7. Introducción a los Sistemas de Información, Arquitectura de Software, Integración de Sistemas y Middleware

Ing. Raquel Sosa

Ing. Bruno Rienzi

Arquitectura de un Sistema de Información

Definiciones

Comenzaremos por repasar algunas definiciones y conceptos sobre qué es la Arquitectura de Software:

"… más allá de los algoritmos y estructuras de datos de la computación, diseñar y especificar la estructura del sistema completo emerge como un nuevo tipo de problema. Los aspectos estructurales incluyen la organización gruesa y la estructura de control global; protocolos para comunicación, sincronización, y acceso a datos; asignación de funcionalidad a elementos de diseño; distribución física; composición de elementos de diseño; escalabilidad y performance; y selección de alternativas de diseño." (Garlan y Shaw, 1993)

"Una arquitectura es el conjunto de decisiones significativas sobre la organización de un sistema de software, la selección de los elementos estructurales y sus interfaces que componen el sistema, junto con su comportamiento tal como se especifica en las colaboraciones entre esos elementos, la composición de estos elementos estructurales y comportamentales en subsistemas progresivamente más grandes, y el estilo arquitectónico que guía dicha organización." (Kruchten, 1999).

"La descomposición de mayor nivel de un sistema en sus partes; las decisiones que son duras de cambiar; hay muchas arquitecturas en un sistema; lo que es arquitectónicamente significante puede cambiar durante el ciclo de vida de un sistema; y, al final, la arquitectura se reduce a las cosas realmente importantes" (Fowler).

"La organización fundamental de un sistema, encarnada en sus componentes, sus relaciones entre sí y con el entorno, y los principios que gobiernan su diseño y evolución." (ANSI/IEEE Std 1471-2000)

"La arquitectura de software de un programa o sistema de cómputo es la estructura o estructuras de un sistema, que comprenden elementos de software, las propiedades externamente visibles de esos elementos y las relaciones entre ellos. La arquitectura se refiere a la parte pública de las interfaces; los detalles privados de los elementos – detalles que tienen que ver sólo con la implementación interna – no son arquitectónicos." (Bass, Clements, Kazman, 2003).

Podemos ver cómo muchos conceptos se repiten, a veces con diferente terminología, entre las distintas definiciones. Tomaremos como punto de partida la última definición para analizar los principales conceptos.

En primer lugar, esta definición deja en claro que la arquitectura puede estar compuesta por más de una estructura. En sistemas complejos, no existe una única estructura que pueda considerarse "la" arquitectura; existen diversas estructuras que especifican la arquitectura bajo diferentes puntos de vista. Esto lo veremos más claramente cuando estudiemos el modelo "4+1".

La arquitectura define elementos. En prácticamente todos los sistemas modernos, los elementos interactúan a través de interfaces que permiten dividir un elemento es sus partes pública y privada. Además de las propiedades de los elementos, el comportamiento de los

elementos es parte de la arquitectura, siempre y cuando este comportamiento pueda ser observado desde el punto de vista de otro elemento.

La definición implica que todo sistema de software tiene una arquitectura, ya que todo sistema, por más simple o complejo que sea, puede ser descompuesto en elementos y relaciones entre ellos (en el caso más trivial, tendríamos un único elemento y ninguna relación). Sin embargo, no todo sistema posee una especificación o descripción conocida, es decir, documentada. En algunos casos, es necesario reconstruir la arquitectura a partir del código.

Finalmente, notamos que la definición no nos dice qué características debe tener una arquitectura para ser considerada "buena" o "mala".

Principios de Diseño

Existen algunos principios de diseño muy difundidos y aplicados que permiten diseñar una arquitectura que minimice los requerimientos de mantenimiento y costos, y promueva la usabilidad y extensibilidad. Los principios clave son:

Separación de Intereses. La aplicación debe dividirse en componentes distintivos que no se solapen en funcionalidad. Esto permite reducir los puntos de interacción para lograr alta cohesión y bajo acoplamiento.

Principio de Responsabilidad Única. Cada componente debe ser responsable por una única característica o funcionalidad, o una agregación de funcionalidades cohesivas.

Principio de Mínimo Conocimiento (también conocido como "Ley de Demeter"). Un componente no debería conocer los detalles internos de otros componentes.

No Te Repitas (DRY, "Don't repeat yourself"). Una funcionalidad debe ser brindada por un único componente, es decir, no debe haber redundancia de funcionalidades.

Minimizar el diseño inicial. Se debe evaluar la necesidad de invertir un gran esfuerzo en el diseño inicial de la aplicación (BDUF, "Big Design Upfront"). En algunas aplicaciones, un diseño inicial abarcativo puede ser necesario si el costo de desarrollo o de una falla en el diseño puede ser muy alto. En otros casos, especialmente si se siguen metodologías ágiles de desarrollo o si los requerimientos son muy cambiantes, se recomienda un diseño inicial básico que permita comenzar con la prototipación e implementación en forma temprana.

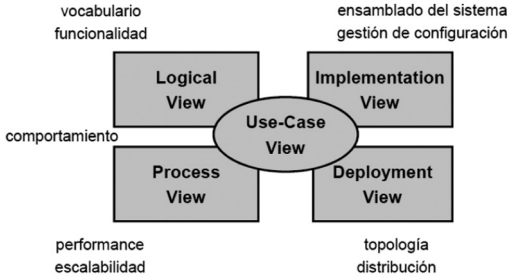
Cuando se diseña una aplicación o sistema la meta del arquitecto de software es minimizar la complejidad separando las áreas de interés. Por ejemplo, la interfaz de usuario (UI), el procesamiento de la lógica de negocio y el acceso a datos, son diferentes áreas de interés. Sin embargo, el arquitecto debe tener en cuenta también la relación costo/beneficio de una decisión de diseño en tal sentido. Esto podría llevar a optar por diseños más sencillos, aunque menos "puros" del punto de vista teórico.

Documento de Arquitectura y Vistas 4+1

La documentación de la arquitectura es una ayuda indispensable para los desarrolladores que necesitan comprender las ideas esenciales del sistema. Tener una arquitectura es una cosa, tener una descripción útil de la arquitectura es otra. Una forma común en la actualidad de describir una arquitectura es utilizar múltiples vistas. Una vista es una perspectiva del sistema que permite focalizarse en la estructura, modularidad, componentes esenciales y principales flujos de control. Podemos pensar en una vista como una ventana hacia el sistema, en

donde se destacan ciertos aspectos y se ignoran otros. Cada vista ilustra cómo la arquitectura permite concretar los objetivos o intereses de un determinado stakeholder (persona, grupo o entidad que tiene un interés particular en el sistema y que puede afectar o ser afectado por el mismo).

El Modelo de Vistas 4+1 de Kruchten (ver Figura 1) describe la arquitectura en cuatro vistas (lógica, de procesos, de distribución y de implementación) más una vista (la +1) de casos de uso. A continuación describiremos brevemente estas vistas.



• **Vista Lógica**. Organización conceptual del software en términos de los elementos más importantes que lo componen (subsistemas, componentes, capas, paquetes, frameworks, interfaces, clases, etc.). Resume la funcionalidad principal de cada componente. Muestra los aspectos dinámicos del sistema mediante diagramas de interacción de algunos escenarios de los casos de uso críticos. En un enfoque top-down, se comienza por descomponer el sistema en un conjunto de subsistemas "grandes", como ser las "capas" (layers) si se utiliza una arquitectura en capas y, a partir de ellos, se realizan sucesivos refinamientos hasta llegar a las unidades lógicas más pequeñas. En el Proceso Unificado, la vista lógica es una vista del Modelo de Diseño.

• **Vista de Procesos.** Muestra procesos e hilos (theads) con sus responsabilidades, colaboraciones y relaciones con elementos de la vista lógica. Se utilizan diagrama de clase (con estereotipos <<thread>> y <<process>>) para mostrar la estructura estática y diagramas de actividad para mostrar la sincronización y comunicación. En el Procesos Unificado, la vista de procesos es una vista de Modelo de Diseño.

• **Vista de Distribución.** Distribución física de componentes y procesos a nodos de procesamiento y configuración de la red física entre dichos nodos. Un nodo puede ser un elemento de hardware (un host, un router, etc.) o un elemento de software (un servidor de aplicaciones JEE, por ejemplo). Para distinguir los nodos físicos (hardware) se utiliza el estereotipo <<device>>. Para cada escenario planteado interesa particularmente la descripción de los requerimientos no funcionales que justifican dicho escenario. De los nodos pueden especificarse requerimientos de software y hardware (sistema operativo, procesador, memoria, almacenamiento secundario, etc.) y de las conexiones los protocolos de comunicación, ancho de banda, etc. En el Proceso Unificado, se corresponde con el Modelo de Distribución. Esta vista, a diferencia de las otras, suele incluir el sistema completo y no un subconjunto.

• **Vista de Implementación.** Muestra los componentes implementados del sistema y sus dependencias (ejecutables, paquetes de clases, módulos, bibliotecas, etc.). En el Proceso Unificado es una vista del Modelo de Implementación.

• Casos de Uso. Resumen de los casos de uso más significativos para la arquitectura y sus requerimientos no funcionales. Los casos de uso deben elegirse de acuerdo a criterios de cobertura (que intervengan el mayor número de componentes del sistema), complejidad, riesgo tecnológico y valor para el cliente. En el Proceso Unificado, es una vista del Modelo de Casos de Uso.

• Generalmente, una vista es más interesante que otra para un determinado stakeholder. Por ejemplo, la vista lógica para un usuario final, la vista de distribución para un administrador de sistemas, etc. A partir de este modelo se elaboraron otros modelos identificando nuevas vistas (vista de datos, vista de seguridad, etc.), por lo que se generalizó como modelo de vistas N+1.

• El documento de arquitectura, conocido por sus siglas en inglés como SAD (Software Architecture Document), suele estructurarse en base a la división en vistas. Dentro de este curso se proporciona una plantilla en donde se muestra más detalladamente el contenido del mismo.

Bibliografía

M.P.&.P. Team, Microsoft Application Architecture Guide, Microsoft Press, 2009.

C. Larman, Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development, Prentice Hall, 2004.

Patrones de arquitectura

1 Estilos Arquitectónicos

1.1 Introducción

Un estilo arquitectónico, también llamado patrón arquitectónico, es un patrón abstracto que permite describir la arquitectura de una familia de sistemas. Un estilo arquitectónico mejora el particionamiento y promueve el reuso de diseños al proveer soluciones a problemas frecuentes de diseño de sistemas.

Garlan y Shaw definen el estilo arquitectónico como:

"… una familia de sistemas en términos de un patrón de organización estructural. Más específicamente, un estilo arquitectónico determina el vocabulario de componentes y conectores que pueden ser usados en instancias de ese estilo, junto con un conjunto de restricciones sobre cómo pueden ser combinados. Estos pueden incluir restricciones topológicas en descripciones arquitectónicas (ej. sin ciclos). Otras restricciones – digamos, que tienen que ver con la semántica de ejecución – podrían también ser parte de la definición del estilo."

La comprensión de los estilos arquitectónicos provee varios beneficios. El beneficio mayor es que proveen un lenguaje común, así como la posibilidad de discutir sobre la arquitectura en términos de patrones y principios sin entrar en detalles sobre las tecnologías.

En la tabla X se muestra una clasificación de varios estilos arquitectónicos de acuerdo a un área clave de interés (comunicación, distribución, dominio, estructura). Existen otros estilos arquitectónicos que no se encuentran habitualmente en sistemas de información, tales como Pipes & Filters (ej. comunicación de procesos mediante pipes en UNIX, compiladores, etc.) y Pizarrón (procesamiento de señales, reconocimiento de patrones, etc.).

