**4.5 Diseño a Nivel de Componentes**

Después de haber definido la arquitectura global (Diseño Arquitectónico) y comprendido el dominio del problema (Análisis Orientado a Objetos), es hora de los detalles de cómo construir las piezas individuales de nuestro sistema. Esta fase se centra en descomponer el sistema en módulos y componentes, definiendo sus interfaces y sus interacciones internas. Es el paso del "qué" al "cómo" en un nivel más granular.

**4.5.1. Modularidad, Cohesión y Acoplamiento**

Estos tres conceptos son los pilares de un buen diseño a nivel de componentes. Son métricas cualitativas que nos guían hacia la creación de software más mantenible, extensible y reutilizable.

* **Modularidad:** Se refiere a la propiedad de un sistema de software que ha sido descompuesto en un conjunto de módulos o componentes discretos y lógicamente separados. Cada módulo encapsula una parte específica de la funcionalidad o los datos del sistema.
  + **Importancia:** Un sistema modular es más fácil de entender, desarrollar, probar y mantener. Permite que diferentes equipos trabajen en distintas partes del sistema de forma concurrente y facilita la reutilización de código.
  + **Ejemplo:** En un sistema de e-commerce, tendríamos módulos para "Gestión de Usuarios", "Gestión de Productos", "Procesamiento de Pagos", "Gestión de Envíos", etc.
* **Cohesión:** Mide el grado en que los elementos dentro de un módulo están funcionalmente relacionados entre sí. Un módulo con **alta cohesión** realiza una única tarea bien definida y sus elementos internos están fuertemente relacionados para lograr esa tarea.
  + **Tipos de Cohesión (de mejor a peor):**
    - **Funcional:** Todas las partes contribuyen a una única tarea bien definida (lo ideal).
    - **Secuencial:** Las salidas de una parte son las entradas de otra.
    - **Comunicacional:** Las partes operan sobre los mismos datos.
    - **Temporal:** Las partes se ejecutan al mismo tiempo (ej., inicialización).
    - **Lógica:** Varias funciones relacionadas lógicamente se agrupan.
    - **Coincidental:** No hay relación discernible (lo peor).
  + **Importancia:** Módulos altamente cohesivos son más fáciles de entender, mantener y reutilizar porque hacen una sola cosa y la hacen bien.
  + **Ejemplo:** Un módulo de "Cálculo de Impuestos" que solo se encarga de determinar impuestos sobre diferentes productos tendría alta cohesión funcional. Un módulo llamado "Utilidades Varias" que hace de todo un poco tendría baja cohesión.
* **Acoplamiento:** Mide el grado de interdependencia entre módulos. Un sistema con **bajo acoplamiento** significa que los módulos son relativamente independientes entre sí, y los cambios en un módulo tienen un impacto mínimo en otros módulos.
  + **Tipos de Acoplamiento (de mejor a peor):**
    - **Acoplamiento de Datos:** Módulos se comunican solo a través de datos simples (lo ideal).
    - **Acoplamiento de Sellos:** Módulos comparten estructuras de datos completas, pero solo usan una parte.
    - **Acoplamiento de Control:** Un módulo controla el flujo de ejecución de otro.
    - **Acoplamiento Externo:** Módulos dependen de un dispositivo o sistema externo.
    - **Acoplamiento Común:** Módulos comparten una estructura de datos global.
    - **Acoplamiento de Contenido:** Un módulo modifica directamente el contenido interno de otro (lo peor).
  + **Importancia:** Bajo acoplamiento facilita la modificación, prueba y depuración de módulos, ya que los cambios están localizados.
  + **Ejemplo:** Si el módulo de "Procesamiento de Pagos" necesita conocer los detalles internos (cómo se almacena el ID) del módulo de "Gestión de Usuarios", tienen alto acoplamiento. Si solo necesitan el ID del usuario como un dato simple, el acoplamiento es bajo.

La meta es siempre buscar **alta cohesión** y **bajo acoplamiento** para lograr una **modularidad** efectiva.

**4.5.2. Diagrama de Paquetes**

Un **Diagrama de Paquetes** en UML se utiliza para organizar elementos del modelo (como clases, interfaces, componentes e incluso otros paquetes) en grupos lógicos. Un paquete es una forma de agrupar elementos relacionados y sirve para estructurar modelos complejos, manejar dependencias y establecer límites de visibilidad.

* **Propósito:** Proporciona una vista de alto nivel de la organización del sistema. Nos ayuda a entender cómo se agrupan las responsabilidades y a identificar dependencias lógicas entre grandes bloques de funcionalidades. Es útil para aplicar los principios de modularidad, cohesión y acoplamiento a un nivel más abstracto.
* **Elementos:**
  + **Paquete:** Representado como una carpeta con una pestaña. Contiene elementos del modelo.
  + **Dependencia:** Una línea discontinua con una flecha que va desde el paquete que depende hacia el paquete del que depende. Indica que los elementos dentro del paquete de origen necesitan elementos dentro del paquete de destino.

