Análisis y Diseño de Sistemas

Tema: RUP Agil. Arquitectura lógica y Paquetes UML

Mg. Ing. Adriana Urciuolo

UNTDF – 2015

Arquitectura Lógica y Paquetes UML

- Arquitectura de Software (Repaso)
  - ¿Qué es la arquitectura de software?
  - De qué se ocupa?
  - Estilos arquitecturales
- Aplicando UML:
  - Vistas de la Arquitectura en RUP.
  - Arquitectura lógica
  - Diagramas de paquetes UML para Arquitectura lógica
  - Guía de diseño arquitectural usando paquetes

RUP Agil. Dónde estamos???

Fase de Elaboración. Iteración I

Arquitectura Lógica
Introducción

Arquitectura de SW. Motivación

- Incremento en el tamaño y complejidad del software
- Necesidad de aprender de la experiencia: reutilización de estructuras asociadas a problemas similares
- Una adecuada estructura general es tan importante como las implementaciones concretas de las partes.

Definición (Bass, 1998)

La arquitectura de software de un programa o de un sistema computacional está definida por la estructura comprendida por los elementos de software, las propiedades visibles de esos elementos y las relaciones entre ellos.

Arquitectura de SW

- Incluyendo
  - la descripción de los componentes con los cuales se construyen los sistemas
  - las interacciones entre esos componentes
  - patrones para guiar la composición
  - restricciones sobre dichos patrones

- Componentes: servidores, clientes, bases de datos, filtros,capas en un sistema jerárquico, etc.

- Interacciones: llamadas a procedimientos, protocolos C/S, protocolos de acceso a BD, etc.

Arquitectura. Definición IEEE

- IEEE Std. 1471-2000

La arquitectura del software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y con el entorno, y los principios que orientan su diseño y evolución.

Arquitectura de SW. De qué se ocupa?

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

Arquitectura. De qué no se ocupa?

- Diseño detallado.
- Diseño de algoritmos.
- Diseño de estructuras de datos

En síntesis, la arquitectura es:

Un conjunto de decisiones acerca de:

- La organización de alto nivel de un sistema de software.
- La selección de los elementos estructurales y sus interfaces, que componen el sistema.
- Su comportamiento, expresado a través de las colaboraciones entre dichos elementos.
- La composición de dichos elementos estructurales y de comportamiento en subsistemas progresivamente mayores.
- El estilo arquitectural que guía esta organización

Estilos arquitectónicos

- Clasificación de los sistemas software en grandes familias cuyos integrantes comparten un patrón estructural común.

- Intervienen: patrón de organización general, tipos de componentes presentes habitualmente, interacciones entre ellos.

Ejemplo: Organización en capas

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

Mecanismos de interacción entre componentes:

- Llamadas a procedimientos.
- Llamadas a métodos.

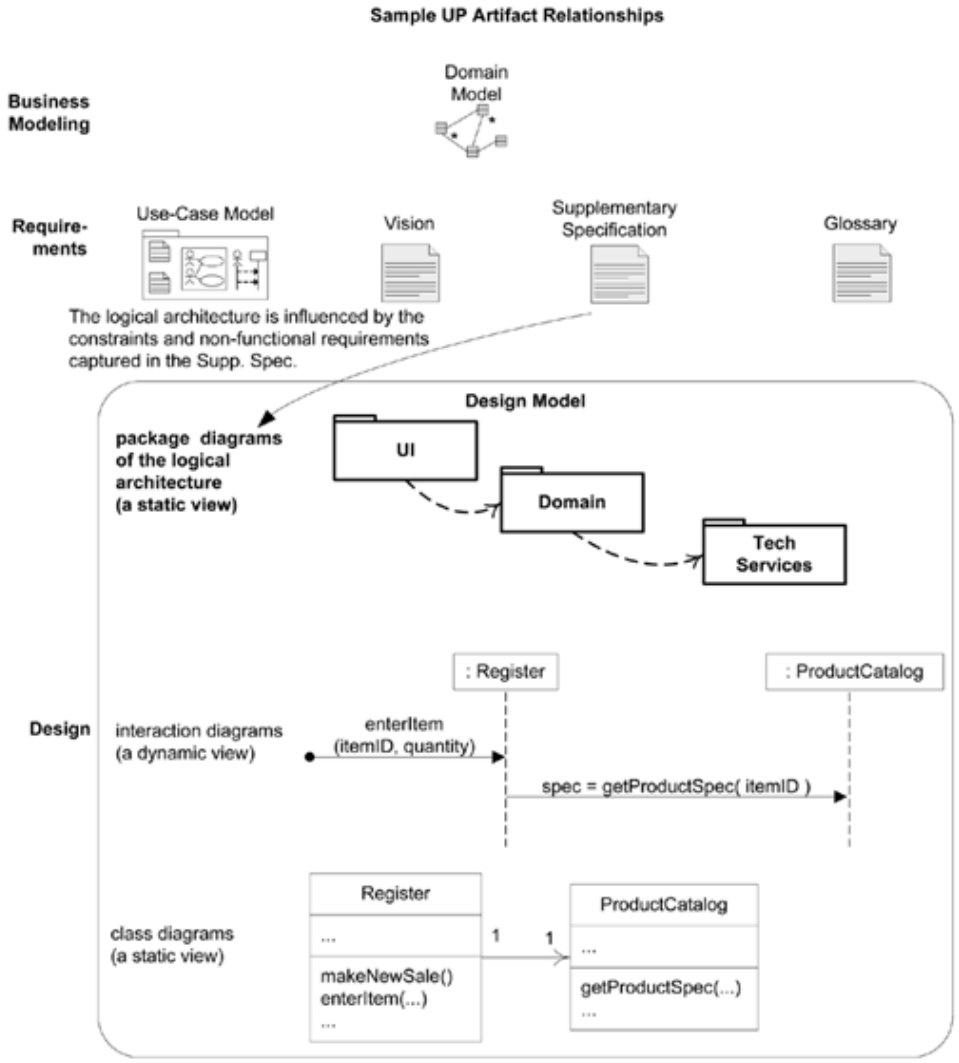
Ejemplo: Organización en capas

- En sistemas grandes o complejos, es conveniente particionar el software en subsistemas autónomos y si es necesario en "capas" (layers), que pueden ser consideradas subsistemas.
- Cada capa es un cluster de objetos colaborantes que dependen de las facilidades que ofrecen las capas inferiores.
- Las capas ayudan a reducir complejidad quebrando el sistema en piezas más manejables. Asimismo incrementan las chances de reuso.

