe el tipo de información que va a manejar el sistema, así como la estructura de los datos y sus posibles valores. Se centra en las clases de información tratadas por el sistema, su semántica, y las restricciones impuestas sobre la utilización e interpretación de dicha información.
- **El punto de vista computacional,** que describe la funcionalidad que ha de ofrecer el sistema, así como su descomposición y organización funcional. Para ello trata de describir el sistema como un conjunto de objetos que interactúan entre sí, definidos mediante interfaces.
- **El punto de vista de la ingeniería,** que describe la infraestructura necesaria para soportar el procesamiento distribuido del sistema, así como la forma de distribución de los datos y operaciones que permitan al sistema proporcionar la funcionalidad requerida.
- **El punto de vista de la tecnología**, encargado de describir la tecnología que soportará el sistema en base a la infraestructura de hardware, software y comunicaciones que permita el procesamiento y la funcionalidad necesaria, así como la representación y distribución de los datos.

Aunque el RM-ODP es un modelo agnóstico de la metodología, es muy utilizado para modelar sistemas distribuidos [7]. Este modelo se puede representar gráficamente en la Figura D-1.

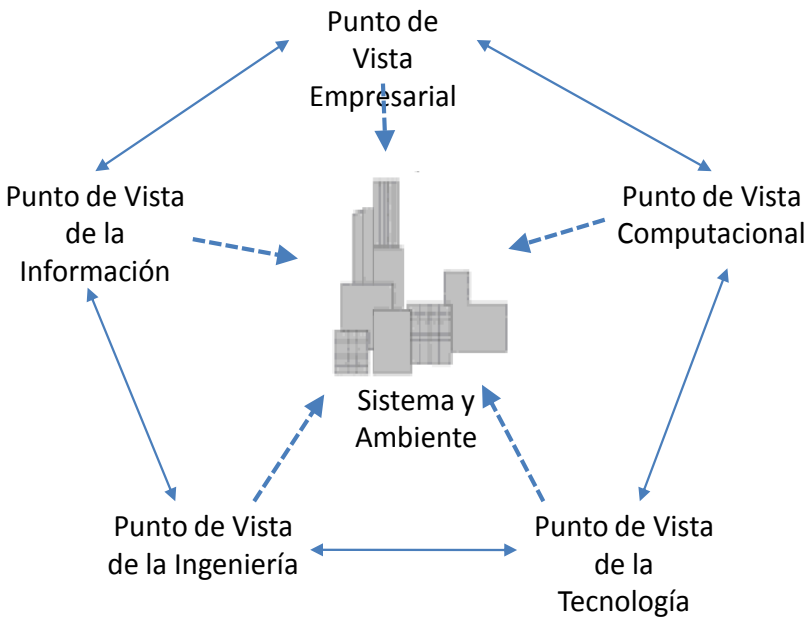


Figura D-1 Modelo de Referencia para el Procesamiento Distribuido Abierto [4]

El propósito de los puntos de vista en RM-ODP es posicionar los servicios relacionados con otros, para guiar la selección de modelos de servicios, y para ayudar en la ubicación de límites de ODP. El framework de puntos de vista se utiliza para dividir los problemas que se deben enfrentar al describir todas las facetas de un sistema ODP, por lo que esta tarea se hace más simple. Como resumen de los puntos de vista de ODP se puede considerar la Tabla D-1

Punto de Vista	Empresarial	Información	Computacional	Ingeniería	Tecnología

Punto de Vista	Empresarial	Información	Computacional	Ingeniería	Tecnología
Áreas de Preocupación	Necesidades de Sistemas de Información Empresariales	Modelos de Información, Estructuras de Información, Flujos de Información, Manipulación de la Información	Particionamiento lógico de aplicación, componentes de aplicación, interfaces de los componentes, interacciones de los componentes, vistas orientadas a servicios de aplicaciones distribuidas	Infraestructura de plataforma distribuida; transparencia de la distribución, apoyo a la comunicación, y otros habilitadores de distribución, de regulación, y mecanismos genéricos de ocultamiento; vistas orientadas a sistemas de aplicaciones distribuida	Artefactos tecnológicos necesarios para la construcción de mecanismos de ingeniería
Principales Conceptos	Agentes, los artefactos, las comunidades, roles, etc.	Esquemas, relaciones, los reglas de integridad, etc.	Objeto de cálculo, la interfaz computacional, las limitaciones de medio ambiente, las interacciones de cálculo, etc.	Los objetos básicos de la ingeniería, los objetos transparentes, el objeto del protocolo, núcleo, etc.	Soluciones tecnológicas que corresponde a los mecanismos de ingeniería y las estructuras
A quién le Afecta	Compradores del Sistema, Directivos de Empresa	Analistas de Información, Analistas de Sistemas, Ingenieros de Información	Diseñares de Aplicación y Programadores	Diseñadores de Sistemas Operativos, Diseñadores de Sistemas de Comunicación, Diseñadores del Sistema	Integradores de Sistemas, Proveedores de Sistemas
Lenguaje - Notación	Lenguajes de descripción de Requerimientos	Modelos de Entidad-Relación, Esquemas Conceptuales, etc.	Entornos de programación de aplicaciones, herramientas, lenguajes de programación, etc.	Plataformas Distribuidas, Ambientes de Soporte a la Ingeniería	Mapeos de tecnología, la identificación de los artefactos técnicos, etc.
Role en la Ingeniería de Software	Captura de Requerimientos y diseño temprano de sistemas distribuidos	Diseño Conceptual y Modelamiento de la Información	Diseño y construcción de software	Diseño y Desarrollo de Sistemas	Identificación de Tecnología, adquisición e instalación