**Ejemplo Aplicado:**

Para un sistema de gestión de un hospital, un diagrama de paquetes podría lucir así:

Aquí, Administracion podría depender de RecursosHumanos (para gestionar personal), AdmisionPaciente podría depender de Administracion (para registrar pacientes), y HistorialClinico podría depender de Laboratorio y Farmacia (para registrar resultados de pruebas y medicamentos dispensados). Facturacion podría depender de varios paquetes para generar facturas.

**4.5.3. Diagramas de Clases de la Solución**

A diferencia del Diagrama de Clases del Dominio (que es conceptual y de análisis), los **Diagramas de Clases de la Solución** son diagramas de **diseño**. Muestran las clases que realmente se implementarán en el código, incluyendo detalles de la implementación como:

* **Atributos:** Con sus tipos de datos (ej., nombre: String, edad: int).
* **Métodos (Operaciones):** Con sus parámetros, tipos de retorno y visibilidad (ej., +calcularPrecio(): double, -validarDatos(datos: String): boolean).
* **Visibilidad:** Pública (+), privada (-), protegida (#), paquete (~).
* **Relaciones de Diseño:**
  + **Asociación:** Relaciones entre clases (como en el dominio, pero ahora reflejando el diseño).
  + **Agregación:** Un tipo de asociación "tiene-un" donde la parte puede existir independientemente del todo (ej., un Departamento tiene Empleados, pero los Empleados pueden existir sin un Departamento).
  + **Composición:** Un tipo de asociación "tiene-un" donde la parte no puede existir sin el todo (ej., un Pedido tiene DetallesPedido; si se elimina el Pedido, los DetallesPedido asociados también se van).
  + **Generalización (Herencia):** Una relación "es-un" donde una subclase hereda de una superclase (ej., Coche es un tipo de Vehiculo).
  + **Realización (Implementación de Interfaz):** Una clase implementa una interfaz (ej., ClaseA realiza InterfazB).
  + **Dependencia:** Una clase usa otra clase, pero no la contiene como un atributo (ej., una clase Calculadora podría depender de una clase Matematica para una operación específica).

**Ejemplo Aplicado:**

Para nuestro Pedido del sistema de e-commerce, la clase de la solución podría ser:

Este diagrama proporciona un "plano" detallado para los desarrolladores, indicando cómo se estructurarán las clases en el código.

**4.5.4. Diagramas de Secuencia**

Los **Diagramas de Secuencia** son diagramas de interacción en UML que muestran cómo los objetos interactúan entre sí en una secuencia de tiempo para lograr un escenario o caso de uso particular. Son excelentes para visualizar la comunicación y el flujo de control entre objetos.

* **Elementos:**
  + **Líneas de vida (Lifeline):** Representan un objeto individual o un participante en la interacción, dibujado como una línea vertical discontinua.
  + **Mensaje:** Una flecha que va de una línea de vida a otra, representando una llamada a un método o un evento. Indica el orden cronológico de las interacciones.
  + **Activación (Execution Occurrence):** Un rectángulo delgado en la línea de vida que indica el período durante el cual un objeto está activo y realizando una operación.
  + **Fragmentos de Combinación (Combined Fragments):** Elementos para modelar lógica condicional (alt), bucles (loop), o referencias a otras secuencias (ref).

**Ejemplo Aplicado: Proceso de "Realizar Pedido"**

Participantes:

:Cliente

:PedidoService (API de la tienda)

:ProductoService

:StockService

:PagoGateway

:RepositorioPedido

Flujo:

1. :Cliente -> :PedidoService: crearPedido(idProducto, cantidad)

2. :PedidoService -> :ProductoService: getPrecio(idProducto)

3. :ProductoService --> :PedidoService: precio

4. :PedidoService -> :StockService: verificarStock(idProducto, cantidad)

5. :StockService --> :PedidoService: stockDisponible (true/false)

6. :PedidoService -> :PagoGateway: procesarPago(monto, datosTarjeta)

7. :PagoGateway --> :PedidoService: resultadoPago (exitoso/fallido)

8. if resultadoPago == "exitoso":

9. :PedidoService -> :RepositorioPedido: guardarPedido(datosPedido)

10. :RepositorioPedido --> :PedidoService: pedidoGuardado

11. :PedidoService -> :StockService: reducirStock(idProducto, cantidad)

12. :StockService --> :PedidoService: stockActualizado

13. :PedidoService --> :Cliente: pedidoConfirmado(idPedido)

14. else:

15. :PedidoService --> :Cliente: pagoFallido

Este diagrama muestra claramente el orden de las llamadas y las dependencias entre los servicios para completar el proceso de un pedido.