Arquitectura de un sistema típico OO

- El diseño de un sistema típico de OO se basa en varias capas de arquitectura, tales como una capa de interfaz de usuario, una capa lógica de aplicación, etc.

Arquitectura de software en el marco de configuración del Proceso RUP Agil



RUP Agil. Dónde estamos???

Fase de Elaboración. Iteración I

Arquitectura Lógica

Disciplina: Diseño

Construyendo el Modelo de Diseño y el Documento de Arquitectura

Arquitectura Lógica

Aplicando UML en el marco de RUP Agil

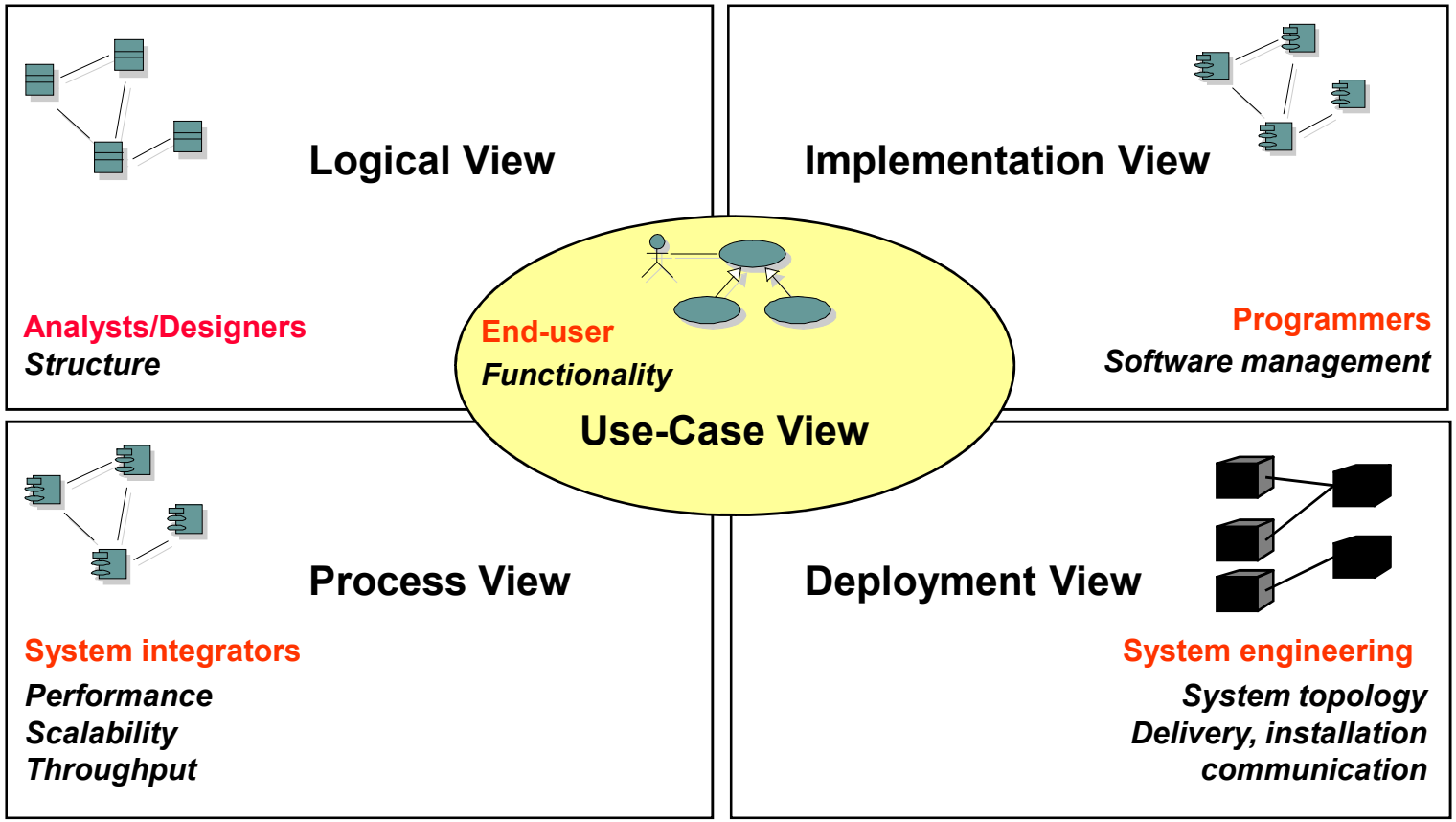
Vistas de la Arquitectura (Modelo 4+1)
Arquitectura Lógica
Diagramas de Paquetes
Guías de Arquitectura usando Paquetes

Vistas de la Arquitectura

- Para la descripción de la Arquitectura de Software, se utiliza un modelo de múltiples Vistas o Perspectivas, cada una de ellas focalizada en un aspecto particular del mismo.
- Existen diferentes modelos de vistas de La Arquitectura de software. Consideraremos el caso de RUP
- **La arquitectura en Proceso Unificado abarca 5 vistas.** Por ejemplo:
- **La vista lógica describe el sistema en términos de su organización conceptual en capas, paquetes, frameworks, clases, interfaces y subsistemas**.
- El despliegue de la arquitectura describe el sistema en términos de la asignación de los procesos a unidades de procesos y la configuración de la red.

