Alumnos

Laguna Montes jose Israel
Mendoza pona Diego
Nina Layme Ronald
Valdez Diaz Luis

PATRÓNES DE DISEÑO

BAJO ACOPLAMIENTO Y ALTA COHESION

¿Qué es un Patrón?

En la tecnología de objetos un **Patrón** es una descripción de un problema y la solución, a la que se le da un nombre, y que se puede aplicar a nuevos contextos.

Un patrón intenta codificar el conocimiento, expresiones y los principios existentes.

Un patrón suele incluir:

- Descripción
- Escenario de Uso
- Solución concreta
- Las consecuencias de utilizar este patrón
- Ejemplos de implementación
- Lista de patrones relacionados

Patrón de diseño

Los **patrones de diseño** son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces.

Un patrón de diseño es una solución a un problema de diseño. Para que una solución sea considerada un patrón debe poseer ciertas características. Una de ellas es que debe haber comprobado su **efectividad** resolviendo problemas similares en ocasiones anteriores. Otra es que debe ser **reusable**, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias.

Objetivos de los patrones

Los patrones de diseño pretenden:

- Proporcionar catálogos de elementos reusables en el diseño de sistemas software.
- Evitar la reiteración en la búsqueda de soluciones a problemas ya conocidos y solucionados anteriormente.

- Formalizar un vocabulario común entre diseñadores.
- Estandarizar el modo en que se realiza el diseño.
- Facilitar el aprendizaje de las nuevas generaciones de diseñadores condensando conocimiento ya existente.

Asimismo, no pretenden:

- Imponer ciertas alternativas de diseño frente a otras.
- Eliminar la creatividad inherente al proceso de diseño.

No es obligatorio utilizar los patrones, solo es aconsejable en el caso de tener el mismo problema o similar que soluciona el patrón, siempre teniendo en cuenta que en un caso particular puede no ser aplicable.

Abusar o forzar el uso de los patrones puede ser un error.

PATRONES GRASP.

- Acrónimo de General Responsability Assignment Software Patterns (Patrones de Software para la asignación General de Responsabilidad).
- Describen los principios fundamentales de diseño de objetos para la asignación de responsabilidades.

Se pueden destacar 5 patrones Principales que son:

- Experto.
- Creador.
- Alta cohesión.
- Bajo acoplamiento.
- Controlador.

DEPENDENCIAS

Para el empleo de patrones es necesario realizar previamente:

- Diagrama de Colaboración.

ALTA COHESIÓN Y BAJO ACOPLAMIENTO

Los conceptos de cohesión y acoplamiento están íntimamente relacionados. Un mayor grado de cohesión implica uno menor de acoplamiento. Maximizar el nivel de cohesión intramodular en todo el sistema resulta en una minimización del acoplamiento intermodular.

Los podemos separar, aunque están íntimamente ligados, de hecho si nos esforzamos en aumentar mucho la cohesión de nuestro sistema software, es muy posible que

perjudiquemos el acoplamiento aumentándolo, y por el contrario si reducimos mucho el acoplamiento, se verá disminuida la cohesión:

BAJO ACOPLAMIENTO

Este patrón es un principio que asigna la **responsabilidad** de controlar el flujo de eventos del sistema, a clases específicas. Esto facilita la centralización de actividades. El controlador no realiza estas actividades, las delega en otras clases con las que mantiene un modelo de alta cohesión. Un error muy común es asignarle demasiada responsabilidad y alto nivel de acoplamiento con el resto de los componentes del sistema.

Acoplamiento bajo significa que una clase no depende de muchas clases. Por lo que debe haber pocas dependencias entre las clases. Si todas las clases dependen de todas ¿cuánto software podemos extraer de un modo independiente y reutilizarlo en otro proyecto? Para determinar el nivel de acoplamiento de clases, son muy buenos los diagramas de colaboración de UML. Uno de los principales síntomas de un mal diseño y alto acoplamiento es una herencia muy profunda. Siempre hay que considerar las ventajas de la delegación respecto de la herencia**.**

El acoplamiento tal vez no sea tan importante, sino se busca la reutilización. Para apoyar una mejor reutilización de los componentes al hacerlo más independientes, el entero contexto de las metas de rehúso ha de tenerse en cuenta antes de intentar reducir al mínimo el acoplamiento.