Tabla D-1 Resumen de Puntos de Vistas de ODP

Uso de un Modelo de Objetos Común y de Técnicas de Orientación a Objetos

RM-ODP utiliza un modelo de objetos común como base para las distintas especificaciones, así como técnicas de orientación a objetos para modelar el sistema general y desde cada uno de los puntos de vista.

Estos métodos se muestran muy apropiados por las ventajas en cuanto a abstracción y encapsulación que proporcionan. La abstracción permite resaltar los detalles interesantes del

sistema sobre aquellos que no se consideran relevantes a un determinado nivel, y describirlos de forma independiente de cualquier implementación concreta. La encapsulación permite definir servicios y funciones ocultando detalles innecesarios sobre su implementación, su posible heterogeneidad, o los mecanismos de provisión de los servicios concretos utilizados por los clientes y servidores.

Transparencias

Las transparencias permiten ocultar las complejidades inherentes a cualquier sistema distribuido a la hora de especificarlo, permitiendo una mayor capacidad de abstracción. Entre ellas se destacan los siguientes aspectos:

- **Transparencia de acceso:** las diferencias de representación de datos y mecanismos de invocación para servicios entre sistemas heterogéneos.
- **Transparencia de fallo:** la ocultación de los posibles fallos que puedan ocurrir en otros objetos, así como de su recuperación, para permitir ciertos aspectos de tolerancia a fallos.
- **Transparencias de ubicación, migración y reubicación:** la búsqueda e invocación de servicios sin necesidad de especificar su ubicación, y con independencia de su migración o posible reubicación.
- **Transparencia de replicación:** la existencia de múltiples copias de un mismo servicio para proporcionar fiabilidad y disponibilidad.
- **Transparencia de transacciones:** la coordinación, supervisión y recuperación de las transacciones entre objetos de forma externa y sin tener que involucrar a los propios objetos.

RM-ODP define una serie de funciones y estructuras para realizar este tipo de transparencias. Sin embargo, cada una de ellas puede llevar asociado un compromiso entre la calidad de su funcionamiento requerido y su coste (ya sea en tiempo o en recursos); de igual forma, puede que no se desee que todas las transparencias sean relevantes en todos los casos. Por lo tanto, ODP no obliga a soportar todas estas transparencias, aunque sí indica que, en caso de soportar alguna de ellas, debe hacerse de acuerdo a sus normas para garantizar la conformidad con este estándar, y la integración con otros sistemas ODP.

Funciones comunes

Una ventaja muy importante en el desarrollo de aplicaciones distribuidas es poder contar con una serie de funciones generales para realizar muchos de los servicios que son precisos en estos ambientes. RM-ODP describe, como parte de su arquitectura básica, 24 funciones comunes a todos los sistemas abiertos y distribuidos. Dichas funciones están organizadas en cuatro grupos:

- Gestión
- Coordinación
- Repositorio
- Seguridad

Estas se utilizan también internamente en ODP para soportar algunos requisitos del lenguaje del punto de vista de ingeniería, y para implementar algunas de las transparencias. La existencia de esos servicios o funciones comunes ya definidos va a permitir la construcción de aplicaciones

distribuidas de una forma modular, y poder mejorar notablemente el tiempo de desarrollo y la fiabilidad del sistema resultante. En la arquitectura de RM-ODP sólo se describen brevemente estas funciones, dejando la especificación detallada de cada una de ellas para normas separadas. Una de las más importantes es la función de intermediación (trading), que por su relevancia será discutida más adelante.