Categoría	Estilos Arquitectónicos
Comunicación	Arquitectura Orientada a Servicios (SOA), Bus de Mensajes
Distribución	Cliente/Servidor, 3-Niveles, N-Niveles
Estructura	En Capas, Basada en Componentes, Orientada a Objetos
Dominio	Diseño Guiado por el Dominio (DDD)

La arquitectura de un sistema de software de mediano o gran porte no suele limitarse a un único estilo sino que, generalmente, es una combinación de estilos arquitectónicos que se utilizan en diferentes subsistemas o niveles de abstracción. Por ejemplo, en una Arquitectura Orientada a Servicios, cada servicio puede ser diseñado mediante una Arquitectura en Capas y cada capa mediante un Arquitectura Basada en Componentes. A su vez, la comunicación entre servicios puede utilizar una Arquitectura de Bus de Mensajes.

1.2 Arquitectura Orientada a Servicios (SOA)

Una Arquitectura Orientada a Servicios (SOA) permite que la funcionalidad de una aplicación sea ofrecida como un conjunto de servicios. Estos servicios están débilmente acoplados ya que utilizan interfaces estándar que pueden ser invocadas, publicadas y descubiertas. Los servicios en una SOA proveen un esquema y una interacción con las aplicaciones basada en mensajes a nivel de aplicación y no a nivel de componentes u objetos.

Los principios fundamentales de una SOA son:

• Los servicios son autónomos. Cada servicio es mantenido, desarrollado, desplegado y versionado en forma independiente.

• Los servicios son distribuibles. Pueden ser alojados en cualquier host, localmente o remotamente, con la única condición de que la red soporte los protocolos de comunicación requeridos.

• Los servicios están débilmente acoplados. Cada servicio puede ser reemplazado o actualizado sin afectar a las aplicaciones, siempre y cuando mantenga una interfaz compatible.

• Los servicios comparten esquema y contrato, no clases. En la comunicación entre servicios se comparten esquemas y contratos, pero nunca clases internas.

• La compatibilidad se basa en políticas. Las políticas, en este caso, refieren a protocolos y seguridad.

Los principales beneficios de una SOA son:

• Alineamiento entre Negocio y Tecnología. Los servicios implementados representan servicios del mundo real del negocio.

• Bajo Acoplamiento y Abstracción. Servicios autónomos y accesibles mediante contratos formales.

• Descubribilidad. Exponen descripciones que permiten que otros sistemas los descubran automáticamente.

• Interoperabilidad. Protocolos y formatos de datos estándares.

• Racionalización. La granularidad de los servicios puede ser la necesaria para evitar la duplicación de funcionalidades.

Ejemplos habituales de aplicaciones orientadas a servicios incluyen aplicaciones que comparten información, procesos en varios pasos como los sistemas de reservas y las tiendas online, mashups que combinan información de varias fuentes, etc.

1.3 Bus de Mensajes

La arquitectura de bus de mensajes describe un sistema que puede recibir y enviar mensajes usando uno o más canales de comunicación, de manera que varias aplicaciones pueden interactuar sin conocer detalles una de la otra. La interacción de las aplicaciones se logra mediante intercambio de mensajes (típicamente asincrónicos) a través de un bus común.

Las principales características de una arquitectura de bus de mensajes son:

• Comunicaciones basadas en mensajes.

• Lógica de procesamiento compleja. Las operaciones complejas se ejecutan como un conjunto de operaciones de menor complejidad, cada una de las cuales ejecuta una tarea en un itinerario de múltiples pasos.

• Modificaciones a la lógica de procesamiento. Dado que la interacción con el bus está basada en esquemas y comandos comunes, se pueden insertar o remover aplicaciones en el bus para cambiar la lógica que se usa para procesar mensajes.

Los principales beneficios de una arquitectura de bus de mensajes son:

• Interoperabilidad. Al utilizar un modelo de comunicaciones basado en estándares se puede interactuar con aplicaciones desarrolladas en diferentes entornos, como ser .NET y Java.

• Extensibilidad. Se pueden agregar o remover nuevas aplicaciones.

• Baja complejidad. Una aplicación sólo se comunica con el bus.

• Flexibilidad. Muchas modificaciones se realizan sólo a nivel de configuración.

• Bajo acoplamiento. Dependencia de interfaces, no de aplicaciones.

• Escalabilidad. El bus permite que se agregan varias instancias de una misma aplicación.

Dos variantes de esta arquitectura son el Enterprise Service Bus (ESB) y el Internet Service Bus (ISB).

1.4 Cliente/Servidor

La arquitectura cliente servidor describe un sistema distribuido que involucra un nodo central (el servidor) y varios nodos (los clientes) que se comunican con el primero a través de la red de comunicaciones. Esta arquitectura también se llama arquitectura de dos niveles.

Históricamente, las aplicaciones clientes/servidor se componían de una aplicación de escritorio que se comunicaba con un DBMS o con un servidor de archivos dedicado. Los DBMS contenían gran parte de la lógica de negocio en forma de procedimientos almacenados.

En esta arquitectura, un cliente, posiblemente provisto de una interfaz gráfica de usuario (GUI), inicia la comunicación enviando una solicitud al servidor, espera por la respuesta y procesa la respuesta cuando llega. El servidor autoriza al cliente y ejecuta la lógica necesaria para generar la respuesta, la que se envía al cliente.

Los principales beneficios de la arquitectura cliente/servidor son:

• Seguridad. Todos los datos se almacenan en el servidor, lo que generalmente es más seguro que almacenarlo en los clientes.

• Acceso a datos centralizado. Esto facilita la administración de los datos.

• Facilidad de mantenimiento. Del lado del servidor, existen varios hosts que pueden tomar el mismo rol, por lo que los clientes no se ven afectados por problemas que surjan con un servidor.

Algunas variaciones de la arquitectura cliente/servidor son: sistemas client-queu-client, sistemas peer-to-peer (p2p), servidores de aplicaciones.

1.5 3 Niveles y N Niveles

La arquitectura en tres niveles es una evolución de la arquitectura cliente/servidor que permite solucionar el problema de acoplamiento entre los datos y lógica de negocio, que suele darse en arquitecturas de dos niveles.

Una arquitectura en tres niveles permite descomponer la funcionalidad de la aplicación en segmentos, de la misma forma que en una arquitectura en capas, pero con la particularidad

de que cada segmento puede ser alojado en una computadora diferente. A veces los niveles (tiers, en inglés) son llamados capas físicas, pero en este documento usaremos la palabra niveles para diferenciarlos claramente de las capas lógicas (layers, en inglés).

La arquitectura de N niveles es una generalización de la anterior, en donde N es un número mayor o igual a 3. Cada nivel es independiente de los demás, excepto de los niveles inmediatamente encima y debajo. El nivel N debe saber cómo manipular una solicitud proveniente del nivel N+1 y cómo realizar una nueva solicitud al nivel N-1 (si N no es 0). La comunicación entre niveles es típicamente asincrónica.

Los principales beneficios de la arquitectura en N niveles son:

- Mantenibilidad. Permite realizar cambios en un solo nivel.
- Escalabilidad. Es sencillo agregar más equipos en un determinado nivel.
- Flexibilidad. La administración de cada nivel se realiza en forma independiente de los otros.
- Disponibilidad. Los tres factores anteriores también inciden positivamente en la disponibilidad del sistema.

1.6 En Capas

La Arquitectura en Capas agrupa la funcionalidad relacionada de una aplicación o sistema en capas que se apilan verticalmente.

La Arquitectura en Capas ha sido descripta como una pirámide invertida de reuso, donde cada capa agrega las responsabilidades y abstracciones de la capa inmediatamente debajo de ella.

Las capas de una aplicación podrían residir en la misma máquina física (el mismo nivel) o podrían estar distribuidas en varias máquina separadas (N niveles), y los componentes de una capa se comunicarían con los de otra a través de interfaces bien definidas. Una Arquitectura en Capas, por lo tanto, no implica una Arquitectura en N Niveles, pero una Arquitectura en N Niveles sí implica una Arquitectura en Capas.

Por ejemplo, una típica arquitectura de aplicación Web consiste en una capa de presentación (funcionalidad relacionada con la GUI), una capa de negocio (procesamiento de reglas de negocio) y una capa de persistencia (acceso a datos, casi siempre implementada mediante frameworks de alto nivel).

Los principios fundamentales de una arquitectura en capas son:

- Abstracción. La arquitectura en capas abstrae la visión del sistema como un todo, al mismo tiempo que provee suficientes detalles para entender los roles y responsabilidad de cada capa y las relaciones entre ellas.
- Encapsulamiento. No se muestran tipos de datos, métodos o propiedades cuando se trabaja a nivel de capas.
- Separación funcional claramente definida. Impuesta por la arquitectura misma.
- Alta cohesión. Cada capa agrupa funcionalidades directamente relacionadas.
- Reusabilidad. Una capa inferior no tiene dependencias sobre la capa superior, permitiendo que se reuse en otras aplicaciones.
- Bajo acoplamiento. La comunicación entre capas se basa en abstracción y eventos, para lograr el bajo acoplamiento entre las capas.

La arquitectura en capas puede ser estricta o no. En una arquitectura en capas estricta los componentes de una capa puede interactuar únicamente con los componentes de la misma capa o de la capa inmediatamente debajo. En una arquitectura en capas no estricta los componentes podrían interactuar, además, con cualquier capa inferior.

Para separar la capa de presentación del resto de la aplicación suelen utilizarse patrones tales como MVC (Model-View-Controller), Passive View y Supervising Presenter.

1.7 Basada en Componentes

La arquitectura basada en componentes descompone el sistema en componentes funcionales o lógicos que exponen interfaces bien definidas conteniendo métodos, eventos y propiedades. Los componentes son abstracciones de mayor nivel que los objetos.

Los componentes se caracterizan por ser:

• Reusables. Aptos para diferentes escenarios y aplicaciones, por lo que la información del contexto (estado de los datos) debe ser suministrada al componente y no obtenida por él mismo.

• Reemplazables. Componentes con la misma interfaz y funcionalidad son intercambiables.

• Extensibles. Un componente nuevo puede extender a otro existente para proveer un nuevo comportamiento.

• Encapsulamiento. No se muestran detalles internos del componente, sólo interfaces.

• Independientes. Mínima dependencia entre componentes.

Los componentes dependen de mecanismos dentro de cada plataforma que provea un entorno en el que puedan ejecutar. Ejemplos de estos son COM+, DCOM y ActiveX en sistemas Windows, CORBA y EJB en varias plataformas. Estas arquitecturas de componentes manejan la mecánica de localizar componentes y sus interfaces, pasar mensajes y comandos entre componentes y, en algunos casos, mantener el estado.

Dentro de los tipos de componentes, los más comunes son los controles que forman partes de una GUI, los componentes remotos y los componentes que se acceden mediante colas de mensajes.

Los principales beneficios de una arquitectura basada en componentes son:

• Mantenibilidad. Es fácil reemplazar un componente por una nueva versión compatible sin impactar sobre el resto de la aplicación.

• Reducción de costos. Se pueden utilizar muchos componentes desarrollados por terceros. Se pueden reusar componentes desarrollados para una aplicación en otra aplicación.

• Mitigación de la complejidad tecnológica. Los componentes ejecutan dentro de un contenedor que brinda una cantidad de servicios que no deben ser desarrollados para cada componente, sino que simplemente se utilizan o configuran (manejo del ciclo de vida, seguridad, comunicación, transacciones, etc.)

