INGENIERÍA DE SOFTWARE I

UNTDF
Universidad Nacional de Tierra del Fuego, Antártida e Islas del Atlántico Sur

Unidad V

Tema: RUP Ágil. Análisis. De los requerimientos al diseño

UNTDF – 2020

RUP Agil. Análisis

- Análisis
  - Modelado estructural
- Análisis de Eventos y Operaciones del Sistema.
  - Completando el Modelo de Casos de Uso con DSS y Contratos
- De los requerimientos al Diseño
  - Diseño de objetos
  - Asignación de responsabilidades a Objetos
  - Patrones GRASP

RUP Agil. Dónde estamos???

Fase de Elaboración. Iteración I

"Haciendo análisis":
Definiendo Modelo de Estructura del Sistema

Análisis

- En esta configuración de RUP Agil, durante el análisis de la estructura y comportamiento del sistema:
  - Se define una Visión Esencial del Sistema, de Nivel de abstracción más alto que en el diseño.
  - Se avanza hacia una versión preliminar del Modelo de Diseño.
  - Se comienza a delinear una arquitectura para el sistema
  - No se produce el Artefacto "Modelo del Análisis" propuesto en UP

Análisis. Modelado estructural

- Se construye un Diagrama de clases conceptuales del Sistema a partir del Modelo de Dominio y de los Casos de Uso. Para ello:
  - A partir del Modelo de Dominio, encontrar clases conceptuales candidatas que describan los objetos relevantes del sistema y registrarlas en un Diagrama de clases.
  - Recorrer los Casos de uso chequeando si son soportados por las clases, agregando y/o quitando al Modelo de Dominio las que sean necesarias
  - Encontrar nuevas relaciones entre clases (nombrar todas las relaciones)
  - Encontrar atributos para las clases

  - El Diagrama de clases conceptuales del sistema es una abstracción a mayor nivel que el de Diseño. Puede omitirse (pasando del Diagrama de clases del Dominio al del Diseño).

RUP Agil. Dónde estamos???

Fase de Elaboración. Iteración I

"Haciendo análisis". De los requerimientos al Diseño

Completando el Modelo de Casos de Uso

Definiendo DSS y Contratos

Proceso: De los requerimientos al Diseño

- A partir del **Modelo conceptual del sistema y de CU:**
- Para cada Caso de uso se define un **DSS (Diagrama de Secuencia del sistema)** que muestra los eventos que genera un actor durante la interacción con el sistema.
- Cada evento da origen a una **operación del sistema**
- El efecto de las operaciones se describe mediante **Contratos** especificados por una plantilla (esto es opcional)
- Para cada operación del sistema se define una **colaboración (Diagrama de interacción)** que satisface la postcondición del Contrato
- Se diseñan las colaboraciones asignando responsabilidades a las clases del Modelo conceptual. Se usarán patrones GRASP
- Se crea el Diagrama de clases del Diseño conforme se definen las colaboraciones.

Análisis de eventos y operaciones del Sistema

- Se realiza el análisis de los eventos de entrada y salida del sistema.
- Por qué?
  - Las operaciones que un actor externo solicita a un sistema constituyen una parte importante de la comprensión del comportamiento del sistema.
  - Se debe diseñar el SW para manejar esos eventos y ejecutar una respuesta del sistema
- Un Sistema de SW reacciona a:
  - eventos externos de actores
  - eventos producidos por el paso del tiempo
  - fallas y excepciones
- *Un diagrama de secuencia del sistema (DSS) es un artefacto que muestra los eventos de entrada y operaciones de salida relacionados con el sistema que se está estudiando.*

Qué son DSS en un marco RUP?

- **Los CU** describen la interacción de actores con el sistema software.
  - Durante una interacción, el actor genera eventos sobre el sistema normalmente solicitando alguna operación como respuesta.

- Un DSS es un gráfico que muestra, para un escenario particular de un CU, los eventos que generan los actores sobre el sistema.
  - UML no define un Diagrama de secuencia "del sistema", sino simplemente diagrama de secuencia.
  - El término DSS subraya la representación del sistema como cajas negras.

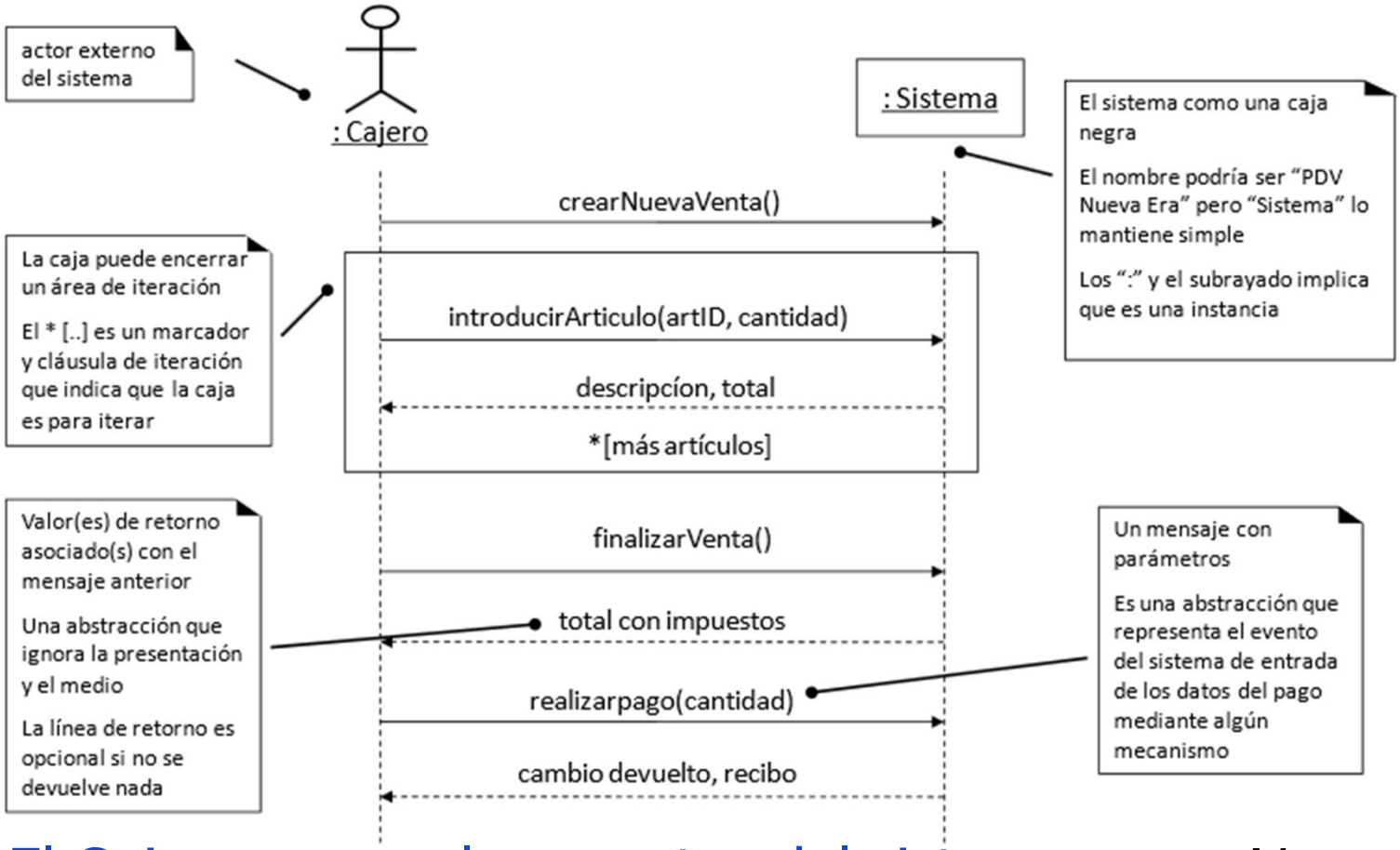
Qué son DSS en un marco RUP?