Modelo 4+1 en RUP

- Framework para Descripción de Arquitectura, basado en vistas lógicas y físicas UML y una vista funcional de casos de uso.



Vistas de la Arquitectura

- Vista Lógica (o de diseño): Describe aspectos estructurales, como la descomposición en subsistemas estructurados en packages con clases de diseño
- Vista de Procesos: Captura aspectos de diseño de concurrencia y sincronización
- Vista de Implementación: Describe el mapeo del software en el hardware y releja sus aspectos distribuidos
- Vista de Despliegue: se enfoca en la organización de los módulos software en el entorno de desarrollo.

Vistas de la Arquitectura

- ***Vista De Casos de Uso:***

- Comprende los casos de uso que describen el comportamiento del sistema, como vistos por los usuarios finales, analistas y testers. No especifica la organización de un sistema de software, sino las fuerzas que le dan forma a su arquitectura.

- Con UML, los aspectos estáticos de esta vista se capturan en los Diagramas de casos de uso, los aspectos dinámicos se capturan en Diagramas de interacciones, diagramas de estado y diagramas de actividad.

Vistas de la Arquitectura

- ***Vista Lógica (De diseño):***
- Soporta el análisis y la especificación de los requisitos funcionales: lo que el sistema debería proporcionar en términos de servicios a sus usuarios. El sistema se descompone en un conjunto de abstracciones clave tomadas mayormente del dominio del problema, en forma de objetos o clases.
- Comprende las clases, interfaces y colaboraciones que forman el vocabulario del problema y su solución. Esta vista soporta los requerimientos funcionales del sistema, es decir lo servicios que el sistema proveerá a sus usuarios.

- Con UML, los aspectos estáticos de esta vista son capturados en Diagramas de de paquetes, de clases y de objetos, los aspectos dinámicos se capturan en Diagramas de interacciones, diagramas de estado y diagramas de actividad.

Vistas de la Arquitectura

- ***Vista de Procesos:***
- Se tratan algunos requisitos no funcionales. Ejecución, disponibilidad, tolerancia a fallos, integridad, etc. Esta vista también especifica que hilo de control ejecuta cada operación identificada en cada clase identificada en la vista lógica y los hilos (threads) y procesos que forman los mecanismos de concurrencia y sincronización del sistema.

- La vista se centra por tanto en la concurrencia y distribución de procesos.

- Con UML, los aspectos estáticos y dinámicos de esta vista son capturados en la misma forma que en la Vista de Diseño, pero con el foco en las clases activas que representan dichos hilos y procesos.

Vistas de la Arquitectura

- ***Vista de Implementación:***
- La vista física se centra en los requisitos no funcionales, tales como la disponibilidad del sistema, la fiabilidad (tolerancia a fallos), ejecución y escalabilidad. Y también presenta cómo los procesos, objetos, etc., corresponden a nodos de proceso: Componentes: nodos de proceso. Conectores: LAN, WAN, bus, etc.
- Comprende los componentes y archivos que se utilizan para ensamblar el sistema físico.

- Con UML, los aspectos estáticos de esta vista son capturados en un Diagrama de componentes; los aspectos dinámicos se capturan en Diagramas de Interacción, de estado y de actividad.

Vistas de la Arquitectura

- ***Vista de Despliegue:***
- La vista de desarrollo o despliegue se enfoca en la organización de los módulos software en el entorno de desarrollo. El software es empaquetado en pequeñas piezas (librerías de programa, subsistemas, componentes, etc.), los subsistemas se organizan en capas jerárquicas, y cada capa proporciona una interfaz bien definida a sus capas superiores
- Comprende los nodos que forman la topología de hardware del sistema sobre los cuales será ejecutado el mismo. Esta vista en forma primaria direcciona la distribución e instalación de partes para conformar el sistema físico.

- Con UML, los aspectos estáticos de esta vista son capturados en un Diagrama de despliegue; los aspectos dinámicos se capturan en Diagramas de Interacción, de estado y de actividad.

Arquitectura Lógica

- La Arquitectura lógica es la organización a gran escala de las clases software en Paquetes, subsistemas y capas.
- Se llama "Lógica" porque no se toman decisiones acerca de cómo son desplegados dichos elementos en nodos físicos
- El sistema se descompone en un conjunto de abstracciones clave tomadas mayormente del dominio del problema, en forma de objetos o clases que se agrupan en subsitemas.

- Esta vista soporta los requerimientos funcionales del sistema, es decir los servicios que el sistema proveerá a sus usuarios.

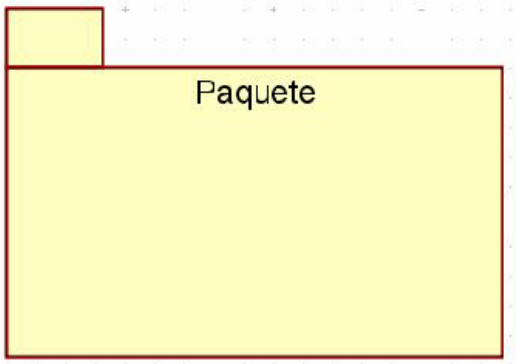
Arquitectura Lógica y Paquetes UML

- Modelar sistemas medianos o grandes conlleva manejar una cantidad considerable de elementos de modelado (clases, interfaces, componentes, nodos, relaciones, diagramas).
  - A partir de un cierto tamaño, es necesario organizar estos elementos en bloques mayores.
  - La mejor forma de comprender un sistema complejo es agrupando las abstracciones en grupos
  - Se agrupan aquellos elementos relacionados entre sí de acuerdo a algún criterio.

- En UML las abstracciones que permiten organizar un modelo se llaman **Paquetes (Packages)**.