Algunos patrones de diseño utilizados para manejar las dependencias entre componentes son Inyección de Dependencias y Localizador de Servicios.

1.8 Orientada a Objetos

La arquitectura orientada a objetos divide las responsabilidades de una aplicación o sistema en objetos que poseen su propio estado y comportamiento y que cooperan entre ellos.

Los objetos se comunican a través de llamadas a métodos o acceso a propiedades de otros objetos, utilizando interfaces bien definidas.

La arquitectura orientada a objetos tiene sentido para describir aplicaciones muy pequeñas (recordemos que toda aplicación, por más simple que sea, tiene una arquitectura) programadas utilizando lenguajes orientados a objetos. En la mayoría de las aplicaciones se utiliza algún otro estilo arquitectónico basado en abstracciones de mayor nivel (ej. capas, componentes, etc.) que se complementa y refina con otros modelos provenientes del Análisis y Diseño Orientado a Objetos (OOAD).

1.9 Diseño Guiado por el Dominio

El Diseño Guiado por el Dominio (DDD) es un enfoque orientado a objetos para diseñar software a partir de los objetos del dominio, con sus comportamientos y relaciones. El equipo de desarrollo trabaja junto con los expertos del negocio en la definición de un modelo de dominio y de un vocabulario común que excluya tecnicismos y ambigüedades. El software construido es una proyección directa del modelo.

Los principales beneficios del Diseño Guiado por el Dominio son:

• Comunicación. Facilita la comunicación entre el equipo de desarrollo y los expertos del negocio.

• Extensibilidad. El modelo de dominio suele ser lo suficientemente modular y flexible para poder ser fácilmente adaptado y extendido a medida que cambian las condiciones y los requerimientos.

• Verificable. Los objetos del dominio poseen alta cohesión y bajo acoplamiento, por lo que son más fáciles de testear.

2 Arquitectura en Capas

La arquitectura más utilizada en los sistemas de información es la arquitectura en capas. Dentro de esta arquitectura, las capas más habituales son: Presentación, Servicios, Negocio y Persistencia. No es necesario que una arquitectura posea todas estas capas (la capa de Presentación sólo es necesaria si la aplicación interactúa con usuarios, la capa de Servicios sólo es necesaria si la aplicación expone funcionalidades a sistemas externos).

En sistemas de información empresariales, es habitual que las capas lógicas se distribuyan en diferentes máquinas, por lo que también tendremos una arquitectura de N-niveles.

2.1 Capa de Presentación

La capa de presentación contiene los componentes que implementan y muestran la interfaz de usuario y manejan la interacción con el usuario. Podemos distinguir dos tipos de componentes dentro de esta capa:

• Controles de la UI. Son los controles visuales que despliegan información y reciben las acciones y datos del usuario.

• Componentes de Lógica de Presentación. Son los componentes no visuales que definen el comportamiento lógico de la UI e interactúan con la capa subyacente (Servicio o Negocio).

2.2 Capa de Servicios

Cuando una aplicación debe proveer servicios a otras aplicaciones, así como a sus propios clientes, es una práctica común la implementación de una capa de servicios que exponga la funcionalidad de negocio de la aplicación.

En este escenario, los usuarios pueden acceder a la aplicación a través de la capa de presentación que se comunica directamente con la capa de negocio, mientras que los clientes externos y otros sistemas pueden acceder a la aplicación y hacer uso de cierta parte de su funcionalidad a través de la capa de servicios. En algunos casos, la presentación también podría comunicarse a través de la capa de servicios, en especial si la capa de presentación y la capa de negocio se distribuyen en diferentes máquinas.

La capa de servicios está formada por los siguientes componentes:

• Interfaces de Servicios. Los servicios exponen interfaces hacia donde llegan todos los mensajes externos.

• Tipos de Mensajes. Cuando se intercambian datos a través de la capa de servicios, las estructuras de datos deben ser adaptadas mediante estructuras de mensajes que soporten la comunicación con clientes externos.

2.3 Capa de Negocio

La capa de negocio (también llamada capa lógica, capa de lógica de negocio o capa de dominio) implementa la funcionalidad principal del sistema y encapsula la lógica de negocio.

La capa de negocio usualmente incluye los siguientes componentes:

• Fachada de la Aplicación. Este componente provee una interfaz simplificada a la aplicación, usualmente componiendo múltiples operaciones de negocio en una única operación. Sirve para reducir dependencias con las capas superiores.

• Entidades de Negocio. Son los objetos que representan las entidades que componen el negocio en el mundo real (ej. Clientes, Cuentas, etc.).

• Componentes de Workflow. Definen y coordinan procesos de negocio de múltiples pasos, utilizando las entidades de negocio. Se suelen implementar mediante BPMS (Business Process Management Systems).

2.4 Capa de Persistencia

La capa de persistencia (también llamada capa de datos o capa de acceso a datos) provee acceso a los datos almacenados dentro del sistema y, en algunos casos, a datos ajenos al sistema a los que se accede mediante servicios externos. Los datos generalmente se encuentran en bases de datos relaciones.

La capa de persistencia usualmente incluye los siguientes componentes:

• Componentes de Acceso a Datos. Son los componentes que abstraen la lógica necesaria para acceder a las fuentes de datos subyacentes. Generalmente se utilizan frameworks de tipo ORM (Object/Relational Mapping) que permiten hacer la conversión entre objetos de la capa de negocio y tablas de una base de datos relacional.

• Agentes de Servicios. Son los componentes que abstraen la lógica necesaria para acceder a los servicios de datos externos. Pueden proveer funcionalidades de cache, transformación de datos, etc.

3 Bibliografía

M.P.&.P. Team, Microsoft Application Architecture Guide, Microsoft Press, 2009.

Integración de Sistemas

1 Introducción

En general, la integración es un proceso que siguen las organizaciones cuando existe la necesidad de que más de un sistema trabaje junto en forma orquestada. En un típico proceso de integración, un gran esfuerzo se emplea en adaptar las interfaces de los sistemas, comprender y diseñar los puntos de interacción entre los componentes de los sistemas y agregar todas las piezas intermedias que hacen que el sistema integrado funcione.

Las organizaciones comprenden cientos de aplicaciones que pueden ser desarrolladas a medida, adquiridas a terceros, sistemas legados, trabajando sobre diferentes sistemas operativos, etc. No es inusual encontrar empresas que tienen 30 sitios web diferentes, tres instancias de ERP e incontables soluciones departamentales.

¿Cómo se llega a esta situación tan caótica? Primero, el desarrollo de aplicaciones de negocio es complejo. La creación de una única gran aplicación que maneje todo el negocio es prácticamente imposible. Los vendedores de ERP han tenido éxito en crear aplicaciones enormes que, sin embargo, solo realizan una parte de las funciones requeridas en una empresa típica. Esto ha llevado a que los ERP sean uno de los puntos de integración más populares.

Segundo, distribuir las funciones de negocio a través de múltiples aplicaciones le ofrece al negocio la flexibilidad de elegir lo mejor de cada una: el "mejor" paquete de contabilidad, el "mejor" software de CRM, etc. Usualmente, las organizaciones no quieren un único sistema que lo haga todo. Los vendedores han aprendido de esto y ofrecen aplicaciones enfocadas a una determinada función básica. Sin embargo, la urgencia de añadir nuevas funcionalidades y la dificultad en establecer la separación funcional de los sistemas ha llevado a que los sistemas agreguen funcionalidades extras que van más allá de su función específica. Por ejemplo, sistemas de facturación con capacidades de contabilidad o sistemas de contabilidad con capacidades de facturación.

Las fronteras de los sistemas deben ser transparentes para clientes y usuarios. Por ejemplo, cuando un cliente realiza una nueva orden de compra, la empresa debería validar su ID, verificar la reputación, chequear el inventario, completar la orden, coordinar el envío, computar los impuestos, imprimir la factura, etc. Todo esto puede fácilmente implicar a unos cinco sistemas o más, pero desde el punto de vista del cliente, es una sola transacción.

Para soportar estos procesos de negocio, las aplicaciones necesitan ser integradas de manera eficiente, confiable y segura.

Desafortunadamente, la integración de sistemas empresariales no es una tarea fácil. Los vendedores de software ofrecen suites de Enterprise Application Integration (EAI) que proveen integración multiplataforma y multilenguaje, así como la posibilidad de interactuar con paquetes de aplicaciones de negocio populares. Sin embargo, estos EAI solucionan una fracción de los desafíos de integración existentes en las organizaciones.

Para empezar, una integración eficaz a nivel de software no se puede alcanzar si no se logra una comunicación fluida entre los diferentes departamentos de la organización y sus unidades informáticas. Las aplicaciones dejan de ser la propiedad de un grupo y pasan a ser una parte de un flujo de aplicaciones y servicios integrados.

Una restricción importante que enfrentan los desarrolladores de soluciones de integración es el poco control sobre las aplicaciones que se van a integrar, tales como sistemas legados y paquetes COTS. Esto lleva a tener que compensar las deficiencias que se encuentran en las aplicaciones utilizando el software de integración, cuando sería más lógico realizar correcciones dentro de las aplicaciones mismas.

A pesar de la gran oferta de soluciones de integración solo algunos pocos estándares se han impuesto en este dominio. El advenimiento del XML, XSL y los Web Services han marcado el avance más significativo en la integración basada en estándares de los últimos años. XML es considerada la "lingua franca" de la integración de sistemas. Sin embargo, la existencia de una presentación común de los datos, utilizando XML, no implica una semántica común en los mismos. El mismo concepto puede representar cosas distintas en cada aplicación.

2 Arquitecturas de Integración

2.1 Portales de Información

Los usuarios muchas veces deben acceder a más de un sistema para obtener respuesta a una pregunta o realizar una función del negocio. Por ejemplo, para verificar el estado de una orden, un representante de servicio al cliente podría tener que acceder al sistema de gestión de órdenes en el mainframe, así como loguearse en el sistema que maneja órdenes realizadas en la Web. Los portales de información agregan información de varias fuentes y muestran al usuario una pantalla única para que no deba acceder a más de un sistema. La pantalla podría

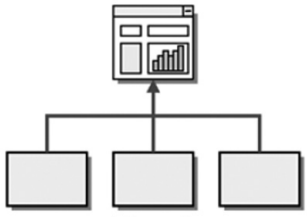


Figura 1

estar dividida entre las aplicaciones que se consultan sin ofrecer mayor integración o las selecciones que se realizan en un sistema podría actualizar los datos que se recuperan de otro.

2.2 Replicación de Datos

Muchos sistemas empresariales necesitan acceder a los mismos datos. Por ejemplo, cuando un cliente llama para cambiar su dirección, todos los sistemas que utilizan la dirección

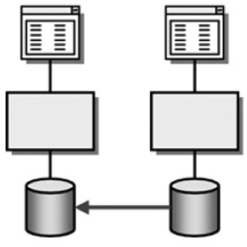


Figura 2

del cliente (CRM, Contabilidad, Envíos, Facturación, etc.) deben actualizar los datos en sus propias bases de datos. La replicación de datos puede realizarse, por ejemplo, mediante las funcionalidades brindadas por el DBMS o utilizando Middleware de Mensajería (MOM).

2.3 Función de Negocio Compartida (Servicios)

Así como las aplicaciones empresariales suelen almacenar datos redundantes, también suelen implementar funcionalidades redundantes, por lo que esas funciones comunes podrían factorizarse e implementarse una sola vez para que la utilicen todas las aplicaciones. Esta estrategia podría utilizarse en el ejemplo anterior, implementando un servicio compartida GetDireccion en lugar de que cada aplicación almacene la dirección del cliente.