- Se dibuja un DSS para el escenario de éxito de cada caso de uso y escenarios alternativos frecuentes o complejos

- Un DSS muestra, para un curso de eventos específico en un caso de uso,
  - los actores externos que interaccionan directamente con el sistema,
  - el sistema (como una caja negra) y
  - los eventos del sistema que genera el actor

Ejemplo DSS

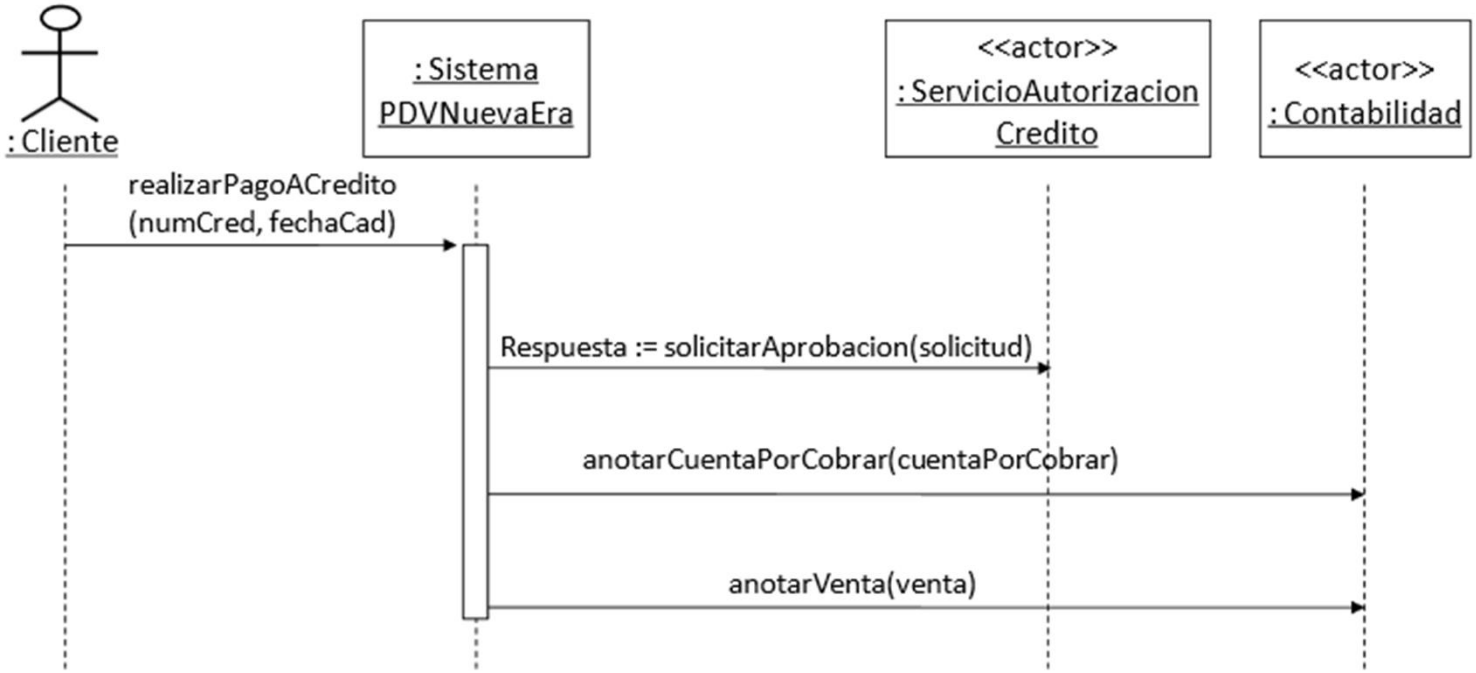
- Escenario principal de éxito del CU ProcesarVenta



- El Cajero genera los eventos del sistema *crearNuevaVenta,*
- *introducirArticulo, finalizarVenta* y *realizarPago*

Ejemplo DSS entre sistemas

- Los DSS también pueden utilizarse para ilustrar las colaboraciones entre sistemas

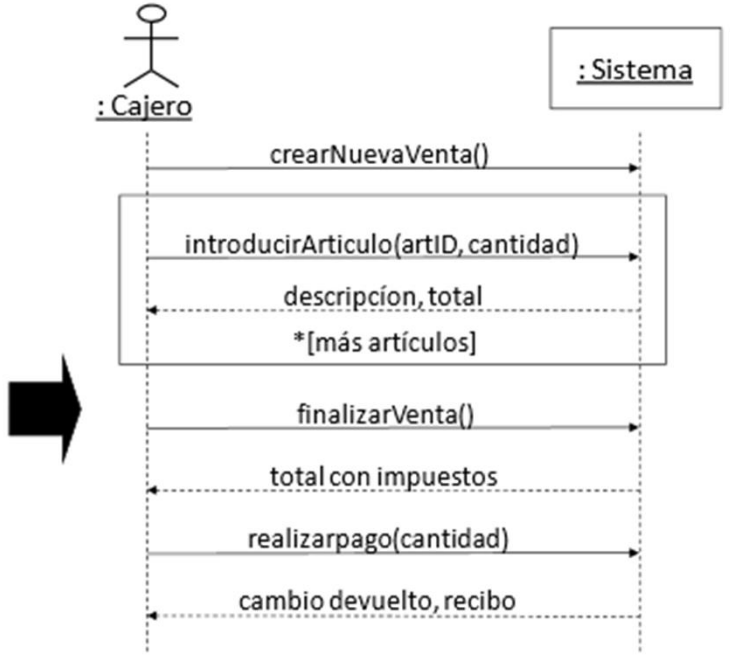


- El Cliente genera el evento realizarPagoACredito

DSS y Casos de Uso

- Se genera un DSS para el estudio de un Caso de uso

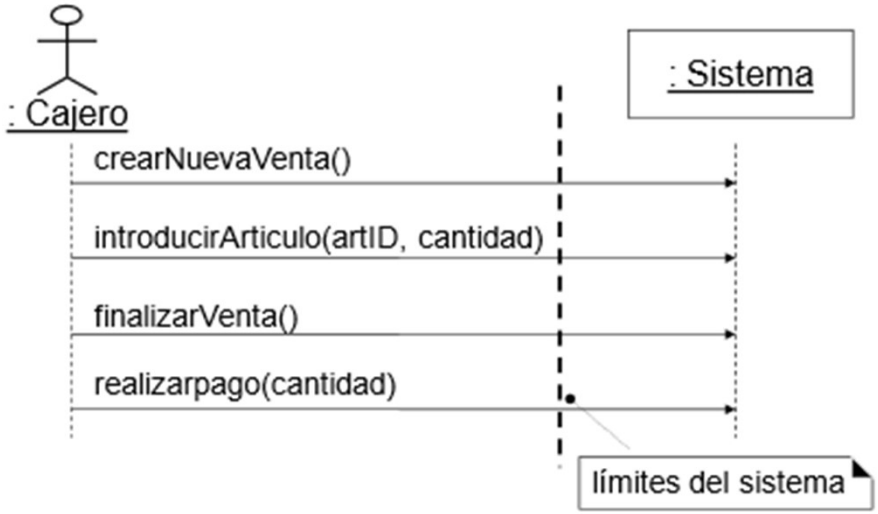
Escenario simple de *ProcesarVenta* para el pago en efectivo:
1. El cliente llega al terminal PDV
2. El Cajero inicia una nueva venta
3. El Cajero inserta el identificador del artículo
4. El Sistema registra la línea de venta y presenta la descripción del artículo, precio y suma parcial
El Cajero repite los pasos 3 y 4 hasta que se indique
5. El Sistema muestra el total con los impuestos calculados
6. El Cajero le dice al Cliente el total, y pide que le pague
7. El Cliente paga y el Sistema gestiona el pago.
...



- Los DSS se derivan de los Casos de uso

DSS y Límites del Sistema

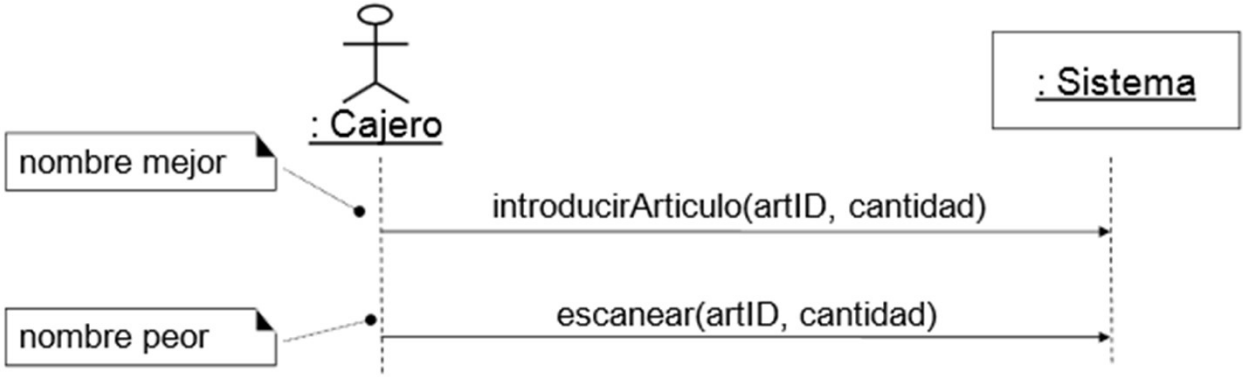
- Para identificar los eventos del sistema, es necesario tener claros los límites del sistema