Conformidad

Una de las ventajas de la existencia de normas internacionales es la posibilidad que ofrecen a los sistemas que son conformes a ellas de integrarse entre sí y asegurar su interoperabilidad. En los sistemas abiertos y distribuidos, la evolución y heterogeneidad de sus componentes obligan a disponer de mecanismos que permitan incorporar diferentes partes por separado, tanto en tiempo como en espacio, así como independientemente de los proveedores que las hayan construido. Por ello es muy importante que los comportamientos de las diferentes partes de un sistema estén claramente definidos, y que sea posible identificar mecanismos para garantizar que un cierto componente del sistema cumple sus especificaciones. El marco definido en RM-ODP para regir la evaluación de la conformidad trata estas cuestiones, abarcando distintas facetas: desde la identificación de los denominados puntos de conformidad hasta la especificación de la naturaleza de los enunciados de conformidad que habrán de hacerse en cada punto de vista, y la relación entre ellos.

4.3. Otras Normas de RM-ODP

Aparte de las cuatro normas básicas que definen los fundamentos del modelo de referencia, RM-ODP va construyéndose y completándose mediante nuevas normas que describen con detalle algunos de los aspectos y conceptos que inicialmente se mencionan en el modelo básico. En este apartado se describirán otros estándares que RM-ODP ofrece actualmente, y que corresponden a varios conceptos muy importantes dentro del modelo: la función de intermediación, el esquema de asignación de nombres, el lenguaje de definición de interfaces, y la forma de referenciar a dichas interfaces y vincular los correspondientes objetos.

La Función de Intermediación

El concepto de intermediación (trading) es fundamental en RM-ODP para la oferta y demanda de servicios distribuidos, y constituye una de las funciones comunes que se han mencionado anteriormente. Un intermediario o corredor de servicios (trader) es una entidad capaz de almacenar información acerca de posibles servicios, de forma que potenciales proveedores de los mismos puedan registrarse en él. Así, los clientes pueden enviar sus peticiones a dicho intermediario, quien se encarga de localizar a un proveedor de entre los que tiene registrados. Una vez ha localizado a alguno que satisfaga los requisitos que impone el cliente, envía la referencia de dicho servidor al cliente para que ellos puedan interactuar entre sí, sin necesidad del intermediario.

La norma que describe esta función de intermediación en RM-ODP (ISO/IEC 13235-1; ITU-T X.950) es la encargada de especificarla de forma independiente de cualquier implementación, asegurar que el servicio puede hacerse de forma federada entre varias funciones de intermediación distribuidas (sean o no del mismo fabricante), y ofrecer suficientes detalles que permitan evaluar alegaciones de conformidad a esta norma. Una característica importante de esta norma es que es capaz de

especificar la funcionalidad y el comportamiento de la función de intermediación utilizando un lenguaje de notación formal como es Z.
Por otro lado, la segunda de las normas que describen esta función (ISO/IEC 13235-3; ITU-T X.951) complementa a la anterior, especificando cómo la función de intermediación puede hacer uso del servicio de directorio X.500 tal y como lo define OSI (Open Systems Interconnection).

El Esquema de Asignación de Nombres

La asignación de nombres de forma global es un serio problema en los grandes sistemas abiertos y distribuidos, más aún cuando éstos pueden estar gestionados simultáneamente por más de una entidad. De ahí que RM-ODP especifique que los nombres han de asignarse de una forma que dependa del contexto, y en este sentido la norma ISO/IEC 14771 (ITU-T X.910) proporciona un esquema de asignación de nombres general dentro del marco de referencia.

El Lenguaje de Definición de Interfaces de ODP

Para poder especificar los servicios que ofrecen los objetos que forman parte un sistema abierto y distribuido, es preciso contar con algún lenguaje preciso, bien definido, e independiente de cualquier posible representación de los datos o estructuras que define, así como de la futura implementación de los objetos que especifica. La norma ISO/IEC 14750 (ITU-T X.920) define dicho lenguaje, al que se conoce como lenguaje de definición de interfaces de ODP, o ODP IDL por su acrónimo en inglés. Su principal objetivo es describir la signatura de los objetos que especifica, en términos de las estructuras de datos que se manejan y el perfil de las operaciones que definen sus servicios. De esta forma se consigue la ocultación necesaria para el desarrollo de aplicaciones abiertas. Además, el ODP IDL está totalmente en consonancia con el lenguaje de IDL que se define para CORBA, desarrollado por el consorcio internacional OMG (Object Management Group, http://www.omg.org/).