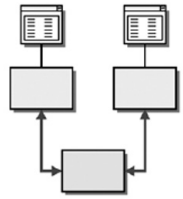


Figura 3

2.4 Arquitectura Orientada a Servicios (SOA)

Una vez que la cantidad de servicios compartidos empieza a crecer, manejar esos servicios se vuelve una función crítica. Primero, las aplicaciones necesitan algún tipo de directorio de servicios que centralice los servicios disponibles. Segundo, cada servicio necesita describir su interfaz de manera que las aplicaciones pueden negociar con un servicio un contrato de comunicación. Las SOAs no solo permiten integrar aplicaciones sino también desarrollar aplicaciones distribuidas. Una nueva aplicación puede ser desarrollada utilizando servicios remotos existentes provistos por otras aplicaciones.

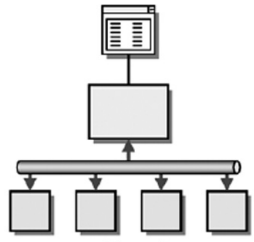


Figura 4

2.5 Procesos de Negocios Distribuidos

Uno de los motivos que hacen más necesaria la integración es que una única transacción de negocio se extiende por varios sistemas. En la mayoría de los casos, todas las funciones necesarias están implementadas en las aplicaciones, pero se necesita un componente que se

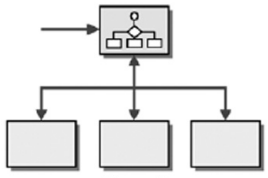


Figura 5

encargue que la coordinación de las mismas. Para esto se agrega un componente que coordine u orqueste a las aplicaciones. Esta solución podría tomar la forma de una SOA, pero no necesariamente.

2.6 Integración Business-to- Business (B2B)

Hasta aquí hemos considerado interacciones entre aplicaciones y funciones del negocio dentro de una misma empresa. En muchos casos, algunas funciones de negocio deben estar disponibles hacia afuera para ser usadas por proveedores o socios de negocios. Por ejemplo, una empresa podría utilizar un servicio externo para calcular los aportes a la seguridad social que deben realizar por sus empleados o una vendedora podría consultar a un servicio de un proveedor para averiguar la fecha de entrega de un producto. Muchas de las estrategias vistas anteriormente pueden utilizarse para integración B2B, sin embargo, la comunicación a través de una red como Internet impone ciertas restricciones adicionales en cuanto a protocolos, eficiencia y seguridad.

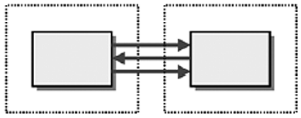


Figura 6

3 Patrones de Integración

3.1 Transferencia de Archivos

Los archivos son un mecanismo universal de almacenamiento de datos, incorporados a cualquier sistema operativo y accesibles para cualquier lenguaje de programación. La forma más sencilla de integrar aplicaciones es, entonces, mediante archivos.

En este enfoque, cada aplicación produce archivos que contienen información que otras aplicaciones deben consumir. Los integradores tienen la responsabilidad de transformar los archivos a diferentes formatos. Los archivos deben ser producidos a intervalos regulares de acuerdo a la dinámica del negocio.

Una decisión importante con los archivos es qué formato utilizar. Es importante contar con formatos de archivos estándar. Por ejemplo, los mainframes utilizan formatos soportados por COBOL, los sistemas UNIX utilizan archivos de texto plano, etc. La tendencia actual es usar archivos XML.

Otro tema importante es cuándo producir y consumir los archivos. Generalmente se posee un ciclo de negocio que realiza la generación con una frecuencia determinada: todas las noches, una vez por semana, etc.

Algunos de los inconvenientes que tiene este enfoque son:

- Cuándo y quién se encarga de borrar los archivos viejos.
- Escritura y lectura concurrente.
- Acceso a discos compartidos.
- Actualización de datos tardía.

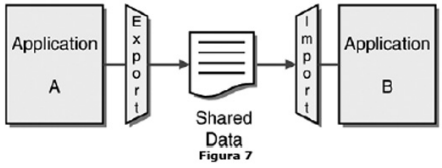


Figura 7

3.2 Base de Datos Compartida

Muchos de los problemas que se presentan con la transferencia de archivos se solucionan integrando las aplicaciones a través de una base de datos. Se define el esquema de la base de datos de manera tal que sirva a todas las aplicaciones. De esta forma, no se incurre en problemas de "disonancia semántica" desde el principio, por lo que no hay que lidiar con transformaciones complejas después.

Algunos de los inconvenientes que se presentan son:

- Diseño del esquema unificado que sirva a todas las aplicaciones.
- Los paquetes de aplicaciones muchas veces solo soportan su propio esquema de bases de datos.
- Problemas de performance con una base de datos única y clientes remotos.

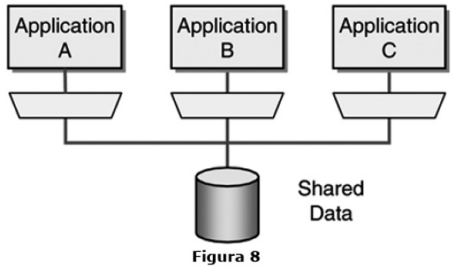


Figura 8

3.3 Llamada a Procedimientos Remotos (RPC)

La llamada a procedimientos remotos consiste en tener aplicaciones que se comportan como grandes componentes que encapsulan los datos y que ofrecen una interfaz para interactuar con otras aplicaciones.

La llamada a procedimientos remotos permite implementar el principio de encapsulamiento de los datos. Si una aplicación necesita datos que los posee otra aplicación, consulta a la otra aplicación directamente invocando un procedimiento. De igual forma, si una aplicación necesita modificar los datos de otra aplicación, invoca un procedimiento de modificación de datos.

Existen muchas tecnologías que permiten implementar un RPC, tales como .NET Remoting, Java RMI, COM, CORBA, etc. Estas tecnologías poseen diferencias en cuanto a grado de interoperabilidad multiplataforma, soporte de transacciones, facilidad de uso, etc. Para garantizar un máximo de interoperabilidad, la opción más recomendable son los Web Services XML, basados en estándares como SOAP, que permiten atravesar los firewalls por usar típicamente el protocolo HTTP como transporte.

Algunos de los inconvenientes que se presentan son:

• Secuenciamiento entre aplicaciones hace difícil cambiar aplicaciones en forma independiente (alto acoplamiento).

• Sincronismo obliga a que todas las aplicaciones estén disponibles.

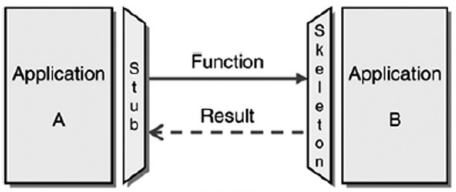


Figura 9

3.4 Mensajes

Muchos de los problemas que se presentan con los enfoques anteriores se solucionan utilizando intercambio de mensajes. Los mensajes permiten enviar paquetes de datos en forma confiable y asincrónica utilizando formatos personalizables. El envío de un mensaje es una solución a los problemas típicos de los sistemas distribuidos, ya que no se requiere que el emisor y el receptor estén disponibles al mismo tiempo.

Los sistemas de mensajería ofrecen un mayor desacoplamiento que cuando se transfieren archivos. Los mensajes pueden ser transformados "en tránsito" sin que el emisor ni el receptor se enteren de las transformaciones intermedias.

El enviar pequeños mensajes en forma frecuente permite un nivel más fuerte de integración entre las aplicaciones que cuando sólo se comparten datos. Si bien la colaboración no es tan rápida como en el caso de RPC, el emisor no necesita parar su procesamiento esperando la respuesta como en el otro caso.

Si bien se considera que los sistemas de mensajería son generalmente la mejor forma de integración de sistemas, no están exentos de problemas, por ejemplo:

• Existe menos experiencia en desarrollo de sistemas asincrónicos que sincrónicos.

• El testing y el debugging de aplicaciones es más lento y complejo.

¿Cómo se transfieren los paquetes de datos?

El emisor envía datos al receptor enviando un mensaje a través de un canal de mensajes que conecta el emisor con el receptor.

¿Cómo se sabe a dónde enviar los datos?

Si el emisor no conoce la dirección exacta del receptor puede enviar los datos a un router de mensajes que redirige los datos al receptor correspondiente.

¿Cómo se sabe qué formatos de datos usar?

Si emisor y receptor no se ponen de acuerdo en un formato de datos, el emisor puede enviar los datos a un traductor de mensajes que convertirá los datos al formato que acepta el receptor.

¿Cómo se conecta una aplicación a un sistema de mensajería?

Las aplicaciones que desean integrarse mediante mensajes deben implementar un endpoint de mensajes.

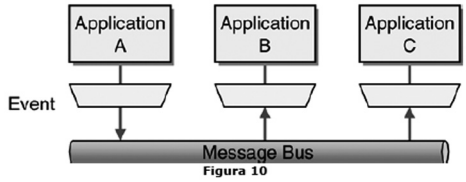


Figura 10

4 Bibliografía

G. Hohpe and B. Woolf, Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions, Addison-Wesley Professional, 2003.

Introducción a los Sistemas de Información

¿Qué es un Sistema de Información?

Un Sistema de Información (SI) es un conjunto organizado que integra personas, datos, actividades y recursos para registrar datos y generar nueva información.

Podemos hacer una diferenciación entre los que son datos e información, aunque en algunos casos se utilizan como sinónimos.

Los datos son las descripciones elementales de las cosas, eventos, actividades y transacciones que son registradas, clasificadas y almacenadas pero que no son organizadas para transmitir ningún significado específico. Los datos pueden ser números, palabras, imágenes, sonidos, etc. Ejemplos de datos pueden ser la nota final de un estudiante en una asignatura o el número de horas que son trabajadas por un empleado en un mes.

La información es la organización de los datos que permite que tengan un cierto valor y significado para el receptor. En algunos enfoques, los datos son vistos como insumos de un proceso de manufactura que termina con un producto, llamado Producto de Información.

El conocimiento consiste en datos o información que ha sido procesada para transmitir comprensión, experiencia, aprendizaje y experticia.

Tipos de Sistemas de Información

Algunos sistemas de información soportan partes de una organización, otros soportan organizaciones enteras y otros conjuntos de organizaciones.

Cada departamento o área funcional dentro de una organización tiene su propio sistema de información (ej. "el sistema de información de recursos humanos"), y una colección de aplicaciones. Las áreas funcionales típicas son Contabilidad, Finanzas, Producción/Operaciones (POM), Marketing y Recursos Humanos.