- El Cliente no genera eventos porque no es usuario directo del sistema

Nombres de eventos y operaciones

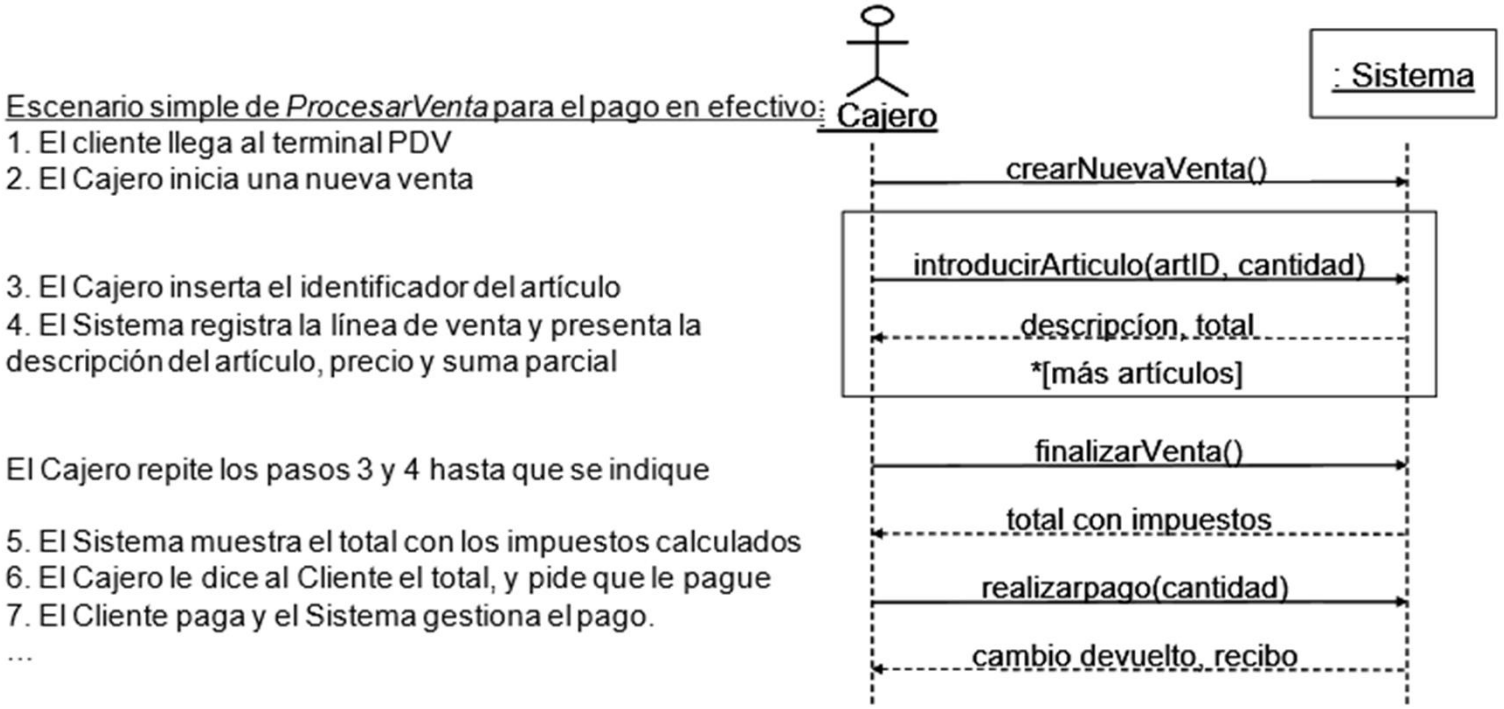
- Los eventos del sistema y operaciones del sistema asociadas deben nombrarse al nivel conceptual en lugar de medios de entrada físico o interfaz



- Se mejora la claridad, comenzando el nombre de un evento del sistema con un verbo (añadir..., insertar...)

DSS con texto de Casos de Uso

- Pueden mostrarse fragmentos del texto del escenario del Caso de uso para clarificar



- El texto da detalles. El Diagrama resume visualmente la interacción

DSS en el Proceso

- Se dibujan sólo los DSS correspondientes a los escenarios elegidos para la iteración
- Los DSS son parte del Modelo de Casos de Uso, visualizan las interacciones en un escenario
- Los DSS no se mencionan en la Descripción original de UP, son una propuesta de esta configuración de RUP
- DSS y Fases UP
  - No se recomienda su construcción durante la Fase de Inicio
  - La mayoría de los DSS se crean durante la Elaboración

De los requerimientos al Diseño

- En el desarrollo iterativo, en cada iteración tiene lugar una transición desde el enfoque centrado en los requerimientos al enfoque centrado en el Diseño y la implementación
- Durante el diseño se modela la estática y dinámica del sistema.
  - Los Modelos dinámicos (Diagramas de Interacción de UML) ayudan a diseñar la lógica y el comportamiento del código (cuerpo de los métodos)
  - Los Modelos estáticos (Diagrama de clases UML) ayudan a diseñar la estructura.
- En el desarrollo ágil se construyen en paralelo los modelos dinámicos y estáticos del sistema (Se realizan conjuntamente Diagramas de Interacciones y de clases)

RUP Agil. Dónde estamos???

Fase de Elaboración. Iteración I

Del Análisis al Diseño de Objetos
Asignación de responsabilidades
Patrones GRASP

Diseño de Objetos

- Para lograr un buen diseño de Objetos se necesita profundizar el conocimiento de:
  - Asignación de responsabilidades
  - Patrones de Diseño

Asignación de responsabilidades I
Realización de Casos de Uso

- *Pasar del Análisis al Diseño >>>>>*
- Partiendo del Modelo de clases del Análisis
- **Realizar los casos de uso Enfoque Tradicional (RUP) :** Recorrer los casos de uso, chequeando si son soportados por los objetos del diagrama, completando clases, atributos y asociaciones a medida que se "realizan" los casos de uso, modelando las interacciones en cada escenario.
- En esta opción se realiza un Diagrama de Interacciones para cada escenario de cada Caso de uso
- Se agregan las operaciones al Diagrama de clases, obteniendo una primera versión del Diagrama de Clases del Diseño.

Asignación de responsabilidades

- *Pasar del Análisis al Diseño >>>>>*
- Partiendo del Modelo de clases del Análisis
- **Realizar los casos de uso Enfoque RUP Agil:** Recorrer los casos de uso, chequeando si son soportados por los objetos del diagrama, completando clases, atributos y asociaciones a medida que se "realizan" los casos de uso, modelando las interacciones en cada escenario.
- En esta opción se realiza un Diagrama de Interacciones para cada operación de cada Caso de uso definida en los contratos, utilizando Patrones Grasp
- Se agregan las operaciones al Diagrama de clases, obteniendo una primera versión del Modelo de Diseño.

Asignanción de responsabilidades II

- *Pasar del Análisis al Diseño >>>>>*
- Partiendo del Modelo de clases del Análisis
- Opción 2 (RUP Agil) Realizar los casos de uso: Recorrer los casos de uso, chequeando si son soportados por los objetos del diagrama, completando clases, atributos y asociaciones a medida que se "realizan" los casos de uso, modelando las interacciones en cada escenario.
- En esta opción se realiza un Diagrama de Interacciones para cada operación de cada Caso de uso definida en los contratos, utilizando Patrones Grasp
- Se agregan las operaciones al Diagrama de clases, obteniendo una primera versión del Modelo de Diseño.

Diseño y Patrones

- Recordar que "*un sistema orientado a objetos es un conjunto de objetos que cooperan entre sí mediante el paso de mensajes para realizar las operaciones del sistema*"
- En el diseño, el esfuerzo se centra principalmente en los diagramas de interacción, mientras que el diagrama de clases de diseño resume las decisiones adoptadas.
  - Distribuir las responsabilidades y definir interacciones es la parte más difícil del diseño OO. Consume la mayor parte del tiempo.

- Para la asignación de responsabilidades es posible basarse en un conjunto de patrones de diseño

Patrones GRASP

- GRASP son un conjunto de patrones sencillos que describen principios fundamentales de asignación de responsabilidades a objetos.

- Los patrones describen un modo común de modelar una solución a un problema recurrente de diseño.
- Un patrón es un par **problema/solución** con nombre y con consejos de cómo aplicarlo según la situación.
- El asignar un nombre al patrón facilita la comunicación y la discusión entre diseñadores.
- Existen diversos patrones. Nos concentraremos en GRASP

Patrones GRASP

- Patrones GRASP: Describen los principios básicos de asignación de responsabilidades a clases
- GRASP es acrónimo de General Responsibility Assignment Software Patterns (Patrones Generales de Software para asignar responsabilidades)

- Los Patrones GRASP son:

Experto
Bajo Acoplamiento
Controlador
Indirección
Creador
Alta Cohesión
Polimorfismo
Variaciones Protegidas

Responsabilidades y métodos

- Dos categorías:

- **Conocer**
  - Datos encapsulados privados
  - Existencia de objetos conectados
  - Datos derivados o calculados

- **Hacer**
  - Algo él mismo
  - Iniciar una acción en otros objetos
  - Controlar y coordinar actividades en otros objetos

Patrón Experto (en Información)

- Problema: ¿Cuál es el principio más básico por el cual las responsabilidades son asignadas a objetos en el DOO?