En qué consiste

Es la idea de tener las clases lo menos ligadas entre sí que se pueda. De tal forma que en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión posible en el resto de clases, potenciando la reutilización, y disminuyendo la dependencia entre las clases

Se clasifica de la siguiente manera:

Acoplamiento de Contenido: Cuando un módulo referencia directamente el contenido de otro módulo. (En lenguajes de alto nivel es muy raro)

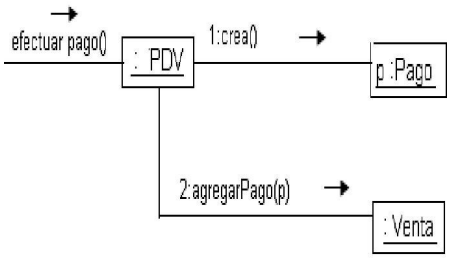
Acoplamiento Común: Cuando dos módulos acceden (y afectan) a un mismo valor global.

Acoplamiento de Control: Cuando un módulo le envía a otro un elemento de control que determina la lógica de ejecución del mismo.

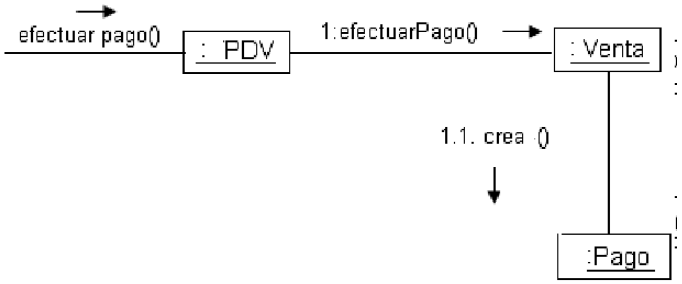
Ejemplo.-

En el caso del punto de ventas se tienen tres clases: Pago, TPDV y Venta y se quiere crear una instancia de Pago y asociarla a Venta. ¿Que clase es la responsable de realizarlo?

- Según el patrón experto la Clase TPDV deberá hacerlo



Según el patrón de Bajo Acoplamiento la relación debería ser de la siguiente manera:



Esta última asociación es mejor dado que Venta realiza la creación del pago y no TPDV por lo tanto se reduce la dependencia de este último con el resto de las clases.

Que representa

El acoplamiento de una clase representa el conjunto de dependencias que tiene con otras clases.

Así, una clase A que deriva de B, tendrá un fuerte acoplamiento con ella. Y otra clase C, que utilice una clase D, tendrá también una relación de dependencia.

Resulta evidente que, cuanto menor sea el acoplamiento entre clases, menor influencia tendrán los cambios.

Así, un cambio en una clase A, que depende de las clases B, C y D será mucho más complejo que si la misma clase A no dependiera de ninguna otra y, por lo tanto, el cambio sólo le afectara a ella.

Mantener Bajo el Acoplamiento entre Clases, más que un patrón que se pueda implementar, es un principio que nos servirá para elegir entre alternativas de diseño: un diseño menos acoplado siempre será mejor que uno más acoplado, porque en el primero será más fácil realizar cambios.

Beneficios

- No afecta los cambios en otros componentes.

- Fácil de entender de manera aislada.
- Conveniente para reutilizar.

ALTA COHESIÓN

La cohesión es una medida de cuán relacionadas y enfocadas están las responsabilidades de una clase.

Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme.

La alta **cohesión** es una medida con la que se relacionan las clases y el grado de responsabilidades de un elemento.

Ejemplos de una baja cohesión son clases que hacen demasiadas cosas.

Ejemplos de buen diseño se producen cuando se crean las clases agrupadas por funcionalidades que son reutilizables.

Algunos escenarios:

- Muy baja cohesión: Una clase es la única responsable de muchas cosas en áreas funcionales heterogéneas.
- Baja cohesión: Una clase tiene la responsabilidad exclusiva de una tarea compleja dentro de un área funcional.
- Alta cohesión: Una clase tiene responsabilidades moderadas en un área funcional y colabora con las otras para llevar a cabo las tareas.
- Cohesión moderada: Una clase tiene peso ligero y responsabilidades exclusivas en unas cuantas áreas que están relacionadas lógicamente con el concepto de clase pero no entre ella

Nos dice que la información que almacena una clase debe de ser coherente y debe estar (en la medida de lo posible) relacionada con la clase.

1. **Cohesión Coincidente**: El módulo realiza múltiples tareas, sin ninguna relación entre ellas.

2. **Cohesión Lógica**: El módulo realiza múltiples tareas relacionadas, pero, en tiempo de ejecución, sólo una de ellas será llevada a cabo.