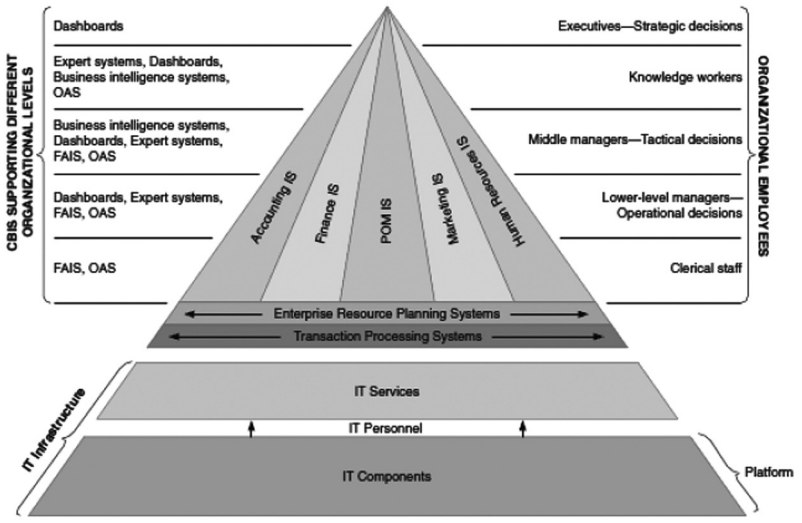


Figura 1

Sin embargo, existen dos sistemas de información que soportan a la organización entera y que están en la base de la infraestructura informática. Estos son el Sistema de Procesamiento de Transacciones (TPS) y el Sistema de Planeamiento de Recursos Empresariales (ERP). Estos sistemas se encuentran en la base la pirámide, como puede observarse en la Figura 1.

En la Figura 2 puede verse en qué momento fueron surgiendo los diferentes tipos de SI que vamos a analizar a continuación.

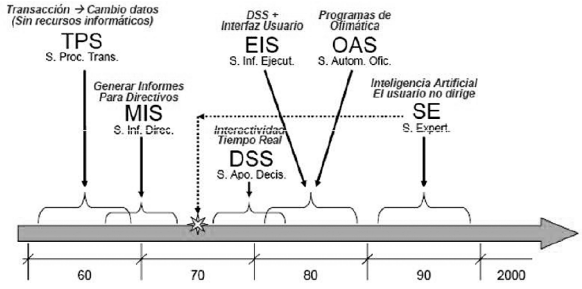


Figura 2

2.1 TPS y ERP

El Sistema de Procesamiento de Transacciones (Transaction Processing System, TPS) se encarga de monitorear, recolectar, almacenar y procesar los datos de las transacciones de negocio básicas de una organización. Millones de transacciones ocurren en la mayoría de las organizaciones cada día.

Una transacción es cualquier evento del negocio que genera datos relevantes para ser capturados y almacenados en una base de datos. En los sistemas empresariales modernos estas transacciones son las entradas de los sistemas de información departamentales o de áreas funcionales (FAIS), los sistemas de soporte a la toma de decisiones (DSS), los sistemas de gestión de relaciones con los clientes (CRM), etc.

Los TPS deben manejar altos volúmenes de datos así como variaciones radicales en el volumen de transacciones, por ejemplo, en las horas pico, en forma eficiente, evitando errores y demoras y registrando los datos en forma precisa y segura.

Sin importar el tipo de datos que procese un TPS, el proceso que sigue es bastante estándar. Primero, los datos son recolectados mediante personas o sensores y son ingresados al sistema mediante un dispositivo de entrada. Se trata que este proceso sea automatizado al máximo. A continuación, el sistema procesa los datos en alguno de los siguientes modos: procesamiento por lotes (batch) o procesamiento en línea (online). En el primer caso, las transacciones se agrupan en lotes y se programa el sistema para que los procese periódicamente, por ejemplo, todas las noches. En el segundo caso, las transacciones se procesan a medida que ocurren. Esto se conoce como Online Transaction Processing (OLTP). Por ejemplo, cuando se compra un ítem de un producto en una tienda, el sistema registra la venta, reduce en una unidad el stock de ese producto, incrementa la suma de efectivo de la tienda con el precio del producto e incrementa en uno la cantidad de ventas de ese producto. En la Figura 3 puede verse como un TPS de una tienda maneja los datos.

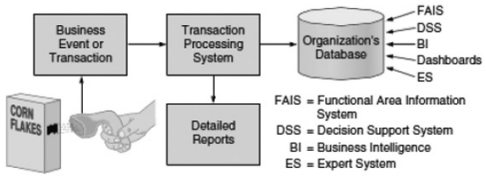


Figura 3

Un Sistema de Planeamiento de Recursos Empresariales (Enterprise Resource Planning, ERP) se encarga de integrar el planeamiento, gerenciamiento y uso de todos los recursos de una organización. Los principales objetivos de un ERP son el de integrar estrechamente las áreas funcionales de una organización y el de permitir que la información fluya sin esfuerzo a través de las áreas funcionales. La integración permite que los cambios en un área funcional sean inmediatamente reflejados en las otras áreas. Los ERP fueron una importante innovación dado que los SI de cada área funcional eran desarrollados normalmente como sistemas independientes y no se comunicaban con los otros. Los ERP son casi siempre TPS, pero no todos los TPS son ERP.

Los ERP proveen la información necesaria para controlar los procesos de negocio de la organización. Un proceso de negocio es un conjunto de pasos relacionados o procedimientos diseñados para producir un resultado relevante dentro del negocio. Los procesos de negocio pueden estar restringidos a una sola área funcional o involucrar varias áreas. El software de ERP incluye un conjunto de módulos independientes para distintas áreas funcionales que se vinculan mediante una base de datos común.

Aunque algunas compañías desarrollan su propio software de ERP, esto es demasiado caro y complejo para la mayoría por lo que, generalmente, se compra como COTS (commercial off-the-shelf) a empresas como SAP y Oracle. Estas empresas incorporan a sus sistemas las "mejores prácticas" (best practices) que han desarrollado para cada proceso de negocio predefinido. Eso puede ser un problema para las organizaciones, ya que podrían tener que cambiar sus procesos de negocio para adaptarse los procesos de negocio predefinidos del software. También existe el inconveniente de que estos productos se venden con todos los módulos como una unidad, aunque la organización solo precise alguno de ellos.

2.2 OAS, FAIS, BI, Sistemas Expertos, Tableros

Dentro de cada área funcional tenemos los siguiente tipos de SI: Sistemas de Automatización de Oficinas (OAS), Sistemas de Información de Área Funcional (FAIS), Sistemas de Inteligencia de Negocio (BI), Sistemas Expertos y Tableros; los que son utilizando en diferentes niveles jerárquicos de la organización.

Un Sistema de Automatización de Oficina (Office Automation System, OAS) es un sistema que ayuda a desarrollar el trabajo diario a nivel de administrativos y gerentes de los primeros niveles. Generalmente involucran generación de documentos, agendas, comunicación y trabajo en grupo.

Un Sistema de Información de Área Funcional (Functional Area Information System, FAIS), también llamado Sistema de Información Gerencial (Management Information System, MIS), es un sistema que permite resumir datos y preparar reportes, dentro de un área

funcional. Generalmente es utilizado por gerentes de niveles medios. Los generadores de reportes (RPG) son un tipo clásico de FAIS.

Un Sistema de Inteligencia de Negocio (Busines Intelligence, BI) es un sistema que permite la consolidación, análisis y acceso a grandes cantidades de datos para generar información de valor para la toma de decisiones estratégicas del negocio. Es utilizado en niveles gerenciales medios y altos. Bajo el nombre colectivo de BI se agrupan diversos tipos de sistemas: los que proveen herramientas de análisis de datos (data mining, OLAP, etc.), los que proveen información fácilmente accesible en un formato estructurado (tableros) y los que pueden tomar decisiones por sí mismos (sistemas expertos). La arquitectura de un típico sistema de BI puede verse en la Figura 4.

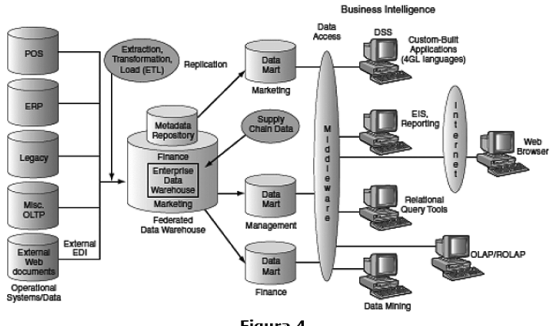
Dentro de los sistemas de BI encontramos los Sistemas de Apoyo a la Toma de Decisiones (Decision Support Systems, DSS), que se utilizan para asistir a gerentes y analistas en la tarea de interpretar la información referente a las operaciones de la organización y extraer conclusiones que sirvan como base para la toma de decisiones futuras. Los DSS surgieron a partir de los MIS y han sufrido varias ramificaciones desde sus comienzos en la década del '60. Podemos establecer una división entre los DSS basados en modelos, que pertenecen principalmente al área de la Investigación Operativa, y los DSS basados en datos, que son los que pertenecen al área de la BI.

A partir de la década del '90 y hasta la actualidad, los DSS basados en datos se construyen alrededor de bases de datos especializadas que almacenan únicamente la información relevante para la toma de decisiones. Este tipo de base de datos se conoce como data warehouse y los DSS cuya arquitectura está centrada en la explotación de un data warehouse se denomina Sistema de Data Warehousing.

Los datos de un data warehouse se obtienen principalmente como agrupamientos y totalizaciones de los datos que se encuentran en las base de datos operacionales a través de procesos de extracción, transformación y carga (ETL). Un concepto muy relacionado es el de data mart, que denomina generalmente a subconjuntos del data warehouse, aunque puede también integrar un número de fuentes heterogéneas e inclusive ser más grande, en volumen de datos, que el propio data warehouse central, aunque comúnmente el volumen es menor, por lo que tienen mejores tiempos de respuesta, costos menores, etc. Los data marts son diseñados para satisfacer las necesidades específicas de grupos comunes de usuarios (divisiones geográficas, divisiones organizacionales, etc.), y su control es local y no centralizado como en el data warehouse.

Los sistemas de datawarehousing permiten realizar lo que se conoce como Online Analytical Processing (OLAP). El OLAP consiste en realizar Análisis Dimensional en forma rápida, dinámica y flexible de la información multidimensional. Por ejemplo, en una empresa pesquera, interesa saber cuál fue la cantidad capturada de pescado de la especie en la zona marítima Z, en el mes M. Lo que estamos haciendo es un análisis de una medida de interés (la cantidad capturada) desde tres perspectivas: la especie, la ubicación geográfica y el tiempo. Cada una de estas perspectivas se denomina dimensión, y el análisis dimensional permite contestar este tipo de consultas en forma inmediata. El sistema debe proveer una visión conceptual multidimensional de los datos, incluyendo soporte para jerarquías, ya que ésta es la forma más intuitiva y natural de interpretar información. A través de una simple navegación los usuarios finales pueden analizar diferentes escenarios, independientemente del tamaño, complejidad y fuente de los datos corporativos. La información es conceptualmente organi-

zada en cubos que almacenan las medidas (dato cuantitativo). El otro concepto importante es la dimensión, que es una jerarquía donde se organizan los datos en diferentes niveles de detalle. Para brindar capacidades OLAP al sistema se pueden utilizar tecnologías ROLAP



(Relational OLAP), MOLAP (Multidimensional OLAP) o HOLAP (Hybrid OLAP).

Un Tablero (Dashboard), también llamado Sistemas de Información Executivo (Executive Information System, EIS) provee un rápido acceso a información relevante para ejecutivos y gerentes de nivel alto a través de interfaces basadas en gráficos. Los tableros permiten realizar operaciones de drill-down sobre los datos agregados, muestran indicadores de performance clave (KPI) del negocio, factores éxito críticos (CSF), análisis de tendencias, etc. En la Figura 5 puede verse una interfaz de ejemplo de un tablero.