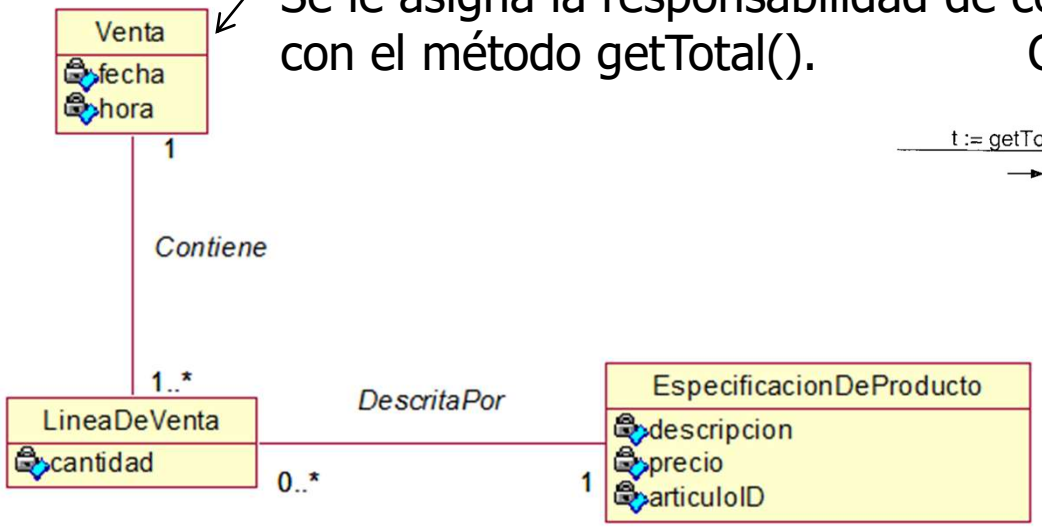
- Solución: Asignar una responsabilidad al experto en el tema. A la clase que tiene la información necesaria para llevar a cabo la responsabilidad

  - Heurísticas relacionadas
    - Distribuir responsabilidades de forma homogénea
    - No crear clases "dios"
  - Beneficios
    - Se conserva encapsulación: Bajo acoplamiento
    - Alta Cohesión: clases más ligeras

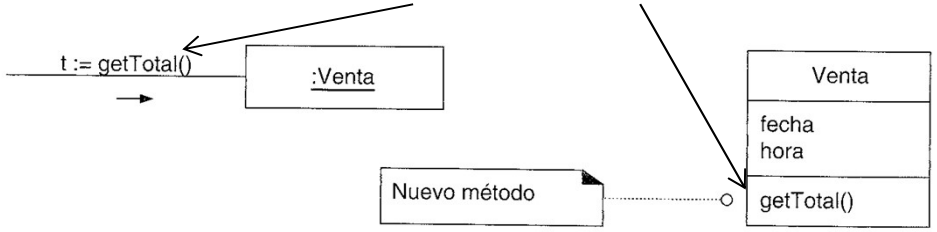
Patrón Experto (en Información)

- Ejemplo TPV: Algunas clases necesitan conocer el total de una venta. ¿A quién se le asigna la responsabilidad de calcular el total de una venta?
- Siguiendo el Experto se deberían buscar las clases de Objetos que tienen la información necesaria para calcular el total.
- Según el modelo conceptual (de dominio o análisis)

Seguramente la Clase Venta contiene la información de una Venta. Se le asigna la responsabilidad de conocer su total representada con el método getTotal().



Cuando se analicen interacciones:

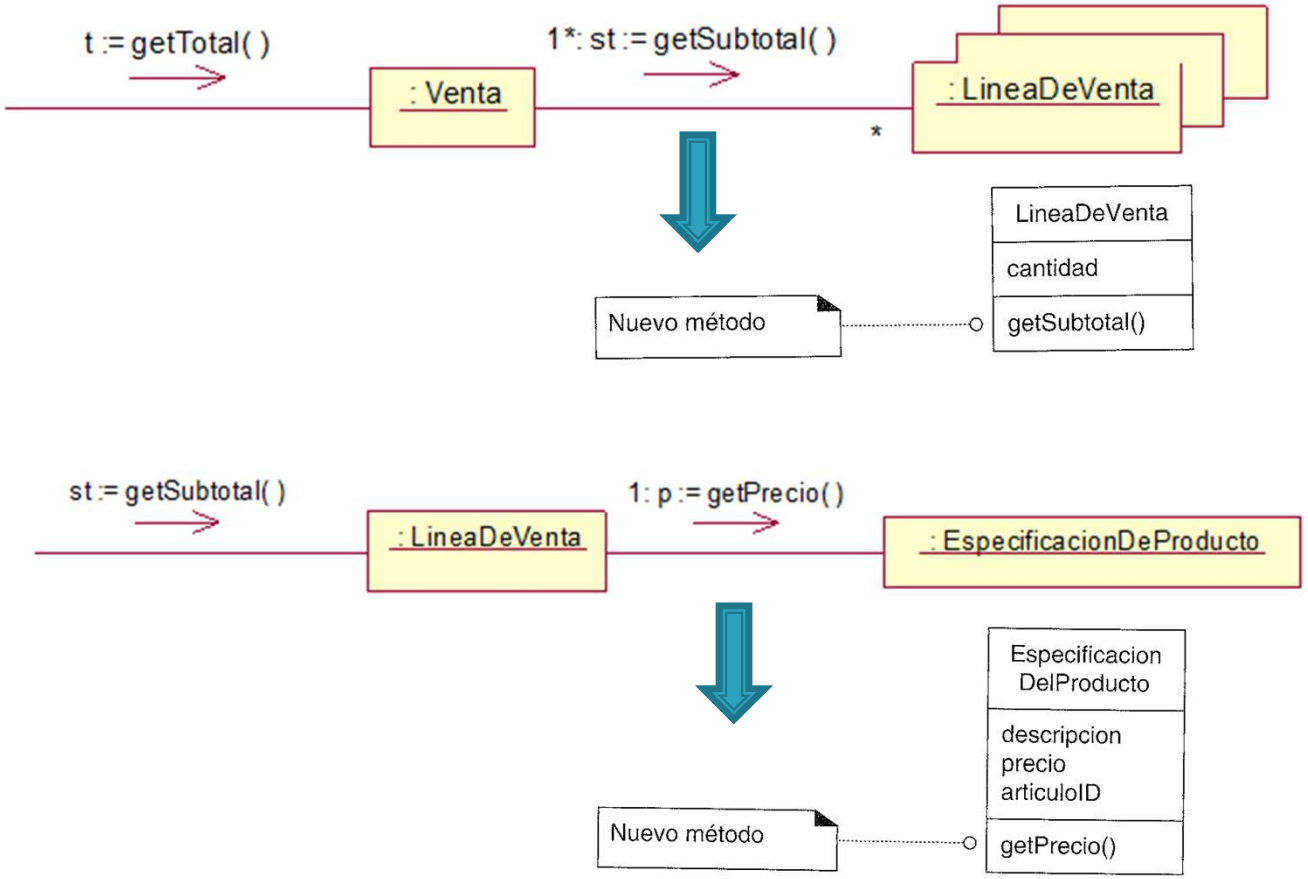


Patrón Experto (en Información)

- Ejemplo TPV (Continuación)
- Qué información se necesita para determinar el Total?
- Es necesario conocer la suma de subtotales de todas las instancias de LíneaDeVenta de una venta. Siguiendo el Experto, una instancia de Venta contiene las instancias, por lo tanto es responsable de ese trabajo.
- Para determinar el total entonces se necesita conocer
  - ✓ todas las líneas de venta que componen la venta (lo conoce la Venta)
  - ✓ el tipo de producto a que se refiere una línea de venta (lo conoce la LineaDeVenta)
  - ✓ el número de unidades del producto incluidas en la línea de venta (lo conoce la LineaDeVenta)
  - ✓ el precio del tipo de producto a que se refiere la línea de venta (lo conoce la EspecificacionDeProducto)

Patrón Experto (en Información)

- Ejemplo TPV (Continuación)
- Qué información se necesita para determinar el Total?



Patrón Experto (en Información)

- Ejemplo TPV (Continuación)
- En conclusión, para realizar la responsabilidad de conocer y proporcionar el total de una venta se asignaron 3 responsabilidades a 3 clases de Diseño

Clase de Diseño	Responsabilidad
Venta	Conocer el total de la venta
LíneaDeVenta	Conocer el subTotal de la línea de venta
EspecificacionDelProducto	Conocer el precio del artículo

- Según el Experto en Información, se colocó cada responsabilidad en el objeto que tiene la información necesaria para realizarla