Paquetes UML

Un paquete es un mecanismo de propósito general para organizar un modelo de manera jerárquica

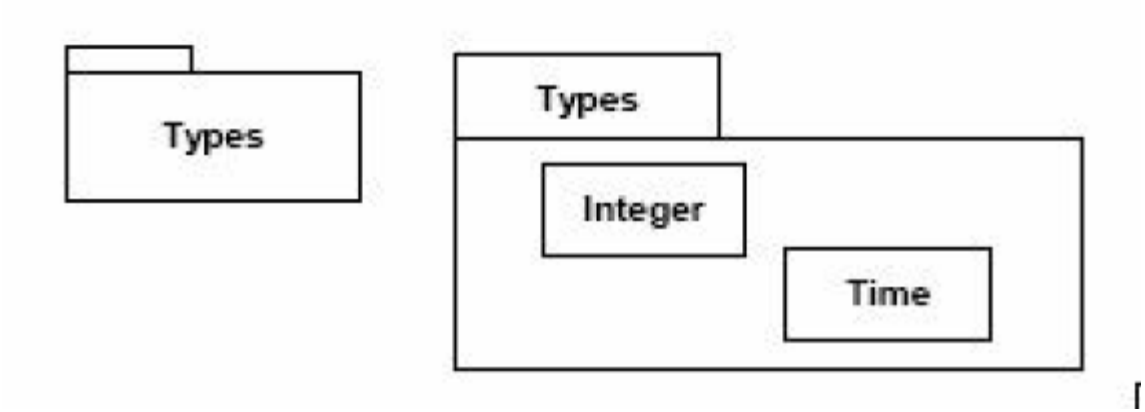


- **Utilidades principales**
  - Organizar los elementos en los modelos para comprenderlos más fácilmente.
  - Controlar el acceso a sus contenidos para controlar las líneas de separación de la arquitectura del sistema
  - Son un mecanismo importante para gestionar la complejidad del modelado.

- Son completamente diferentes a las clases:
  - Las clases son abstracciones de aspectos del problema o la solución.
  - Los paquetes son mecanismos para organizar, pero no tienen identidad

Paquetes UML. Notación

- Existen varias maneras de representar gráficamente el contenido de un paquete.
  - Sin especificar su contenido => El nombre aparece en la carpeta
  - Notación interna: incluyéndolo dentro de la carpeta => El nombre aparece en la pestaña



Paquetes UML. Contenidos

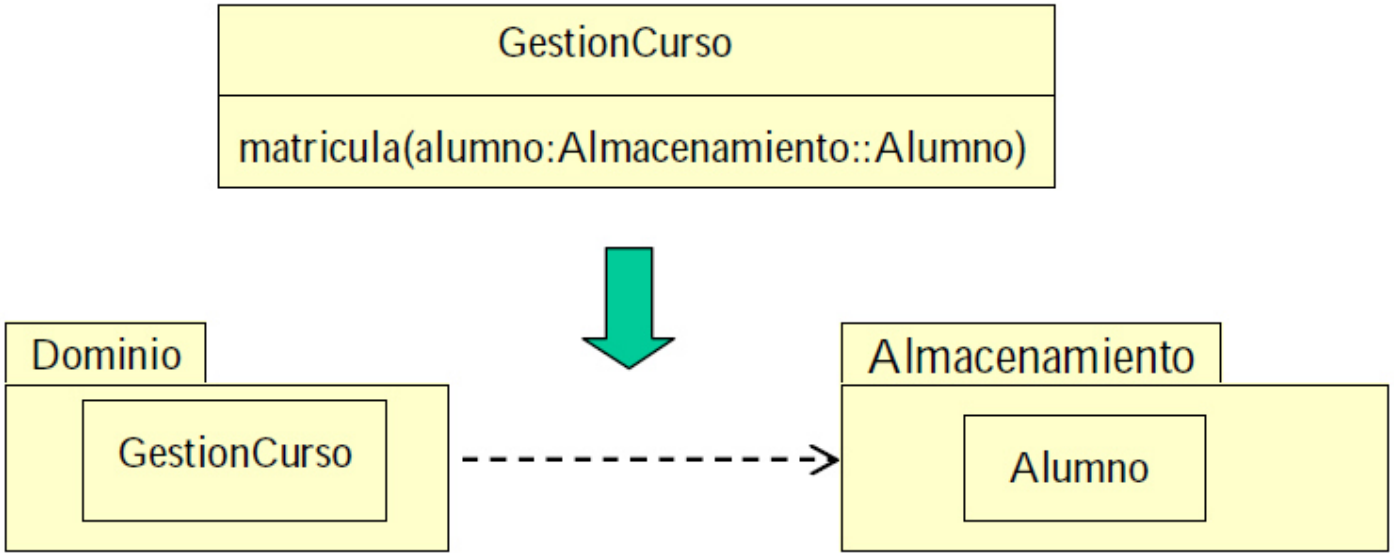
- Existen varias maneras de representar gráficamente el contenido de un paquete.
- Un paquete puede contener diferentes tipos de elementos como clases, interfaces, componentes, nodos, colaboraciones, casos de uso e incluso otros paquetes.
- Los paquetes bien diseñados agrupan elementos cercanos semánticamente:
  - Fuertemente cohesionados
  - Débilmente acoplados
- Entre un paquete y sus elementos existe una relación de composición =>
  - Cada elemento del modelo pertenece a un único paquete (aquel en el que es declarado) aunque puede ser referenciado desde otros.
  - Si el paquete se destruye, todos los elementos que contiene son también destruidos.

Paquetes UML. Contenidos

- Los paquetes pueden contener a otros paquetes (se forman jerarquías de paquetes).
- Los paquetes anidados tienen acceso al espacio de nombres del paquete que los contiene.
  - No hace falta cualificar los nombres
- No recomendable más de 3 niveles.
- UML asume que existe un paquete raíz anónimo (root).

Paquetes UML. Relaciones

- Las dependencias entre paquetes denotan que algún elemento de un paquete depende de los elementos en otro paquete.