Referencias a Interfaces y Vinculación

Puesto que en un modelo de objetos los servicios vienen definidos por una interfaz, la forma de referenciar interfaces se convierte en fundamental para la interacción de objetos y sistemas en ODP, así como para conseguir la federación de grupos de sistemas. En RM-ODP, una referencia a una interfaz incluye la información necesaria para establecer vinculaciones entre objetos, incluidas aquellas entre objetos que soporten varios protocolos de comunicación, o que se encuentren ubicados en dominios de gestión distintos. Asimismo, una referencia a una interfaz ha de contener suficiente información para soportar la transparencia de reubicación, ya comentada anteriormente. La norma ISO/IEC 14753 (ITU-T X.930) es la encargada de especificar todos estos aspectos, fundamentales a la hora de construir aplicaciones distribuidas.

4.4. Normas Actualmente en Proceso

Aparte de las normas que se acaban describir, los comités de normalización de ISO e ITU-T trabajan actualmente en la elaboración de otras normas que formarán parte del modelo de referencia RM-ODP una vez sean consensuadas y publicadas. Estas normas profundizan sobre más aspectos fundamentales del modelo, como pueden ser la forma de interactuar de los objetos computacionales del mismo, la especificación de ciertas funciones comunes, o sobre cómo manejar los aspectos de calidad de servicio de una aplicación. En concreto, algunas de las normas actualmente en proceso de elaboración son:

- ODP-Protocol Support for Computational Interactions (ISO/IEC 14752; ITU-T X.931)
- ODP-Type Repository Function (ISO/IEC 14769; ITU-T X.960)
- ODP-Reference Model: Enterprise Viewpoint (ISO/IEC 15414; ITU-T X.911)
- ODP-Reference Model: Quality of Service (ISO/IEC 15935; ITU-T X.905)

4.5. Utilidad Práctica del Modelo

Una vez presentada la estructura y composición básica de ODP, su motivación, y las principales ventajas que ofrece este modelo de referencia para el desarrollo de sistemas abiertos y distribuidos, cabe preguntarse por la utilidad práctica que tiene. De hecho, ODP suele verse a un nivel demasiado abstracto y elevado para que pueda tener aplicación directa en las empresas, o en los desarrollos tradicionales. Sin embargo, esto no tiene por qué ser así. En primer lugar, ya existen aplicaciones concretas que se han basado en RM-ODP para ser especificadas, diseñadas, y construidas, así como empresas que están utilizando este modelo de referencia para tratar de organizar sus aplicaciones de IT de acuerdo a un norma consensuada internacionalmente y con amplias perspectivas de futuro (desde bancos suizos a empresas de telecomunicaciones norteamericanas). En segundo lugar, ya existe tecnología que soporta este modelo de referencia lo suficientemente madura y extendida como para que construir aplicaciones basándose en él no sea una apuesta arriesgada. Por ejemplo, la plataforma de objetos distribuidos CORBA (Common Object Request Broker Architecture) definida por OMG, está totalmente en concordancia con RM-ODP, y presenta soluciones tecnológicas muy probadas que soportan varios de los puntos de vista. En particular, CORBA ofrece una plataforma de componentes distribuidos junto con una serie de servicios y facilidades comunes (similares a las funciones comunes de RM-ODP) para el desarrollo de aplicaciones distribuidas en ambientes heterogéneos y multidisciplinares. Y en el campo de las telecomunicaciones, TINA (Telecommunications Information Networking Architecture) definida por TINA-C (TINA Consortium) es una arquitectura para el desarrollo de aplicaciones de este tipo basada en los conceptos que ofrece RM-ODP, y que actualmente proporciona la infraestructura más rica y consolidada dentro de esta disciplina. RM-ODP ofrece varias ventajas muy interesantes, entre las que se destacan las siguientes:

- En primer lugar, obliga a pensar desde diferentes perspectivas o puntos de vista, lo que permite un mejor análisis y recolección de los requisitos que se deben imponer a los sistemas.
- En segundo lugar, proporciona una infraestructura y un modelo común desde donde los requisitos expresados en diferentes lenguajes (los de los distintos puntos de vista) pueden ser integrados para formar un sistema globalmente consistente.
- En tercer lugar, proporciona toda una serie de patrones de razonamiento ya establecidos sobre los cuales es posible apoyarse a la hora de especificar y diseñar un sistema, y poder identificar sus elementos fundamentales y las relaciones entre ellos. En este sentido, RM-ODP ayuda a hacer las preguntas adecuadas a los responsables apropiados, con un nivel suficiente de abstracción y de precisión para analizar correctamente los sistemas.
- Y por último, RM-ODP ofrece un conjunto de mecanismos muy útiles a la hora de diseñar y desarrollar aplicaciones distribuidas, junto con un soporte tecnológico suficientemente

maduro como para construir aplicaciones robustas, eficientes y competitivas, a la vez que integrables con otros sistemas que cumplan estos estándares.