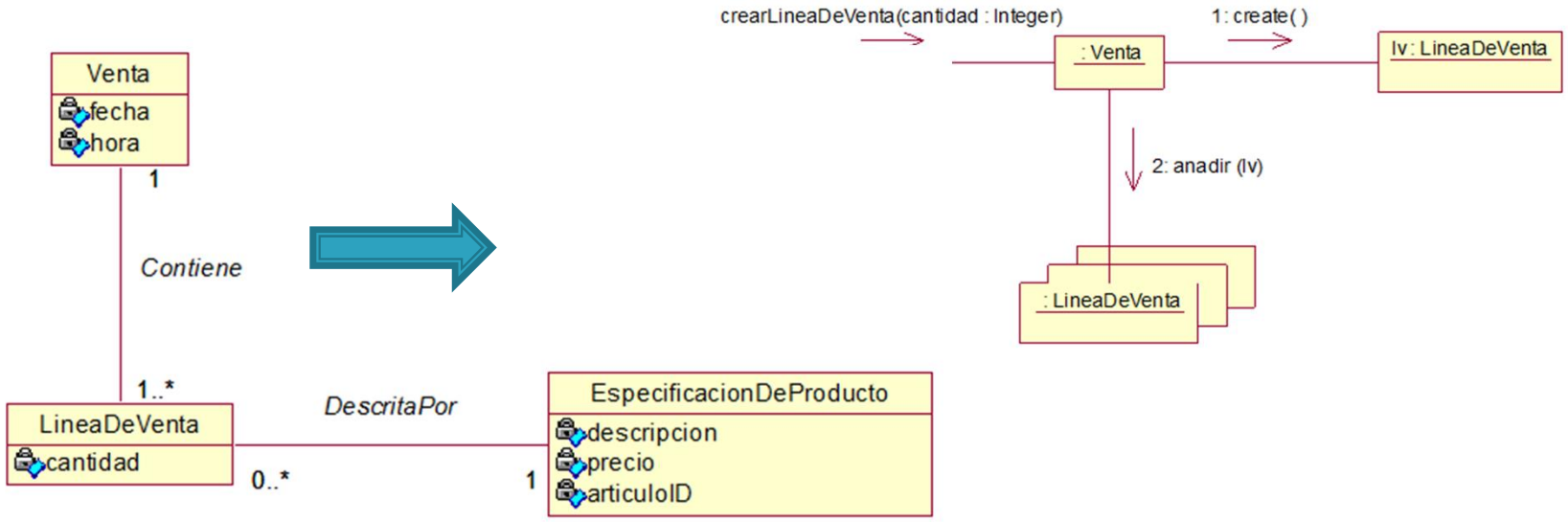
Patrón Creador

- Problema: ¿Quién es responsable de la creación de una nueva instancia de una clase?

- Solución: Asignar a la clase B la responsabilidad de crear una instancia de la clase A si sucede alguna de las siguientes situaciones:
  - B agrega objetos A
  - B contiene objetos A
  - B registra o lleva el control de las instancias que existen de A
  - B usa en gran medida objetos A
  - B tiene los datos necesarios para inicializar A (esto sería una aplicación del patron Experto)
- Si hay varias alternativas, es preferible seleccionar aquella clase que agregue objetos de la otra clase.

Patrón Creador

- Ejemplo TPV: ¿A quién se asigna la responsabilidad de crear una línea de venta?
- Según el modelo del dominio, la Venta agrega varios objetos de la clase LineaDeVenta, por lo que es un buen candidato para crearlas.



Patrón Bajo Acoplamiento

- Qué es el Acoplamiento:
- Es una medida del grado en que una clase está conectada a, sabe de o depende de otras clases.
- Desventajas de tener una clase acoplada con otras:
  - si cambia una clase relacionada puede que tengamos que cambiar la clase actual
  - es difícil de comprender de forma aislada
  - es difícil de reutilizar porque requiere llevar consigo las clases de las que depende
- Un acoplamiento muy bajo tampoco es deseable. Va en contra de la idea de resolver problemas mediante cooperación, inherente a la orientación a objetos.

Patrón Bajo Acoplamiento

- Formas de Acoplamiento:
  - una clase tiene un atributo cuyo valor es una instancia de otra clase o un puntero a ella
  - una clase tiene un método que hace referencia a una instancia de otra clase o a otra clase (parámetro o variable local de dicho tipo, o valor de retorno de un mensaje de dicho tipo)
  - una clase es subclase directa o indirecta de otra clase. Es un tipo de acoplamiento muy fuerte.
  - una clase es una interfaz y la otra clase implementa dicha interfaz

Patrón Bajo Acoplamiento

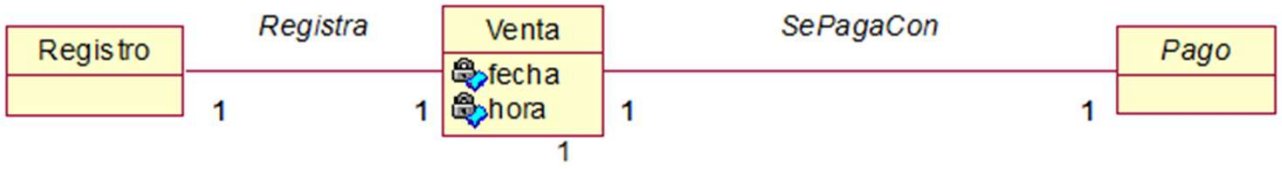
- Problema: ¿Cómo conseguir menos dependencia entre objetos y una mayor capacidad de reutilización?

- Solución: Asignar una responsabilidad de modo que el acoplamiento permanezca bajo

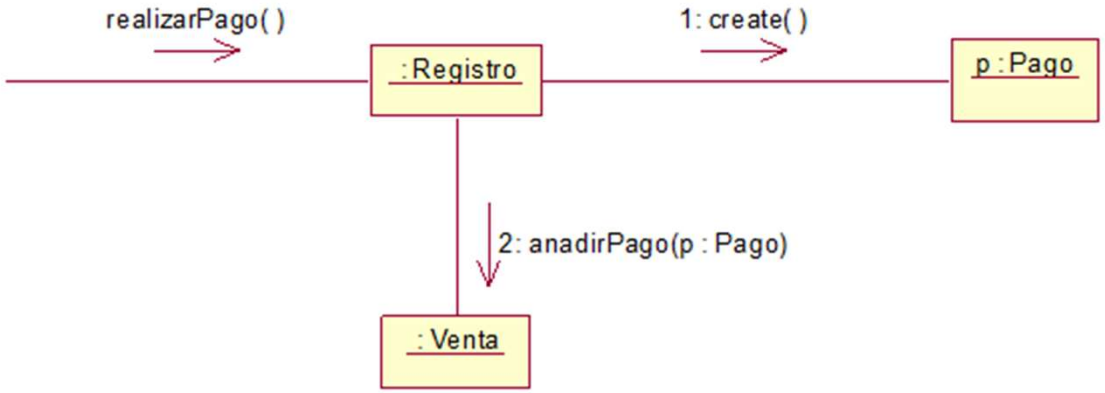
- Este es un patrón de tipo Evaluativo, es decir, es un principio a tener en mente en todas las decisiones de diseño, pero no se puede considerar de manera aislada a otros patrones.
- A veces no es un problema el tener un acoplamiento alto si éste se produce con clases estables y generalizadas

Patrón Bajo Acoplamiento

- Ejemplo TPV: ¿A quién se le asigna la responsabilidad de crear un Pago?. Según el modelo del dominio:

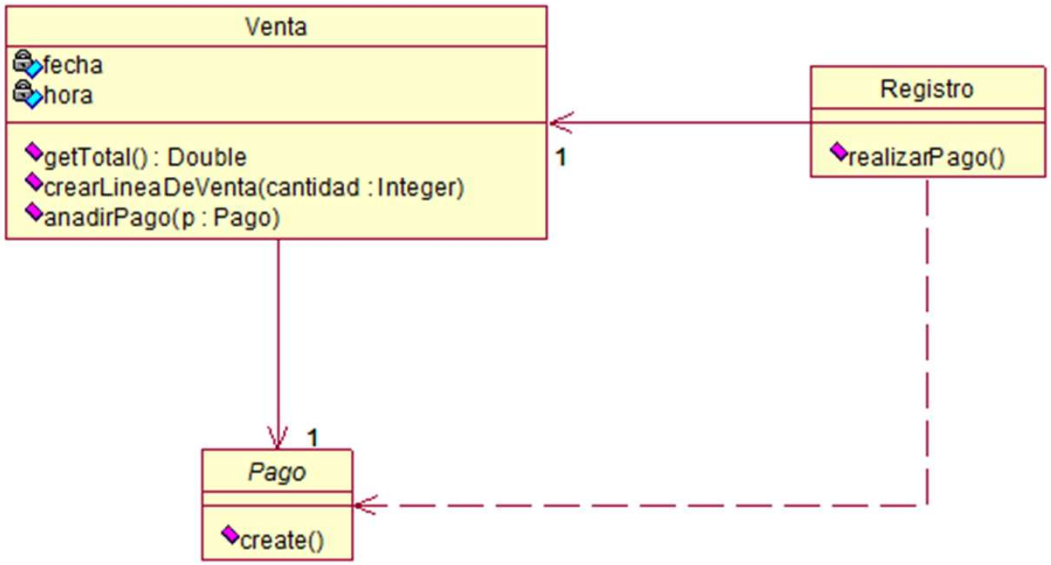


- En el mundo real es el Registro el que registra el pago. Podría entonces ser esta clase la que cree el pago y se lo pase a la Venta para que se asocie con él



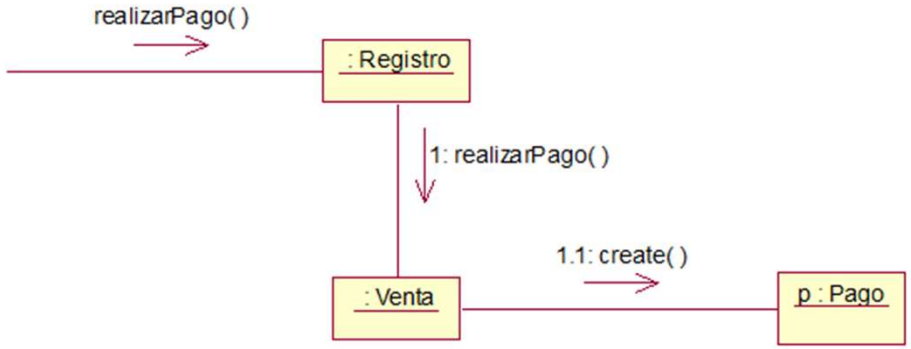
Patrón Bajo Acoplamiento

- Pero esta solución genera un acoplamiento de la clase Registro respecto a la clase Pago