Paquetes UML. Relaciones

- Entre paquetes puede haber tres tipos de relaciones de dependencia:
  - Importación: modelado como una dependencia estereotipada con <<Import>>
  - Acceso: modelado como una dependencia estereotipada con <<Access>>
  - Exportación: modelado implícitamente a través de la visibilidad pública en los elementos de un paquete
    - No se exporta explícitamente a algún paquete.
    - Se pone público, para que cualquier otro paquete pueda importarlo.
- También se pueden importar o acceder elementos de un paquete en vez de paquetes completos.

Paquetes UML. Relaciones

- Todos los mecanismos de extensibilidad pueden ser aplicados a paquetes

- Con frecuencia se añaden estereotipos y valores etiquetados para incorporar nuevas propiedades a los paquetes.

- Existen varios estereotipos que definen nuevas categorías de paquetes (además de profile):
  - <<framework>>: Contiene elementos reusables como clases, patrones y plantillas que definen una arquitectura aplicable a un sistema.
  - <<modelLibrary>>: Contiene elementos de modelado que pueden ser reutilizados por diferentes modelos.

Diagramas de Paquetes

- Presentan cómo se organizan los elementos de modelado en paquetes y las dependencias entre ellos, incluyendo importaciones y fusiones de paquetes.

- Contenido de un diagrama de paquetes:
  - Paquetes
  - Dependencias entre paquetes

Diagramas de Paquetes

- Los paquetes son especialmente útiles en modelado de:
  - Grupos de Elementos Relacionados
  - Vistas Arquitecturales

Paquetes y Diagramas de clases

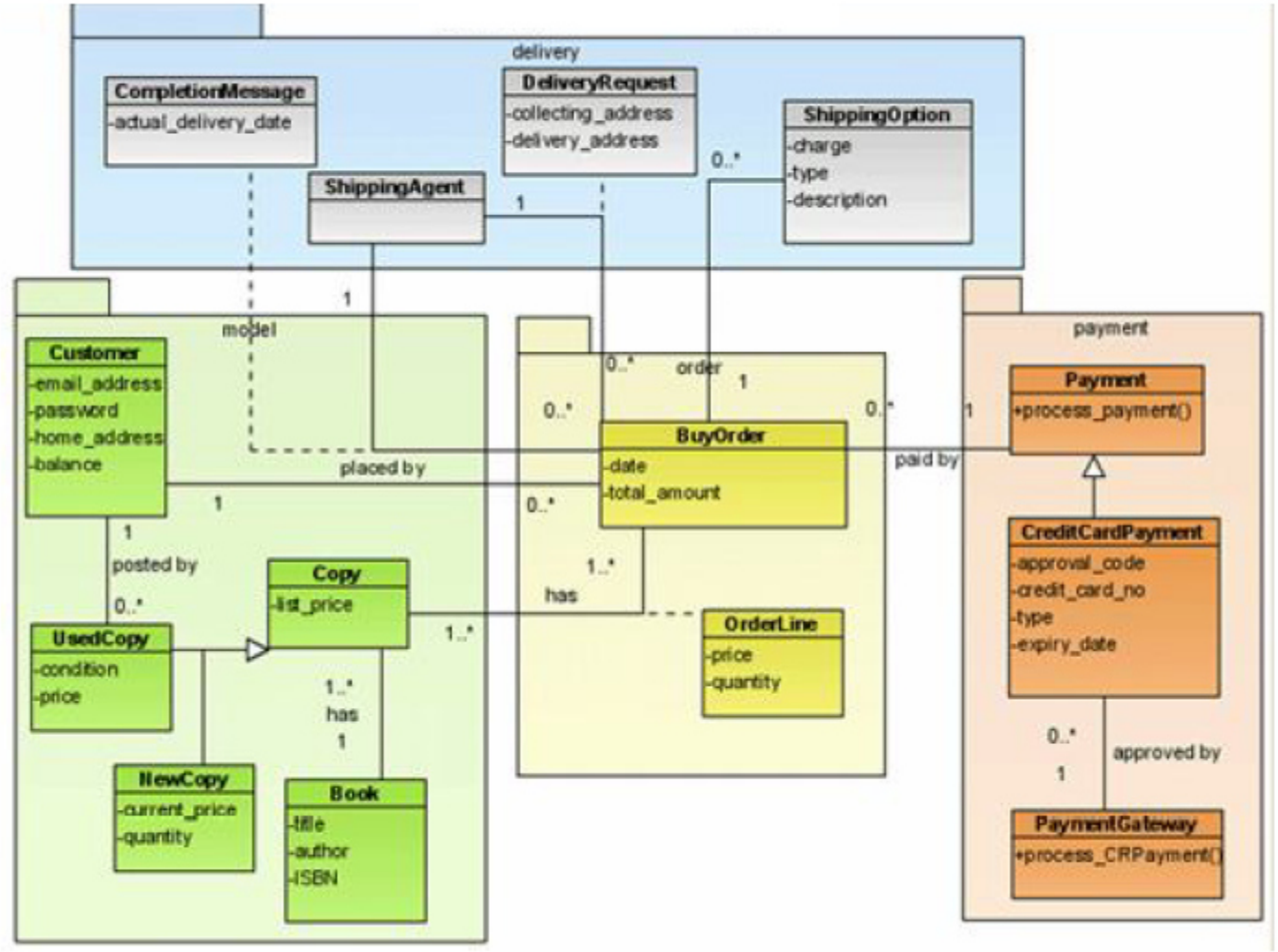
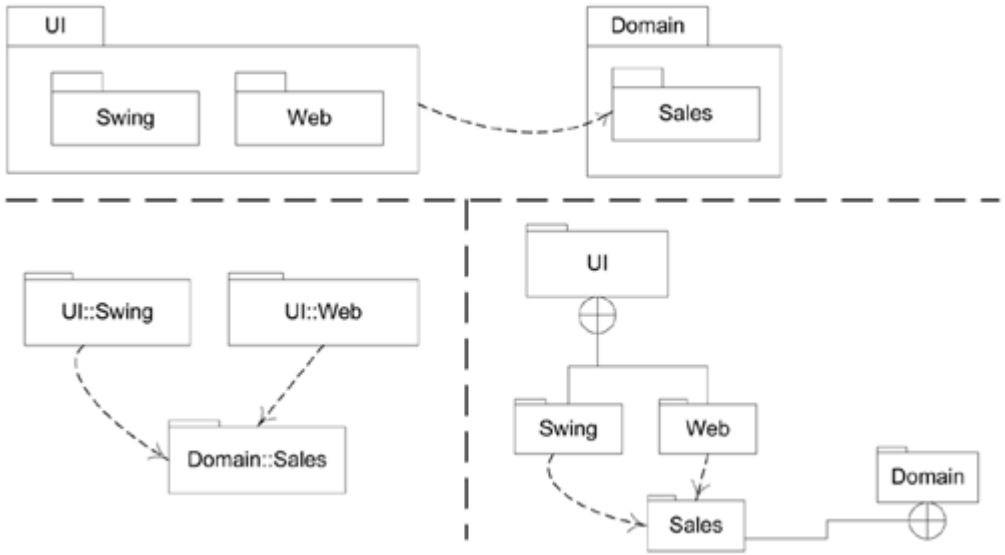