3. **Cohesión Temporal**: Las tareas llevadas a cabo por un módulo tienen, como única relación: el ser ejecutadas "al mismo tiempo".

4. **Cohesión de Procedimiento**: La única relación que guardan las tareas de un módulo es que corresponden a una secuencia de pasos propia del "producto".

5. **Cohesión de Comunicación**: Las tareas corresponden a una secuencia de pasos propia del "producto" y todas afectan a los mismos datos.

6. **Cohesión de Información**: Las tareas llevadas a cabo por un módulo tienen su propio punto de arranque, su codificación independiente y trabajan sobre los mismos datos. El ejemplo típico: OBJETOS

VENTAJAS

Una clase cohesionada facilita el cambio (objetivo principal de los patrones de diseño). Al realizar un cambio en una clase muy cohesionada, todos los métodos que pueden verse afectados, toda la información que necesitamos controlar, estará a la vista, en el misma clase.

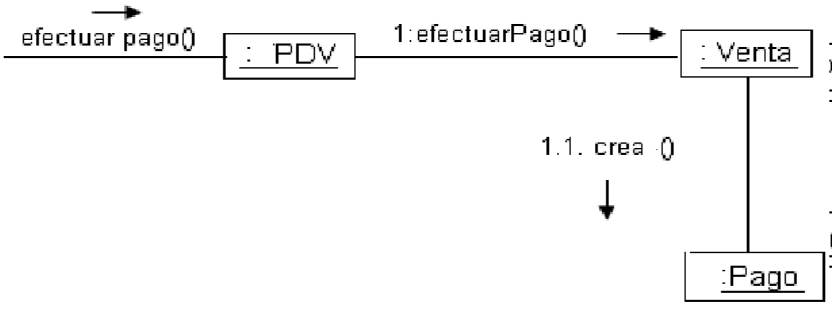
Este diseño es conveniente ya que da soporte a una alta cohesión y a un bajo acoplamiento.
En la práctica, el nivel de cohesión no puede ser considerado independiente de los otros patrones y principios (e.g. Patrones "Experto" y "Bajo Acoplamiento").

BENEFICIOS

* Mejoran la claridad y facilidad con que se entiende el diseño
* Se simplifica el mantenimiento y las mejoras de funcionalidad
* A menudo se genera un bajo acoplamiento
* Soporta mayor capacidad de reutilización.

Ejemplo

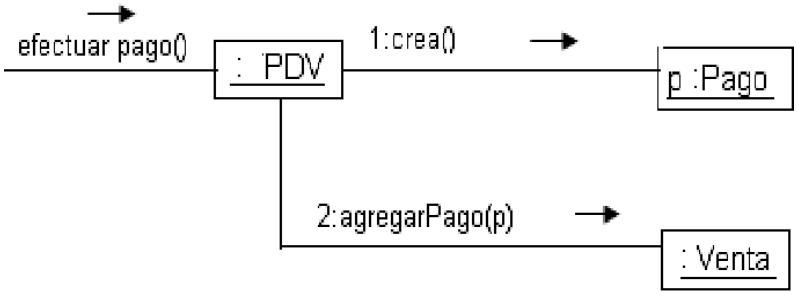
En el caso del punto de ventas se tienen tres clases Pago, TPDV y venta y se quiere crear una instancia de Pago y asociarla a Venta. Según el principio del patrón Creador la clase TPDV debe ser la encargada de realizar el pago.



¿Qué pasa si el sistema tiene 50 operaciones, todas recibidas por la clase TPDV ?

La clase se iría saturando con tareas y terminaría perdiendo la cohesión

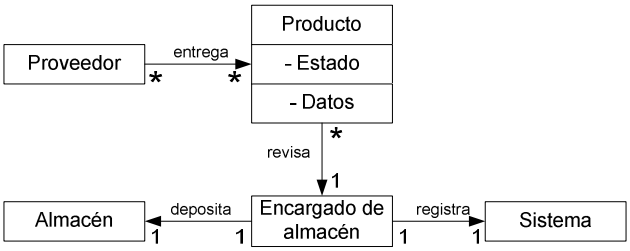
Un mejor diseño de lo anterior sería:



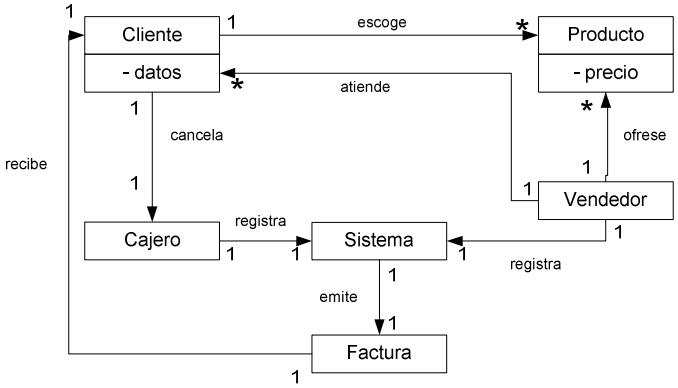
APLICACION DE LOS PATRONES EN EL PROYECTO

Modelo conceptual:

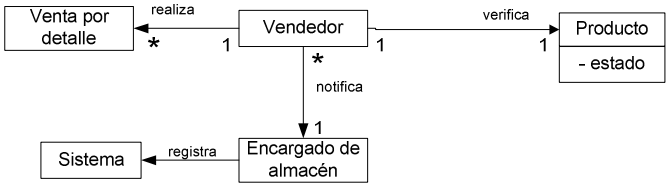
Caso de uso: Registro de Productos nuevos en Almacén



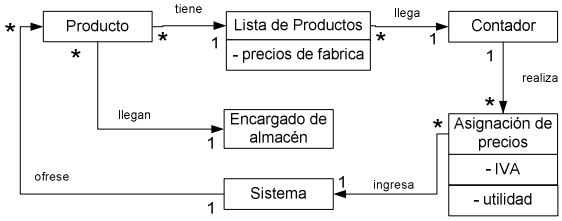
Casos de usos.- Registro de Ventas.



Caso de uso.- Retiro de productos defectuosos de almacén



Caso de uso.- Asignación de precios.



Caso de uso.- Registro de Usuario

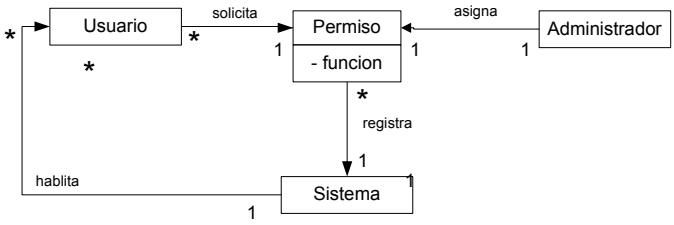


Diagrama de casos de Uso (un diagrama global):

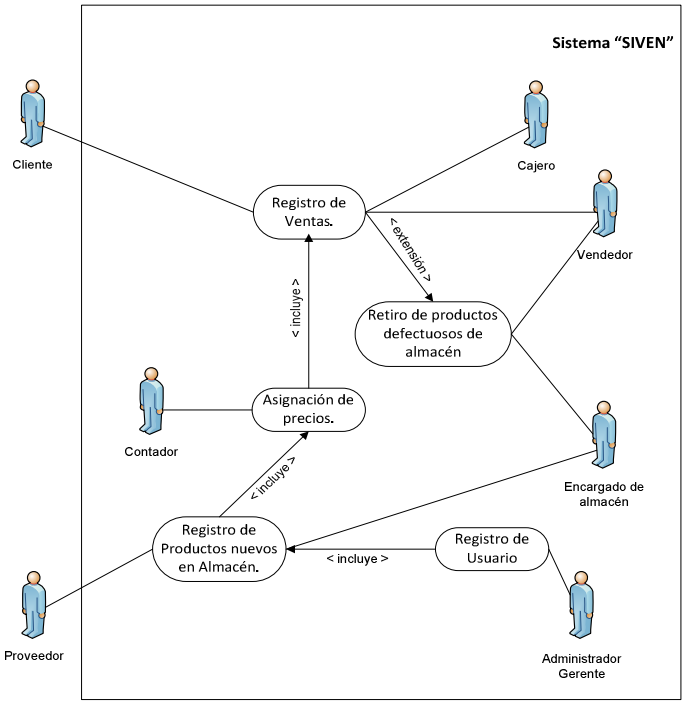
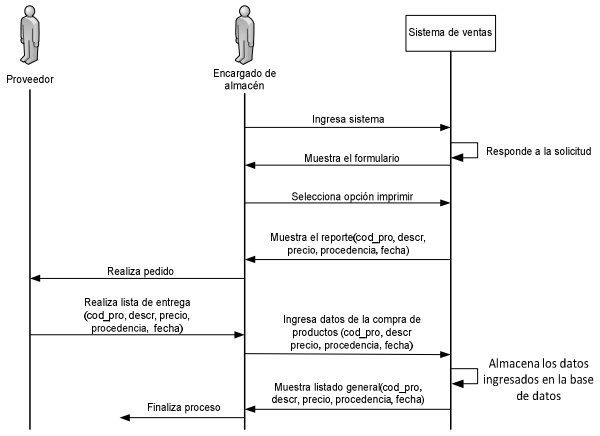
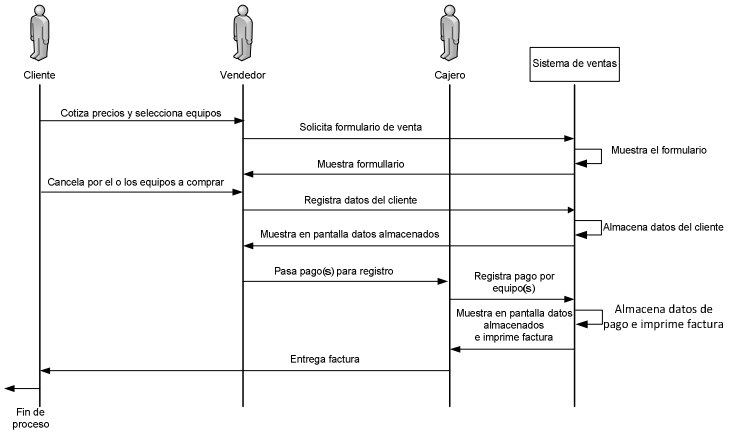