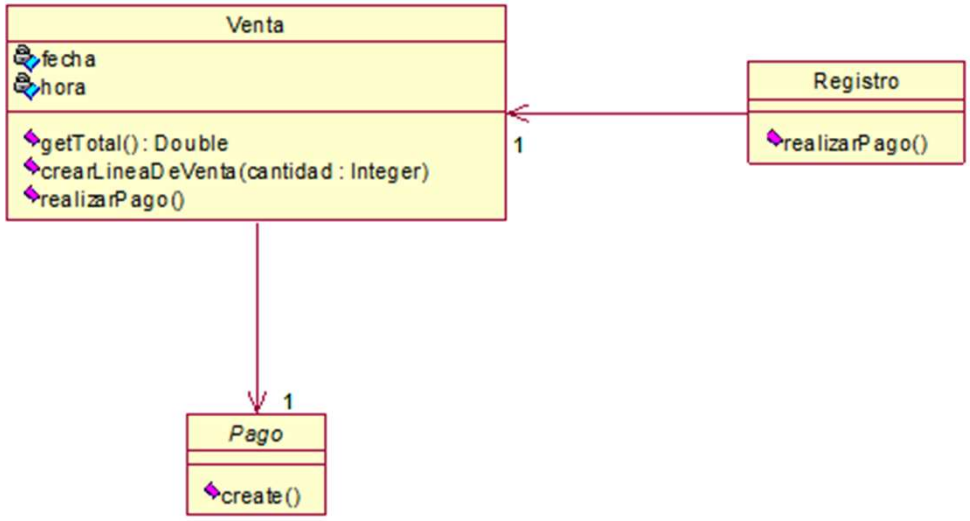


Patrón Bajo Acoplamiento

- Para reducir el acoplamiento sería preferible que sea la propia clase Venta la que cree el Pago



- De esa forma se eliminó la dependencia de Registro respecto de Pago



Patrón Alta Cohesión

- ¿Qué es la Cohesión?
  - Es una medida del grado de relación entre sí y coherencia que tienen las responsabilidades asignadas a una clase.
  - Una clase con poca cohesión hace muchas cosas que no tienen nada que ver unas con otras.
  - Suele ser consecuencia de no haber delegado lo suficiente en otras clases.
- Desventajas de la baja cohesión:
  - la clase es difícil de comprender
  - es difícil de reutilizar
  - es difícil de mantener
  - es delicada. Se verá afectada por los cambios constantemente

Patrón Alta Cohesión

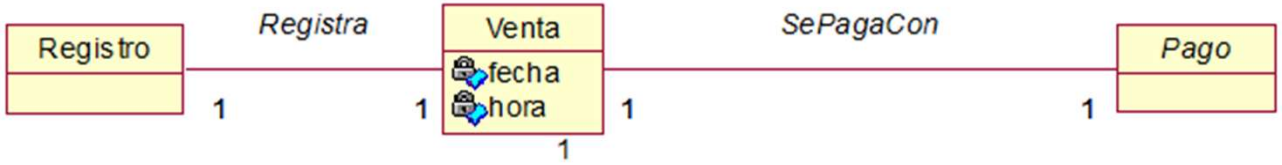
- Problema: ¿Cómo mantener la complejidad del sistema en un nivel manejable?

- Solución: Asignar una responsabilidad de tal manera que la cohesión permanezca alta.

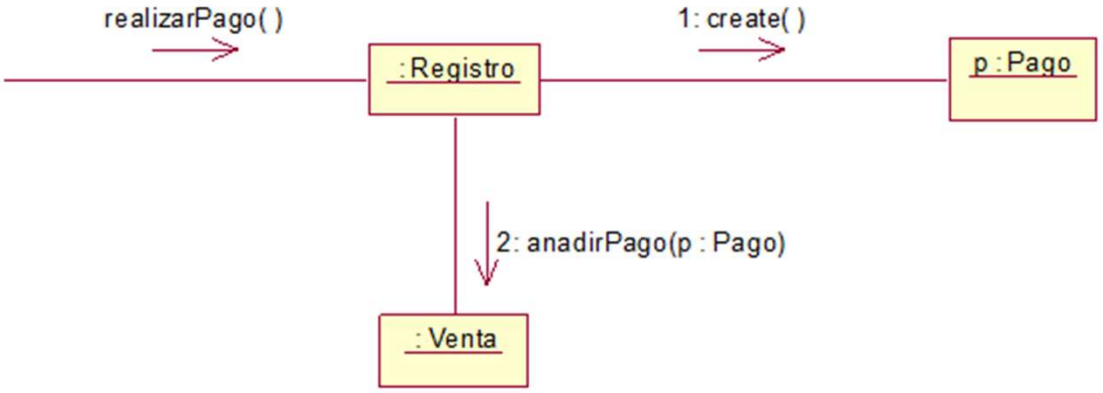
- Este es un patrón de tipo Evaluativo, es decir, es un principio a tener en mente en todas las decisiones de diseño, pero no se puede considerar de manera aislada a otros patrones.
- Beneficios: Se facilita la claridad y comprensión del diseño, se simplifican el mantenimiento y las mejoras, y facilita la reutilización, ya que es poco probable que una clase que hace muchas cosas diferentes se pueda reutilizar.

Patrón Alta Cohesión

- Ejemplo TPV: ¿A quién se le asigna la responsabilidad de crear un Pago?. Según el modelo del dominio:

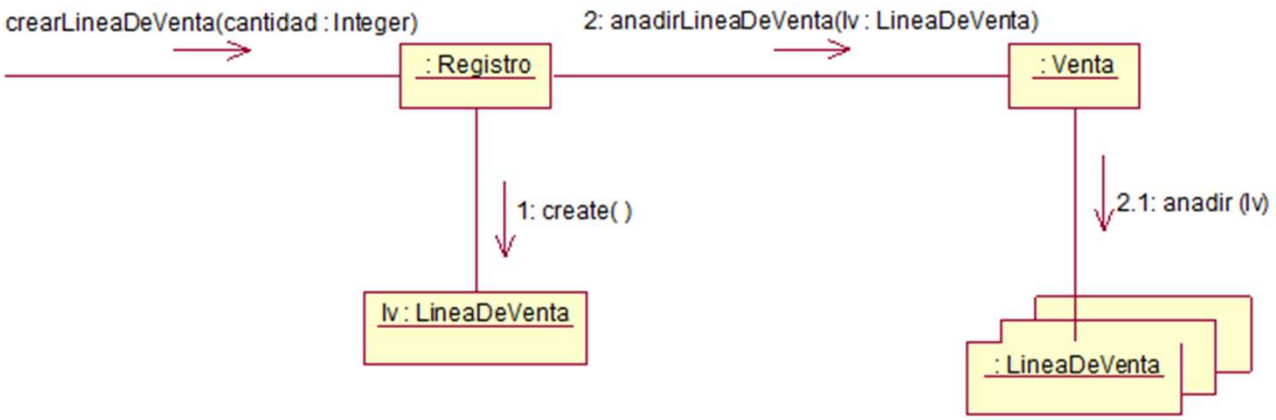


- En el mundo real es el Registro el que registra el pago. Podría entonces ser esta clase la que cree el pago y se lo pase a la Venta para que se asocie con él:



Patrón Alta Cohesión

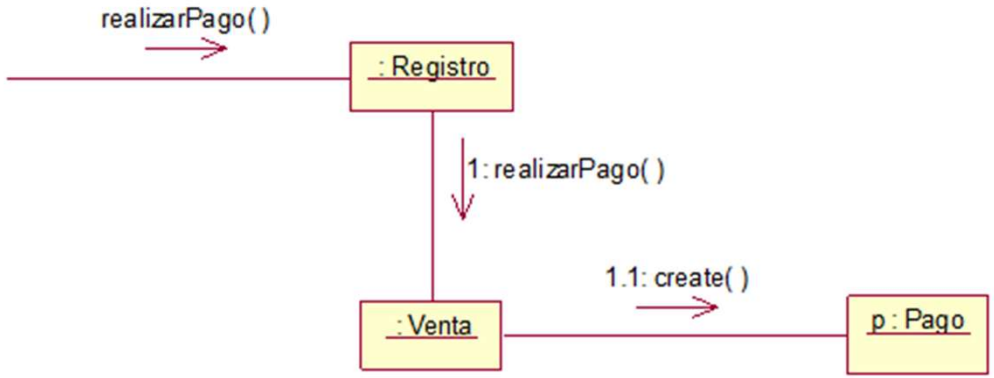
- Si continuamos asignando responsabilidades a la clase Registro, puede sobrecargarse la clase y perder la cohesión.
- Por ejemplo, podríamos asignarle también la responsabilidad de crear las líneas de venta:



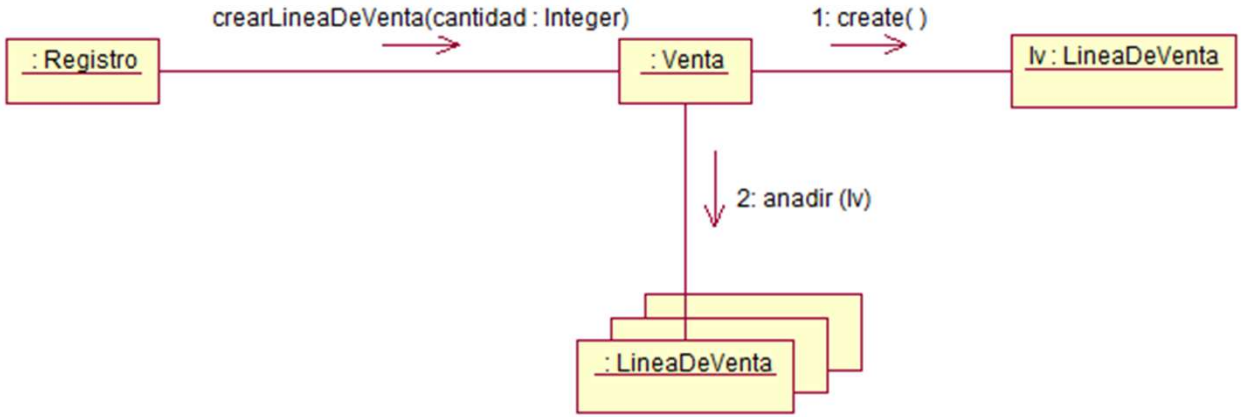
- Por ello es preferible que el Registro delegue en la clase Vanta la responsablidad de crear pagos y lineasDeVenta

Patrón Alta Cohesión

- Creación de Pagos



- Creación de Líneas de Venta



Patrón Controlador

- Para cada uno de los eventos del sistema generados por los actores, se hace necesario decidir qué objeto va a ser el primero en recibirlo (es decir, quién va a ofrecer la operación del sistema correspondiente).

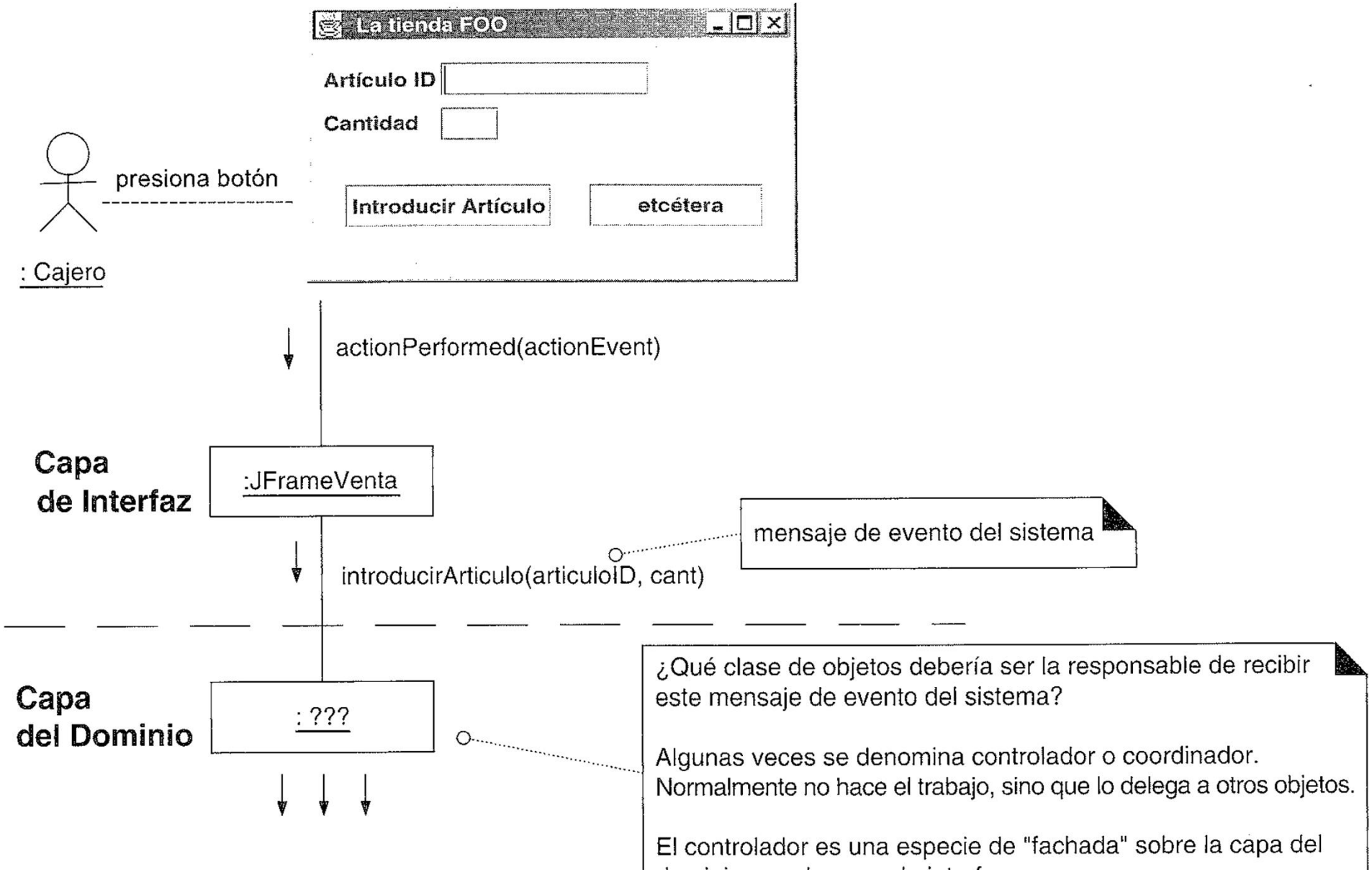
- Problema: ¿Quién debería ser el responsable de tratar un evento del sistema?

- Solución: Asignar la responsabilidad de tratar un evento del sistema a una clase representando:
  - El sistema completo, dispositivo o subsistema (*Controlador de Fachada*), o bien
  - Un manejador encargado de todos los eventos de un caso de uso (*Controlador de sesión o de caso de uso*)

Patrón Controlador

- ¿Quién puede ser un Controlador?
- Clases que no tienen nada que ver con la interfaz de usuario. Los objetos de interfaz (ventanas, frames, etc.) no deben ser responsables de llevar a cabo los eventos del sistema, sino que los mismos se manejan en el nivel de la lógica de la aplicación o del dominio.
  - Esto permite independizar el nivel de Presentación del nivel de Negocio (patrón Layers o Capas)
  - Facilita cambiar la interfaz sin afectar a la lógica de negocio
  - Facilita la reutilización de la lógica de negocio
- El controlador no debe acumular demasiadas responsabilidades (siguiendo el patrón Alta Cohesión)

Controlador para introducirArticulo?

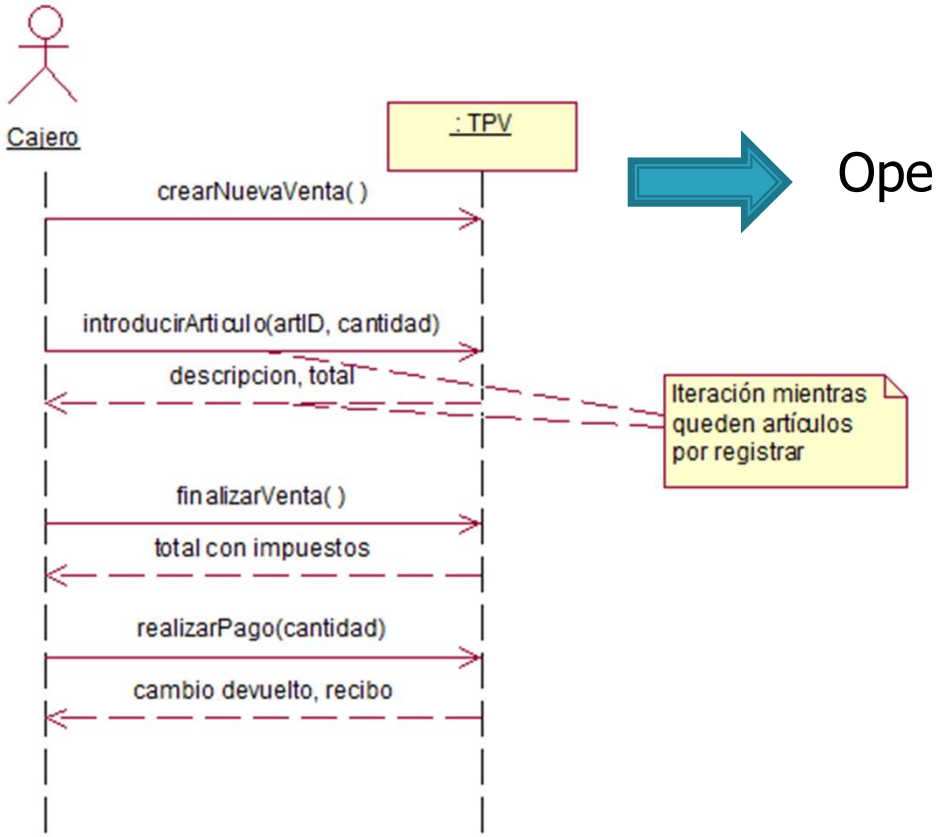


Patrón Controlador

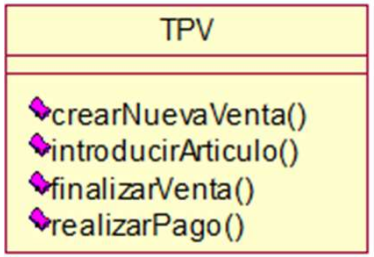
- ¿Cuántos controladores?
- Se puede tener un controlador único por sistema sólo si no hay muchas operaciones del sistema.
- Un controlador para un caso de uso tiene la ventaja adicional de que permite mantener información sobre el estado del caso de uso y detectar eventos fuera de secuencia