Diagrama de Paquetes para Arquitectura

- Los Diagramas de paquetes UML a menudo se utilizan para ilustrar la arquitectura lógica de un sistema, las capas, los subsistemas, etc.
- Una capa puede ser modelada como un paquete UML.
- Un diagrama de paquetes UML proporciona una forma de agrupar elementos



Las líneas punteadas muestran acoplamiento (dependencia) entre packages

Capas

- Una capa es un grupo de clases, packages, o subsistemas que tienen responsabilidades cohesivas para un aspecto del sistema.

- Las capas se organizan de forma que las capas más altas llaman servicios de las bajas (no viceversa).

- Normalmente las capas de un sistema OO incluyen:
  - Interface de usuario
  - Lógica de aplicación u Objetos de Dominio
  - Servicios técnicos

Guías de Diseño con capas

- Las ideas esenciales del uso de capas son simples:
  - Organizar la estructura lógica a gran escala de un sistema en capas discretas de distintas responsabilidades relacionadas, con una separación clara y cohesiva de asuntos tal que las 'capas bajas' son servicios de bajo nivel y generales, y las capas más altas son más específicas.
  - La colaboración y el acoplamiento es de capas más altas a capas inferiores; el acoplamiento de capa menor a mayor se evita.

Beneficios de Arquitectura con capas

- El uso de capas ayuda a resolver varios problemas:
  - Muchas partes del sistema están altamente acopladas.
  - La lógica de la aplicación se entrelaza con la interfaz de usuario.
  - Los Servicios técnicos generales o la lógica de negocio se entrelazan con más lógica específica de la aplicación.
  - Existe alto acoplamiento en las diferentes áreas de interés.

Beneficios de Arquitectura en capas

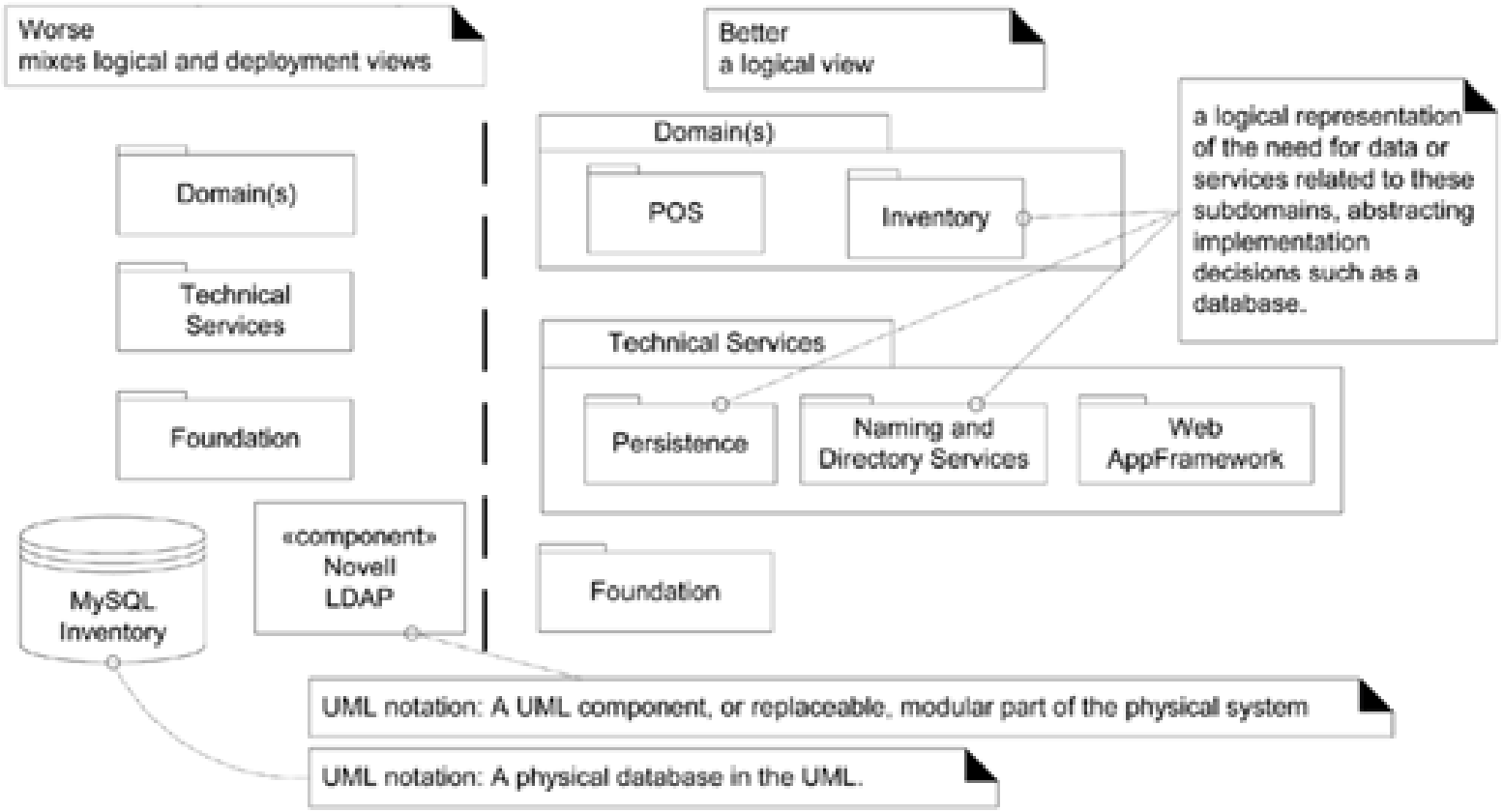
- Ventajas de usar capas
  - Hay una separación de aspectos, una separación de servicios de alto y bajo nivel, y de los servicios específicos de los servicios generales. Esto reduce el acoplamiento y dependencias, mejora la cohesión, aumenta el potencial de reutilización, y la claridad aumenta.
  - La complejidad relacionada es encapsulada y descomponible.
  - Algunas capas pueden ser reemplazadas con nuevas implementaciones.
  - Las capas inferiores contienen funciones reutilizables.
  - El desarrollo por equipos es ayudado por la segmentación lógica.