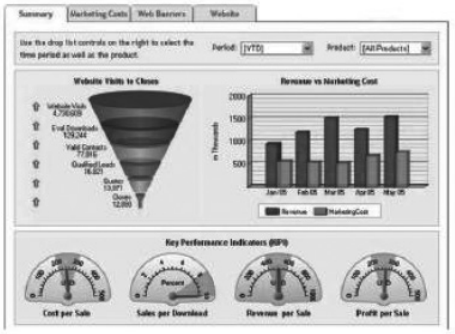


Figura 5

Un Sistema Experto (Expert System, ES) es un sistema informático que trata de emular a los expertos humanos aplicando experticia en un dominio específico. Los sistemas expertos no sólo pueden apoyar la toma de decisiones sino que pueden tomar las decisiones por sí mismos. Los sistemas expertos son el caso más utilizado de Inteligencia Artificial en productos de software comerciales. Los ES suelen estar compuestos de una base de conocimiento, en donde residen los hechos y las reglas, un motor de inferencia que razona a partir de esos hechos y reglas, y una interfaz de usuario que, generalmente, se maneja mediante preguntas y respuestas. Los ES suelen tener arquitecturas de estilo pizarrón (blackboard).

2.3 GIS, CRM

En esta sección vamos a comentar algunos otros tipos de SI que han comenzado a incorporarse masivamente a las organizaciones en los últimos años: los sistemas de información geográficos y los sistemas de gestión de relaciones con los clientes

Un Sistema de Información Geográfica (Geographic Information System, GIS) trabaja con información que ha sido georreferenciada, es decir, que tiene coordenadas sobre la superficie de la Tierra, como ser países, ciudades, calles, ríos, puntos de interés, etc. El propósito de estos sistemas es permitir que los usuarios visualicen, comprendan, interpreten y cuestionen la información geográfica en su forma más natural y amigable, que es mediante su despliegue en un mapa (aunque estos sistemas también soportan otro tipo de visualizaciones, como ser globos 3D, gráficas, reportes, etc.). Una gran cantidad de información sobre las relaciones espaciales de la información geográfica puede ser representada en un mapa en forma compacta, conveniente e intuitiva. Los GIS han dejado de estar confinados a los especialistas en información geográfica y han empezado a utilizarse en conjunto con otros SI empresariales y de uso masivo.

La Gestión de Relaciones con los Clientes (Customer Relashionship Management, CRM), es la gestión que realiza la organización para obtener y retener clientes. El CRM reconoce que los clientes son el núcleo del negocio y que el éxito del mismo depende de manejar efectivamente las relaciones con los mismos. La CRM se basa en una relación uno-a-uno entre empresa y cliente, es decir, en el trato diferencial entre distintos clientes. La importancia de contar con una buena CRM radica en que, económicamente, es mucho más conveniente mantener a cliente actual que conseguir un nuevo cliente.

En el pasado, los datos de los clientes estaban localizados en sistemas aislados en varias áreas funcionales. Además, el comercio electrónico generaba grandes cantidades de datos de clientes que no se integraban en estos sistemas. Los sistemas de CRM tienen por objetivo integrar datos de clientes desde varias fuentes organizacionales, analizar estos datos y proveer una vista unificada de los clientes para toda la organización. Los CRM tienen aplicación fundamentalmente en las áreas de ventas, marketing y servicio al cliente.

Para finalizar, mencionaremos que existe otro tipo de SI que son los SI interorganizacionales que involucran el manejo de los flujos de información entre más de una organización.

Distribución, Autonomía, Heterogeneidad

Diferentes criterios pueden ser adoptados para clasificar los SI y sus correspondientes arquitecturas. Analizaremos una clasificación en base a los criterios de distribución, autonomía y heterogeneidad.

La distribución evalúa la posibilidad de distribuir los datos y las aplicaciones en una red de computadoras. Por simplicidad, se adopta un dominio booleano para esta dimensión (sí, no). La heterogeneidad considera todos los tipos de diversidades tecnológicas y semánticas en el modelado y representación física de los datos que se dan en los componentes del sistema como ser bases de datos, lenguajes de programación, sistemas operativos, middleware, etc. Por simplicidad, se adopta un dominio booleano para esta dimensión. La autonomía tiene que ver con el grado de jerarquía y las reglas de coordinación, estableciendo derechos y deberes, definidos en la organización que utiliza el SI. Los dos extremos son: (i) completamente jerárquico, donde un sujeto decide por todos y (ii) completamente anárquico, donde cada organización es autónoma y toma las decisiones en forma libre. En este caso, se adopta un dominio (no, semi, total) para la autonomía.

Entre los múltiples tipos de SI posibles se analizan cinco ejemplos típicos, que son los SI monolíticos, distribuidos, datawarehousing, cooperativos y peer-to-peer. Una representación gráfica de esta clasificación puede verse en la Figura 6.

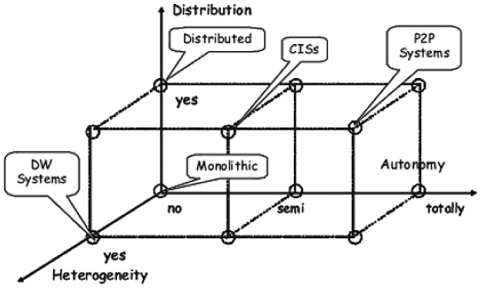


Figura 6

En los SI monolíticos la presentación, la lógica de aplicación y el manejo de datos se encuentran en un solo nodo. Estos sistemas son extremadamente rígidos pero aún se utilizan, ya que tienen costos reducidos por ser homogéneos y tener una administración centralizada.

Los sistemas de data warehousing, que describimos anteriormente, se caracterizan por poseer una base datos centralizada obtenida a partir de diferentes bases de datos fuente.

Los SI distribuidos permiten la distribución de recursos, datos y aplicaciones en una red. Estos sistemas utilizan arquitecturas de N-niveles. Los niveles no son autónomos y generalmente no existen demasiadas heterogeneidades.

Los SI cooperativos (Cooperative Information System, CIS) pueden ser definidos como SI de gran escala que interconectan varios sistemas de organizaciones autónomas que comparten objetivos en común. Estos sistemas pueden tener una arquitectura orientada a servicios (SOA).

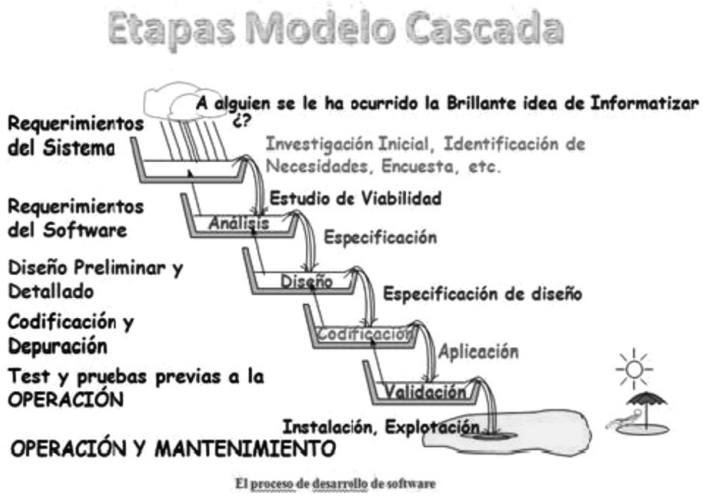
Los sistemas peer-to-peer (P2P) eliminan la distinción entre clientes y servidores, típica de los sistemas distribuidos a favor de un red de aplicaciones autónomas y heterogéneas, que no poseen un coordinación central ni comparten una base de datos.

Proceso de desarrollo de software y su documentación

¿Qué es?

- Es un enfoque para organizar la construcción, instalación y mantenimiento del software.
- Es una "receta" de cómo atacar el desarrollo de un sistema.
- Ejs. Modelo Cascada, Proceso Unificado (UP), etc, etc.

Modelo Cascada (Waterfall)



El proceso de desarrollo de software

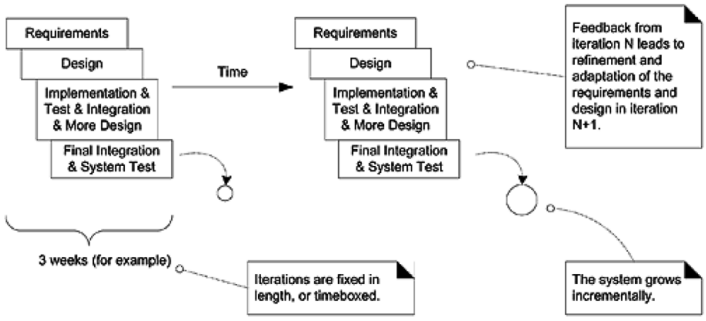
- En el modelo cascada cada actividad (análisis, diseño, etc.) termina antes que comience la siguiente.
- Es un modelo de proceso secuencial.
- Existen variaciones: prototipaciones intermedias, saltos a la actividad previa, etc.
- "Mentalidad" cascada:
  - "Documentemos todos los requerimientos antes de hacer el diseño."
  - "Terminemos el diseño antes de comenzar a programar."

- Problema del modelo cascada:
  - Inestabilidad de los requerimientos.
  - No tiene en cuenta el feedback.

¿Qué ejemplos se le ocurren de cómo una actividad puede proporcinar feedback para otra?

Procesos Iterativos e Incrementales

- El software moderno se desarrolla casi totalmente siguiendo procesos iterativos e incrementales
- El proceso se desarrolla en etapas llamadas iteraciones, de largo fijo (2 a 4 semanas).
- En cada iteración el producto se refina y agranda.
- "Abrazar el cambio" en vez de "Luchar contra el cambio".



Proceso Unificado (UP)

- Es un proceso de desarrollo de tipo iterativo e incremental.
- Muy populares dentro de sistemas orientados a objetos.
- Guiado por casos de uso.
- Centrado en la arquitectura.
- Para planificar las iteraciones se tiene en cuenta:
- Riesgos: se implementa primero lo que plantea más riesgos (permite validar la arquitectura).
- Clientes: se implementa primero lo que es más importante para el cliente.
- El Rational Unified Process (RUP) es un refinamiento detallado del UP.
- El Agile Unified Process (AUP) es una adaptación del UP con metodologías ágiles.

Fases del UP

- Incepción: visión aproximada, conocimiento del negocio, alcance, estimaciones iniciales.
- Elaboración: visión refinada, implementación iterativa de la arquitectura base, resolución de riesgos altos, identificación de la mayoría de los requerimientos, alcance y estimaciones más realistas.
- Construcción: implementación iterativa del resto de las funcionalidades y preparación de la implantación
- Transición: beta testing, implantación.

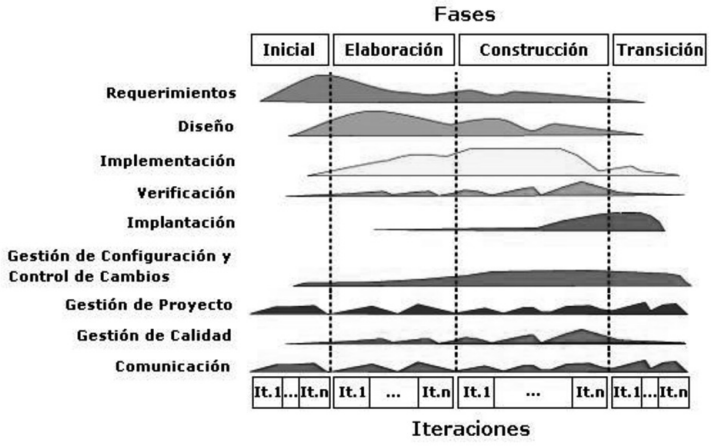
Disciplinas del UP:

- Análisis del Negocio (Business Modelling).
- Requerimientos.
- Diseño.
- Implementación.
- Verificación (Testing).
- Implantación (Deployment).
- Gestión de Configuración y Control de Cambios.
- Gestión de Proyecto.
- Gestión de Calidad.
- Comunicación.