**4.5.5. Diagramas de Colaboración (o de Comunicación)**

Los **Diagramas de Colaboración** (renombrados a **Diagramas de Comunicación** en UML 2.0) también son diagramas de interacción, pero se enfocan en la **estructura de la colaboración entre objetos** para lograr un objetivo. A diferencia de los diagramas de secuencia que enfatizan el tiempo, los diagramas de colaboración enfatizan las relaciones entre los objetos y el orden de los mensajes se indica con números.

* **Elementos:**
  + **Objetos (o Líneas de vida):** Representados como nodos.
  + **Enlaces:** Conectan los objetos, indicando que pueden comunicarse.
  + **Mensajes:** Flechas numeradas colocadas sobre los enlaces, mostrando el orden de la interacción. Los números pueden ser anidados para indicar llamadas dentro de llamadas (ej., 1:, 1.1:, 1.2:).

|

*Note: La representación de las condiciones y bucles es menos intuitiva que en los diagramas de secuencia.*

Mientras que un diagrama de secuencia es mejor para visualizar el flujo temporal, un diagrama de colaboración es excelente para mostrar la conectividad entre objetos y qué objetos participan en una interacción específica.

**4.5.6. Diagramas de Actividades**

Los **Diagramas de Actividades** en UML se utilizan para modelar el flujo de control o el flujo de trabajo de un proceso, ya sea un caso de uso, una operación, o incluso un proceso de negocio completo. Son muy similares a los diagramas de flujo, pero con elementos más ricos para modelar el comportamiento concurrente y las decisiones complejas.

* **Elementos:**
  + **Acción:** Una unidad discreta de trabajo, representada por un rectángulo con esquinas redondeadas.
  + **Nodo Inicial:** Un círculo relleno que marca el inicio del flujo.
  + **Nodo Final:** Un círculo relleno dentro de otro círculo que marca el final del flujo.
  + **Flujo (Control Flow):** Flechas que conectan las acciones, indicando la secuencia.
  + **Decisión (Decision Node):** Un rombo que representa un punto donde el flujo se bifurca basándose en una condición.
  + **Fusión (Merge Node):** Un rombo donde múltiples flujos condicionales se unen de nuevo.
  + **Bifurcación (Fork Node):** Una barra gruesa horizontal o vertical donde el flujo se divide en múltiples flujos concurrentes.
  + **Unión (Join Node):** Una barra gruesa donde múltiples flujos concurrentes se unen antes de continuar con un único flujo.
  + **Carriles (Swimlanes):** Rectángulos que dividen el diagrama para mostrar qué actor o departamento es responsable de qué actividades.

**Ejemplo Aplicado: Proceso de "Gestionar Devolución de Producto"**

+------------------------------------------------------------------+

| <<actividad>> Gestionar Devolución de Producto |

+------------------------------------------------------------------+

| |

| (Inicio) |

| O |

| | |

| V |

| +---------------------+ |

| | Cliente solicita | |

| | devolución | |

| +---------------------+ |

| | |

| V |

| +---------------------+ |

| | Registrar solicitud | |

| +---------------------+ |

| | |

| V |

| /------------------\ |

| | [Solicitud Válida?] | |

| \------------------/ |

| | \ |

| | \ [NO] |

| V \ |

| +--------------+ +------------------------+ |

| | [SI] | | Notificar cliente y | |

| | Aprobar | | cerrar solicitud | |

| | devolución | +------------------------+ |

| +--------------+ |

| | |

| V |

| +---------------------+ |

| | Generar etiqueta | |

| | de envío de retorno | |

| +---------------------+ |

| | |

| V |

| +---------------------+ |

| | Cliente envía | |

| | producto | |

| +---------------------+ |

| | |

| V |

| +---------------------+ |

| | Recibir producto | |

| | en almacén | |

| +---------------------+ |

| | |

| V |

| +---------------------+ |

| | Inspeccionar | |

| | producto | |

| +---------------------+ |

| | |

| V |

| /------------------\ |

| | [Producto OK?] | |

| \------------------/ |

| | \ |

| | \ [NO] |

| V \ |

| +--------------+ +------------------------+ |

| | [SI] | | Notificar cliente y | |

| | Procesar | | gestionar producto dañado | |

| | reembolso | +------------------------+ |

| +--------------+ |

| | |

| V |

| +---------------------+ |

| | Notificar cliente | |

| | reembolso realizado | |

| +---------------------+ |

| | |

| V |

| (Fin) |

| O |

+------------------------------------------------------------------+

Los diagramas de actividades son excelentes para modelar procesos de negocio complejos, flujos de trabajo paralelos y la lógica de control dentro de un algoritmo. Son particularmente útiles para la comunicación con las partes interesadas del negocio, ya que son fáciles de entender.

En resumen, el diseño a nivel de componentes nos permite pasar de la visión general del sistema a los detalles implementables, asegurando que cada pieza del rompecabezas esté bien definida, sea robusta y se integre armoniosamente con las demás. El uso de estos diagramas UML es fundamental para documentar, comunicar y validar nuestras decisiones de diseño.