Diseño de Arquitectura en capas

- Capa de dominio vs Capa Lógica de Aplicación; objetos de dominio
  - Un sistema de software típico tiene una lógica de interfaz de usuario y la lógica de la aplicación.
  - Los **objetos de dominio representan "una cosa" en el espacio de** dominio del problema, y tienen lógica de aplicación y de negocio relacionada.
  - Diseñar objetos de ésta manera hace que a la capa de lógica de aplicación se le llame más precisamente la **capa de dominio de la** arquitectura: La capa que contiene los objetos de dominio para manejar el trabajo de la lógica de la aplicación.

Diseño de Arquitectura en capas

- No mostrar recursos externos como la capa inferior



Guía: Principio de separación Modelo-Vista

- Este principio tiene por lo menos dos aspectos:
  1. No conecte o acople objetos que no son de interfaz de usuario directamente a los objetos de interfaz de usuario.
  2. No ponga lógica de la aplicación en los métodos de los objetos de interfaz de usuario.

- En éste contexto, el modelo es un sinónimo de la capa de dominio de objetos.
- La Vista es un sinónimo de objetos de interfaz de usuario, tales como ventanas, páginas web, applets e informes.

Principio de separación Modelo-Vista

- El principio de separación de Modelo-Vista afirma que los objetos del modelo (dominio) no deben tener un conocimiento directo de la vista (UI).

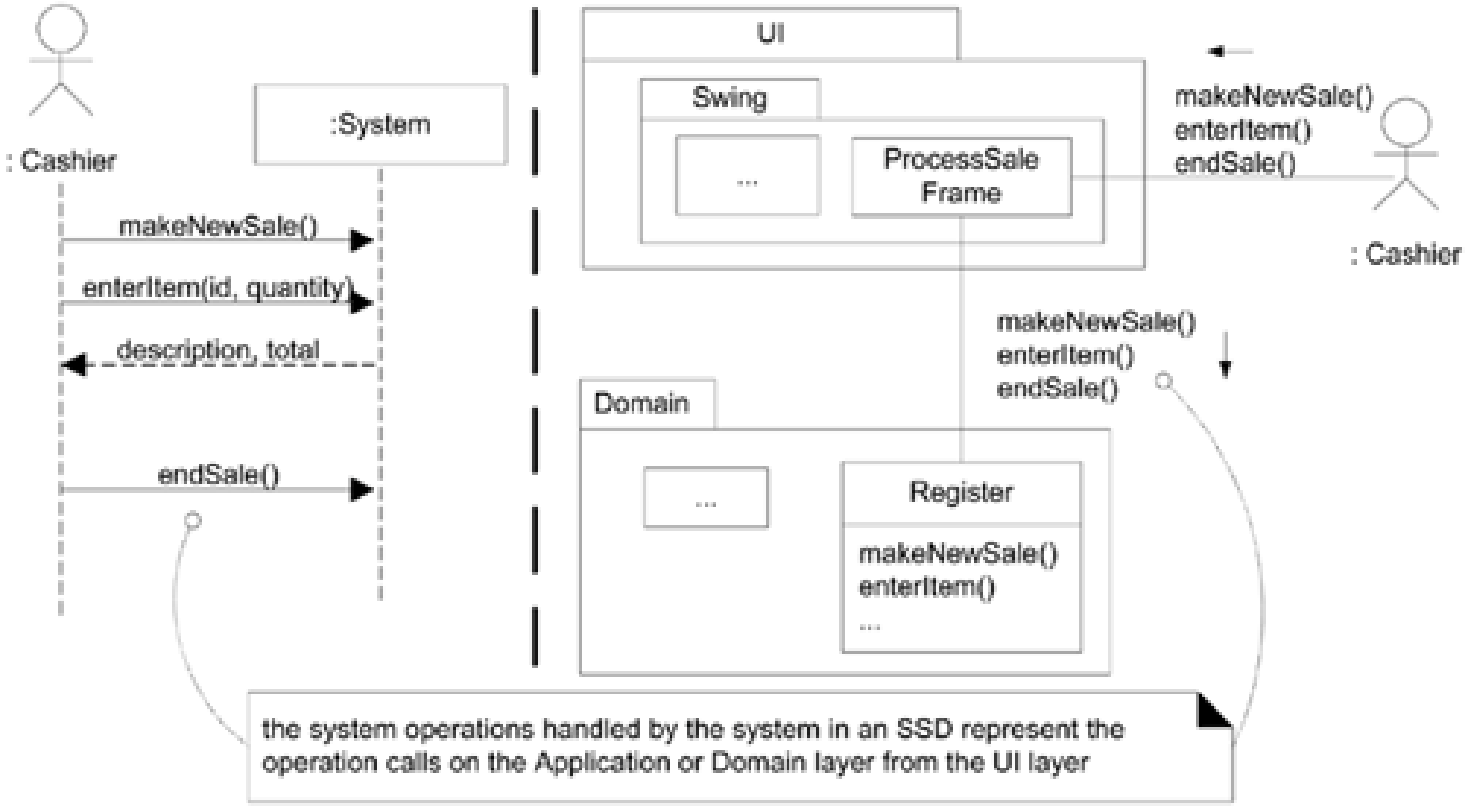
- Éste es un principio fundamental en el patrón Modelo-Vista-Controlador (MVC).
- El modelo es la capa de dominio, la vista es la capa de interfaz de usuario y los controladores son los objetos de flujo de trabajo en la capa de aplicación.