Cada disciplina posee:

- Roles.
- Actividades.
- Entregables.

¿Qué ejemplos se le ocurren roles, actividades y entregables dentro de cada disciplina?



Documentación con UML

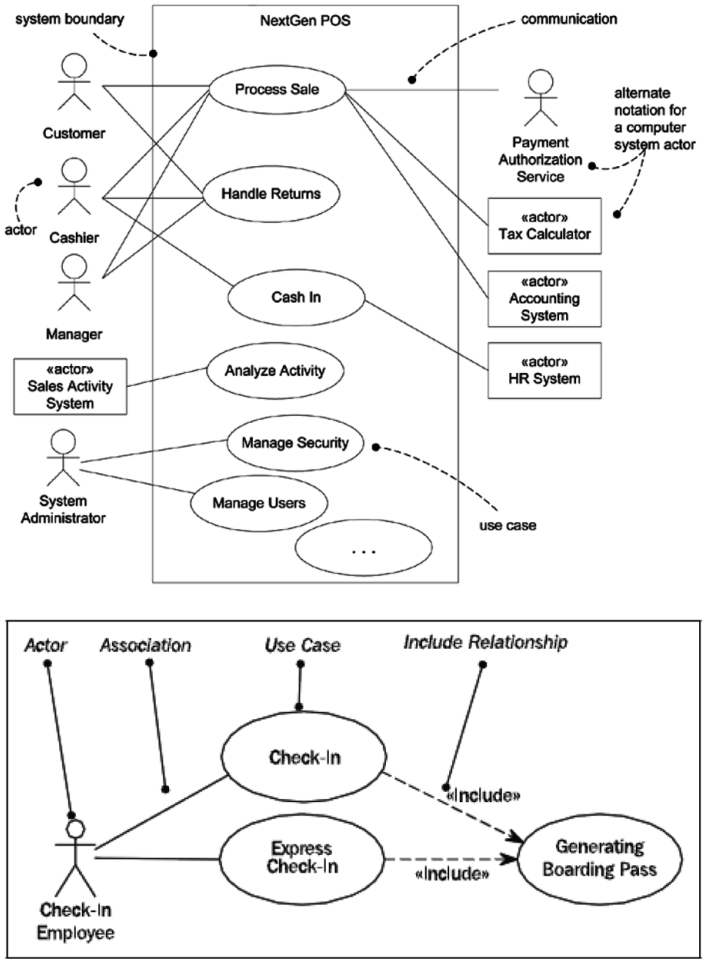
Casos de Uso

• Los casos de uso son un forma estructurada de escribir requerimientos funcionales narrando las interacciones entre los actores y el sistema.

• Los casos de uso son texto, no diagramas. El diagrama de casos de uso de UML es una herramienta secundaria para documentar casos de uso.

• Cada caso de uso posee varias instancias llamadas flujos o escenarios (ej. compra exitosa, compra no exitosa por tarjeta sin saldo).

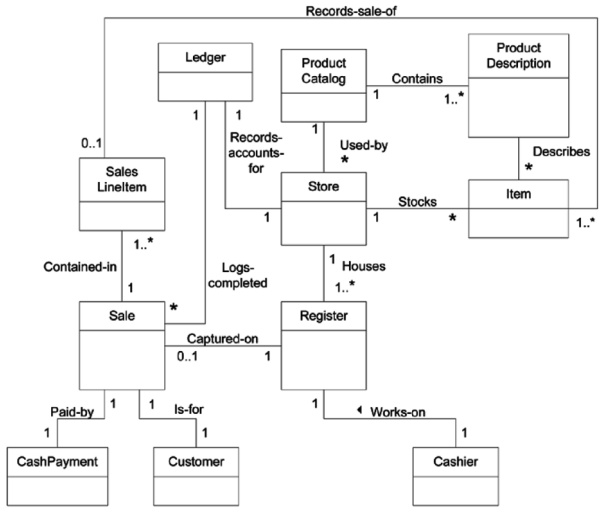
Diagrama de Casos de Uso UML



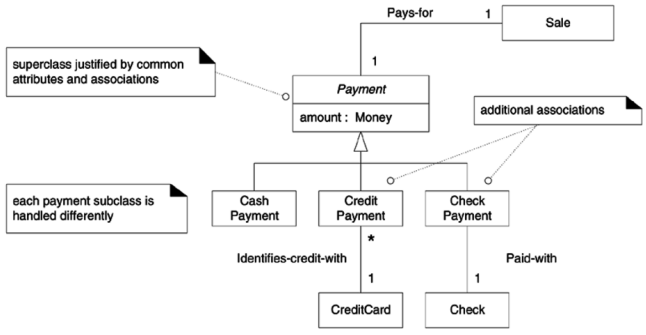
Modelo de Dominio

• El modelo de dominio (o modelo conceptual) es el principal artefacto del Análisis de Dominio Orientado a Objetos.
• Es una representación visual de los objetos del dominio y sus relaciones.
• No es un modelo de datos (no implica persistencia).
• No son clases implementadas (no hay métodos).
• No representan componentes de software.

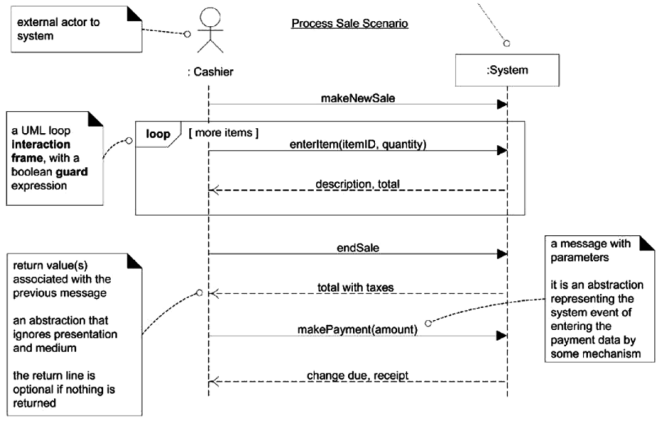
Diagrama de Clases



Generalizaciones



Diagramas de Secuencia del Sistema (SSD)



Diagramas de Secuencia y Comunicación (equivalentes, para Diseño)

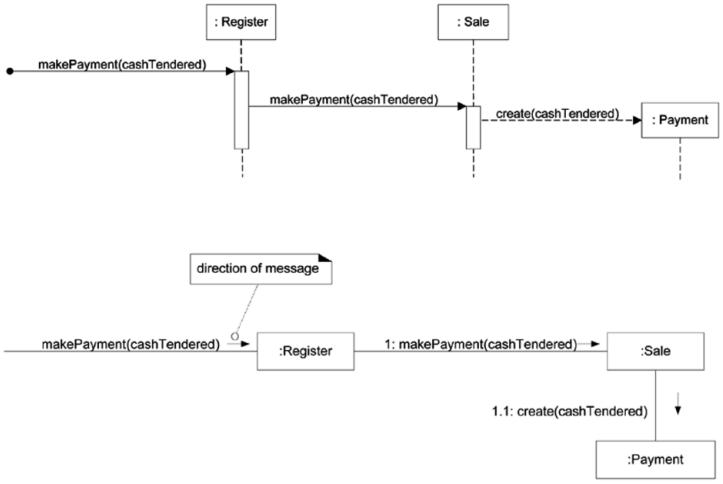
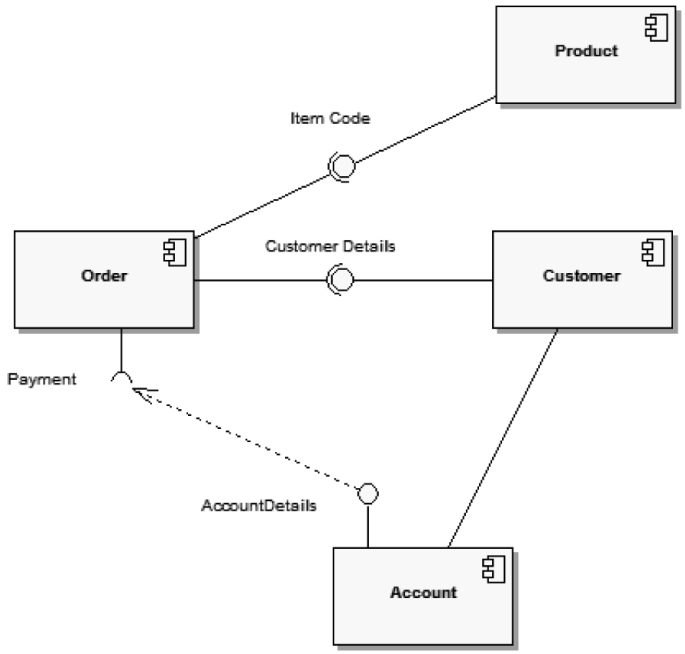


Diagrama de Componentes



Arquitectura de Software

Modelo de Vistas 4+1

Casos de Uso

- CU Críticos, no todos los escenarios.

Lógica

- Estilo, subsistemas (diagramas de componentes), interacciones (diagramas de secuencia).

Distribución

- Cómo se distribuyen los subsistemas en niveles físicos (nodos), diagrama de componentes.

Implementación

Procesos

Documento de arquitectura de software

Introducción.

[
En esta sección se realiza una introducción al sistema cuya arquitectura se describe.
Para la elaboración de esta plantilla se tomó como base el modelo 4+1 [10].
]

Vista de Casos de Uso

[
• Los elementos de las cuatro vistas desde las que se analiza la arquitectura son "ejercitados" por un pequeño subconjunto de casos de uso (o escenarios de esos casos de uso) que llamamos casos de uso críticos. Este subconjunto se debe elegir cuidadosamente, utilizando principalmente dos criterios:

- Que intervengan el mayor número de componentes arquitectónicos posibles.
- Que intervengan los componentes que se vislumbran como críticos o más complejos.

]

Diagrama de Casos de Uso Críticos

[
En esta sección se utilizan los Diagramas de Casos de Uso de UML para mostrar los casos de uso críticos para la arquitectura.
Los elementos a utilizar en estos diagramas son:

Caso de Uso	Secuencia de acciones con un fin específico (Ej. Enroll in University)
Actor	Persona, organización o sistema externo que interactúa con el sistema (Ej. Student)
Asociación	Relación entre un Caso de Uso y un Actor que participa en él.
Frontera del Sistema	La Frontera del Sistema se indica con un rectángulo que encierra los Casos de Uso.

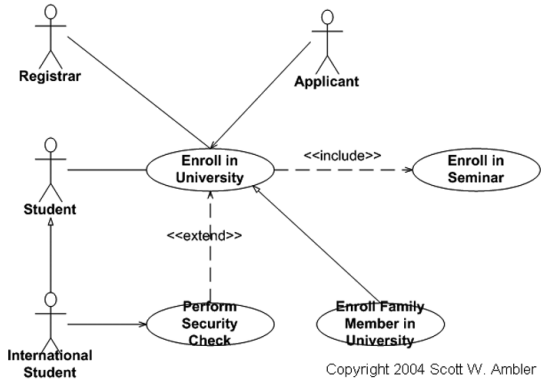


Diagrama de Ejemplo 1 (Ver [2])

]

2.2 Actores

[

En esta sección se realiza una descripción de cada actor perteneciente al diagrama anterior.