Patrón Controlador

- Ejemplo de TPV: ¿Quién va a tratar los eventos del sistema?
- Vemos un diagrama de secuencia del sistema para el caso de uso Procesar Venta y las operaciones generadas

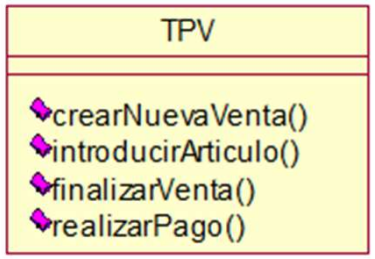


Operaciones del sistema para el escenario

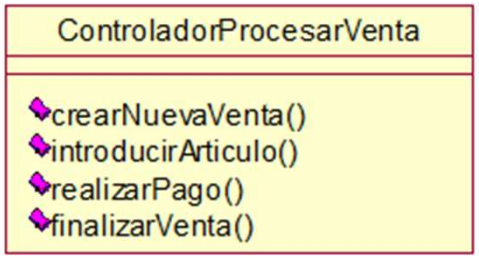


Patrón Controlador

- Solución 1: Controlador de Fachada
- Puede ser una clase que represente el sistema completo TPV (El sistema Tienda Punto de Venta) o la clase Registro

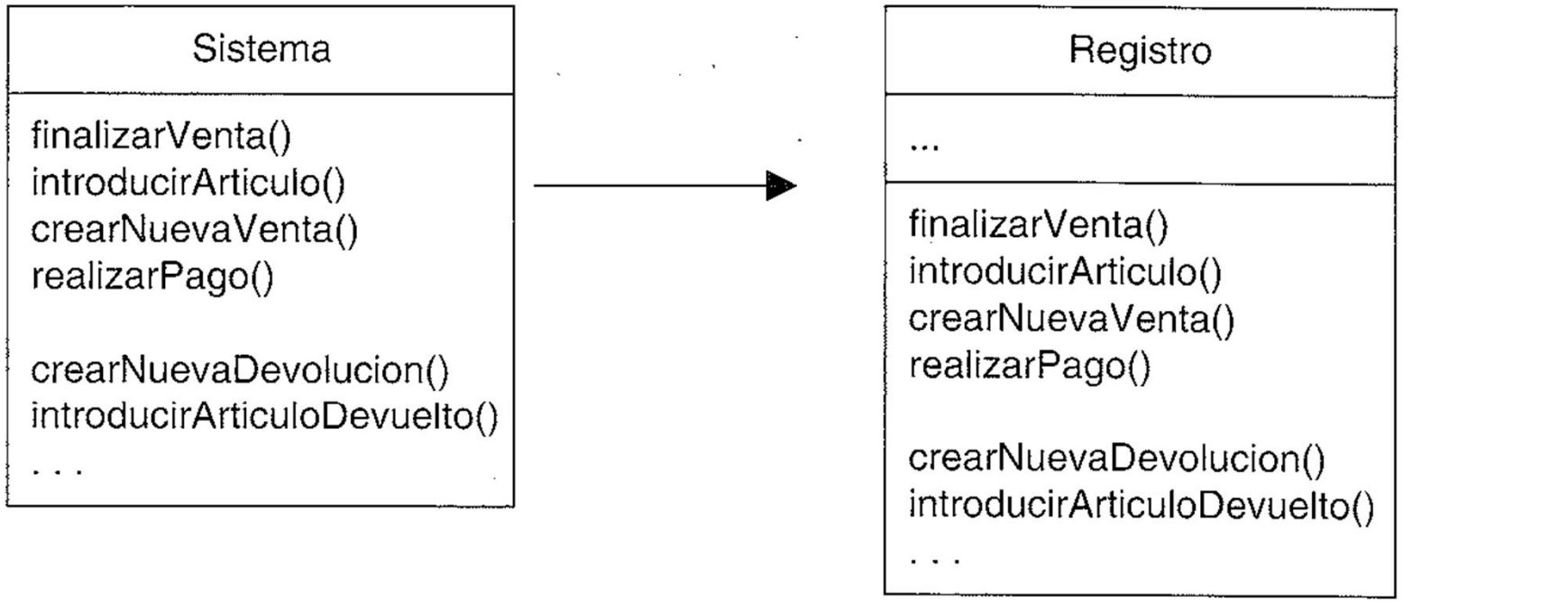


- Solución 2: Controlador de Caso de Uso
- Se puede crear una clase ControladorProcesarVenta, que reciba todos los eventos de este caso de uso:



- La diferencia es que Registro acumularía otras responsabilidades procedentes de otros casos de uso.

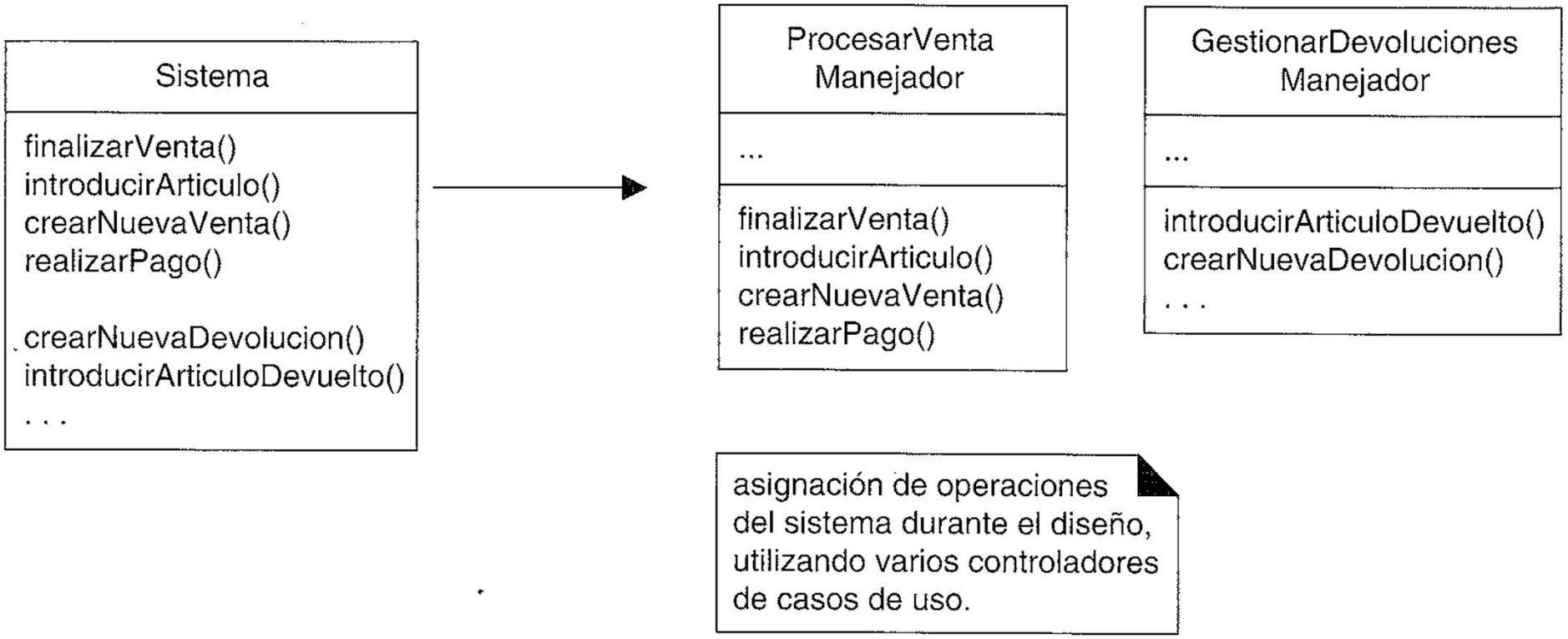
Patrón Controlador



Operaciones del sistema descubiertas durante el análisis del comportamiento del sistema.

Asignación de operaciones del sistema durante el diseño, utilizando un controlador de fachada.

Patrón Controlador



Referencias

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