Motivación Separación Modelo-Vista

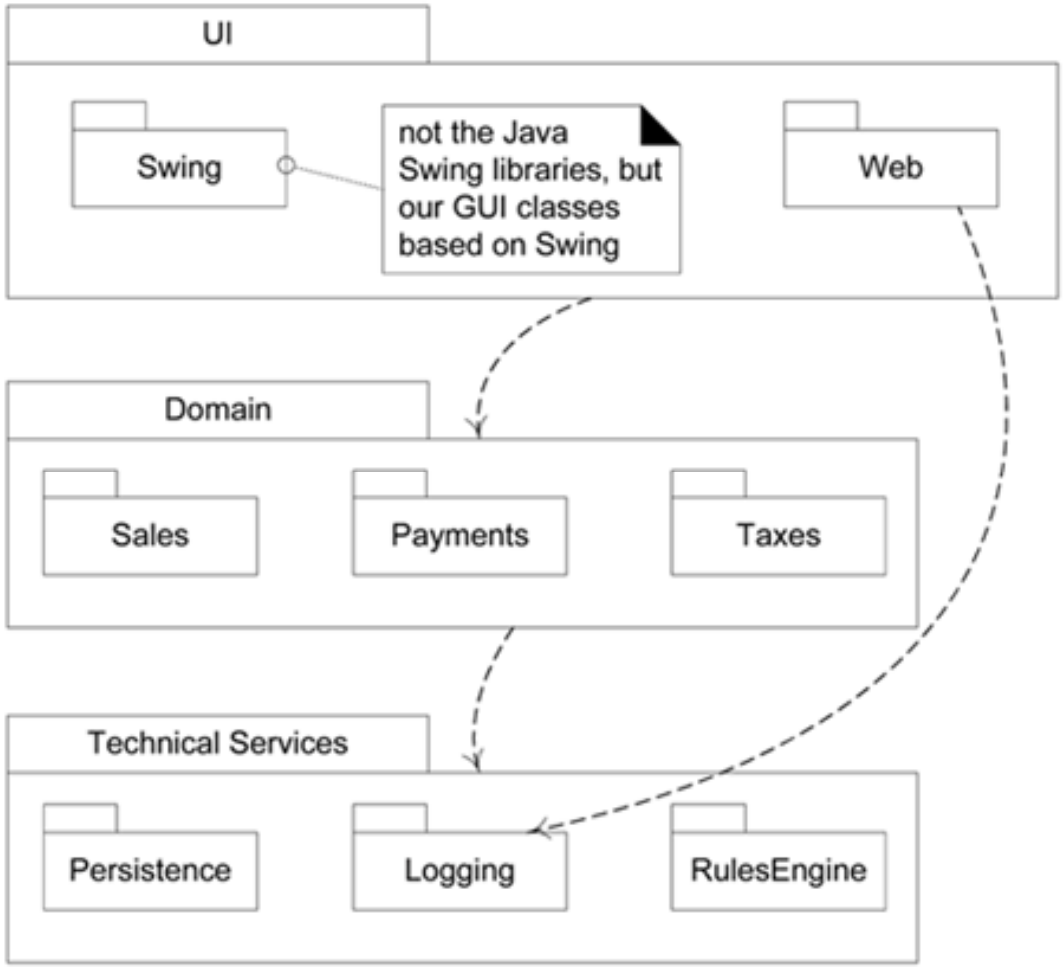
- Apoyar las definiciones de modelo coherentes que se centran en los procesos de dominio, en lugar de en las interfaces de usuario.
- Permitir el desarrollo independiente de la capa de modelo y de la capa de interfaz de usuario.
- Minimizar el impacto de los cambios en los requisitos de la interfaz sobre la capa de dominio.
- Permitir a nuevas vistas ser fácilmente conectadas a una capa de dominio existente, sin afectar a la capa de dominio.
- Permitir múltiples vistas simultáneas en el mismo objeto modelo.
- Permitir la ejecución de la capa de modelo independiente de la capa de interfaz de usuario, para permitir la fácil portabilidad de la capa de modelo a otro framework de interfaz de usuario.

Relación DSS y Capas

- Los mensajes enviados desde la capa de interfaz de usuario a la capa de dominio serán los mensajes ilustrados en los SSD



Arquitectura del Ejemplo PDV



Referencias

- El Proceso Unificado de Desarrollo de Software. Jacobson, Booch, Rumbaugh. Addison Wesley, 2000
- Applying UML and Patterns 3ª ed. Larman. Addison Wesley, 2004