]

2.2.1 [Actor 1]

2.2.2 [Actor 2]

2.3 Especificación de Casos de Uso Críticos

[

En esta sección se incluye un ítem por cada caso de uso mostrado en el diagrama anterior. De cada caso de uso debe brindarse su descripción, así como uno o más flujos de eventos que detallen la interacción actor-sistema para cada escenario relevante del caso de uso. Opcionalmente, pueden especificarse pre y post-condiciones para cada caso. Las precondiciones son las condiciones que deben cumplirse para que el flujo del caso de uso pueda realizarse. Las post-condiciones son las condiciones que se cumplen al terminar el flujo del caso de uso (modificaciones en el estado del sistema).

]

2.3.1 [Caso de Uso Crítico 1]

Descripción

Pre-condiciones

Flujo de Eventos

Acción de Actor	Respuesta del Sistema
1	
	2
3	

Post-condiciones

2.3.2 [Caso de Uso Crítico 2]

3 Vista Lógica

[

La vista lógica permite describir el sistema en base a abstracciones fundamentales del diseño orientado a objetos para dar soporte a los requerimientos funcionales. En un enfoque top-down se comienza por descomponer el sistema en un conjunto de subsistemas "grandes", como ser las "capas" (layers) si se utiliza una arquitectura en capas, y a partir de ellos se realizan sucesivos refinamientos hasta llegar a las unidades lógicas más pequeñas.

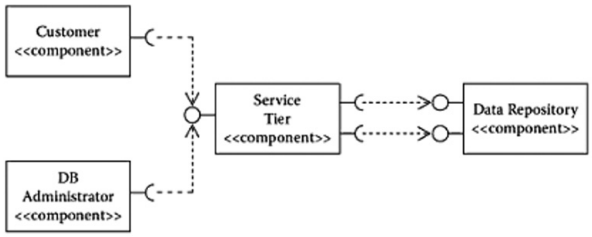
]

3.1 Estilo Arquitectónico

[

En esta sección se describe el estilo arquitectónico elegido para el sistema (capas estricto, capas no estricto, etc.). En el lenguaje de modelado UML 2 se utiliza el Diagrama de Componentes para representar componentes lógicos creados en tiempo de diseño, y no componentes físicos (ejecutables, bibliotectas, etc.) creados en tiempo de implementación, como sucede con el Diagrama de Componentes de UML 1. Se aconseja utilizar diagramas de componentes para representar subsistemas (considerar el clasificador <<subsystem>> como alternativa a <<component>>.

Diagrama de Ejemplo 2 (Ver [4])



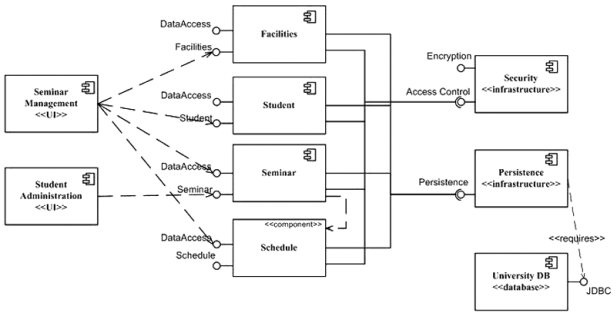


Diagrama de Ejemplo 3 (Ver [4])

]

3.2 Subsistemas

[

En esta sección se describe cada uno de los subsistemas. El diagrama de componentes muestra en detalle los componentes que corresponden a ese subsistema. Para cada componente deben detallarse las interfaces que implementa.

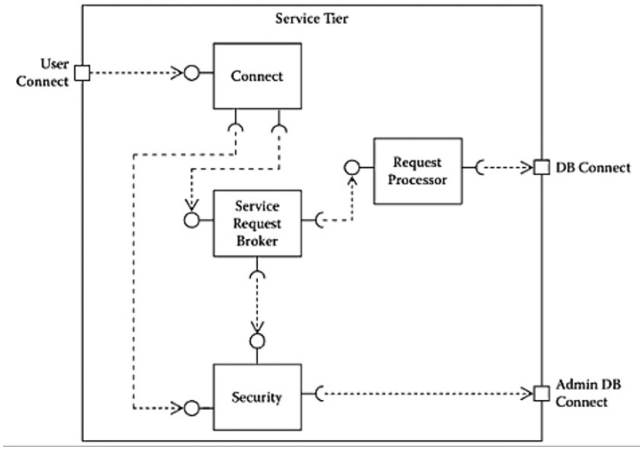


Diagrama de Ejemplo 4
(El subsistema es la capa de servicios)

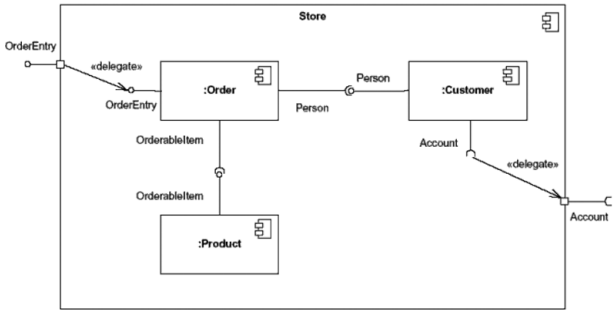


Diagrama de Ejemplo 5
(El subsistema es el componente Store)

]

3.2.1 [Subsistema 1]

Diagrama de Componentes.

Descripción.

3.2.1.1 [Subsistema 2]

3.3 Diagramas de Interacción

[

En esta sección se incluyen el o los diagramas de interacción de uno o más escenarios de cada caso de uso crítico que se considere apropiado especificar con mayor detalle. Se recomienda utilizar el Diagrama de Secuencia de UML, a nivel de componentes y no de clases.

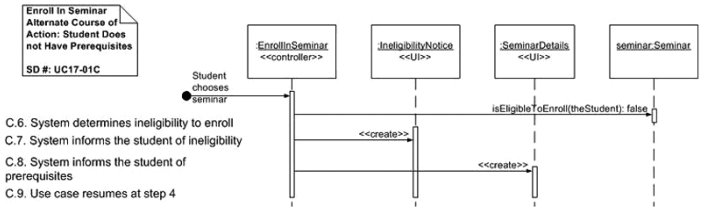


Diagrama de Ejemplo 6
(Diagrama de Secuencia para un caso de uso, ver [3])

]

4 Vista de Distribución (Deployment)

[

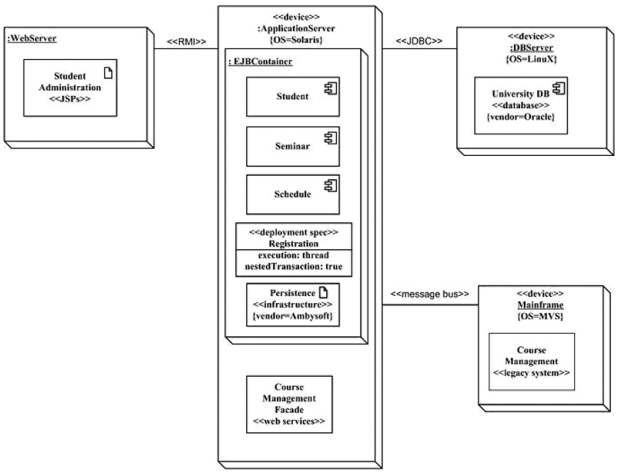
En la vista de distribución o deployment se plantean uno o varios escenarios de distribución de los componentes en tiers o nodos. Se utiliza el Diagrama de Deployment de UML que permite mostrar la estructura del escenario en base a nodos, conexiones entre nodos y componentes de cada nodo. Un nodo puede ser un elemento de hardware (un host, un router, etc.) o un elemento de software (un servidor de aplicaciones JEE, por ejemplo). Para distinguir los nodos físicos (hardware) se utiliza el estereotipo <<device>>.

Para cada escenario planteado interesa particularmente la descripción de los requerimientos no funcionales que justifican dicho escenario.

Opcionalmente, pueden describirse los nodos y conexiones presentes. De los nodos pueden especificarse requerimientos de software y hardware (sistema operativo, procesador, memoria, almacenamiento secundario, etc.) y de las conexiones los protocolos de comunicación, ancho de banda, etc.

Diagrama de Ejemplo 7
(Diagrama de Deployment con cuatro nodos, ver [5])

]



4.1 [Escenario 1]

Descripcion.
Nodos.
Conexiones.

4.2 [Escenario 2]

5 Vista de Implementación

[

La vista de implementación se focaliza en los componentes en tiempo de ejecución que forman el sistema (ejecutables, archivos de clases, bibliotecas, frameworks, etc.), que son la implementación de los componentes lógicos (provenientes de Diseño).

Dentro de esta vista, interesa mostrar las dependencias entre componentes implementados (utilizando los "artefactos"de UML).

También pueden mostrarse la distribución de los artefactos en los nodos.

En ambos casos se utiliza el Diagrama de Deployment de UML.

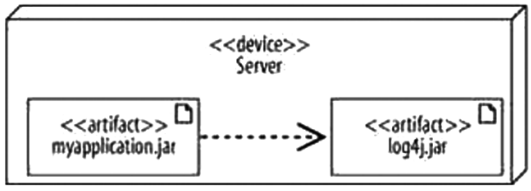


Diagrama de Ejemplo 9
(Dependencias entre artefactos, ver [6])

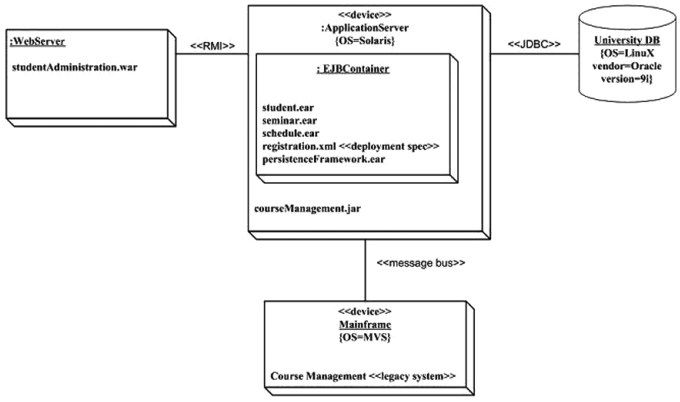


Diagrama de Ejemplo 10 (Ver [6])

Bibliografía

Agile Models Distilled: Potential Artifacts for Agile Modeling
http://www.agilemodeling.com/artifacts/
Agile Modeling. UML 2 Use Case Diagrams
http://www.agilemodeling.com/artifacts/useCaseDiagram.htm
Agile Modeling. UML 2 Sequence Diagrams
http://www.agilemodeling.com/artifacts/sequenceDiagram.htm
Agile Modeling. UML 2 Component Diagrams
http://www.agilemodeling.com/artifacts/componentDiagram.htm
Agile Modeling. UML 2 Deployment Diagrams
http://www.agilemodeling.com/artifacts/deploymentDiagram.htm
Deployed Software: Artifacts
http://codeidol.com/other/learnuml2/Modeling-Your-Deployed-System-Deployment-Diagrams/Deployed-Software-Artifacts/

Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives. N. Rozanski, E. Woods. Addison-Wesley, 2005

Software Architecture in practice, Second Edition. L. Bass, P. Clemens, R. Kazman. Addison-Wesley, 2003

An Introduction to Software Architecture. D. Garlan, M. Shaw. 1994. http://www.cs.cmu.edu/afs/cs/project/able/ftp/intro_softarch/intro_softarch.pdf

Architectural Blueprints — The "4+1" View Model of Software Architecture. Kruchten, Philippe. 1995. http://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf

Software Architecture Links. Bredemeyer Consulting.
http://www.bredemeyer.com/links.htm