Diagrama de secuencias:

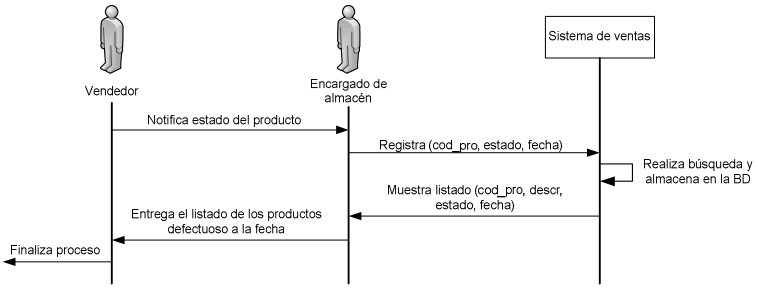
Registro de Productos nuevos en Almacén



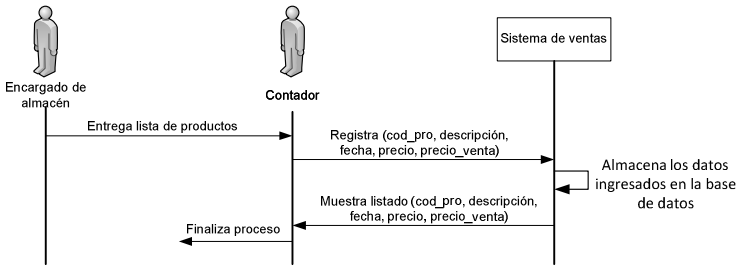
Registro de ventas



Retiro de productos defectuosos



Asignación de precios.



Registro de Usuario

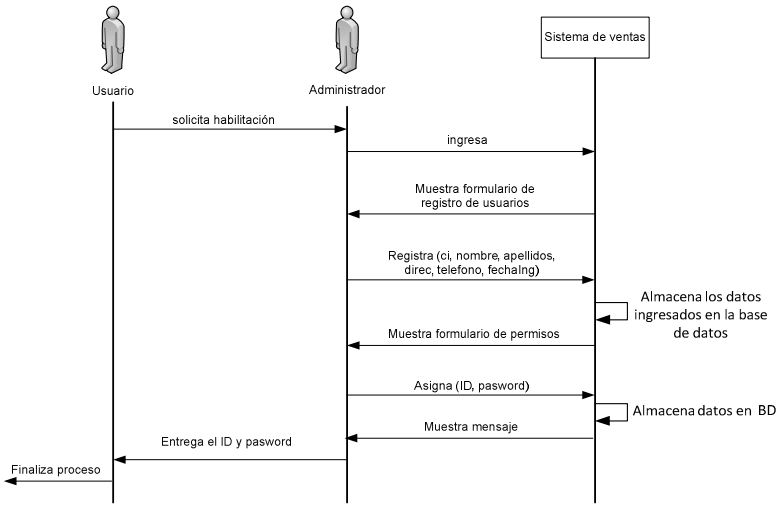
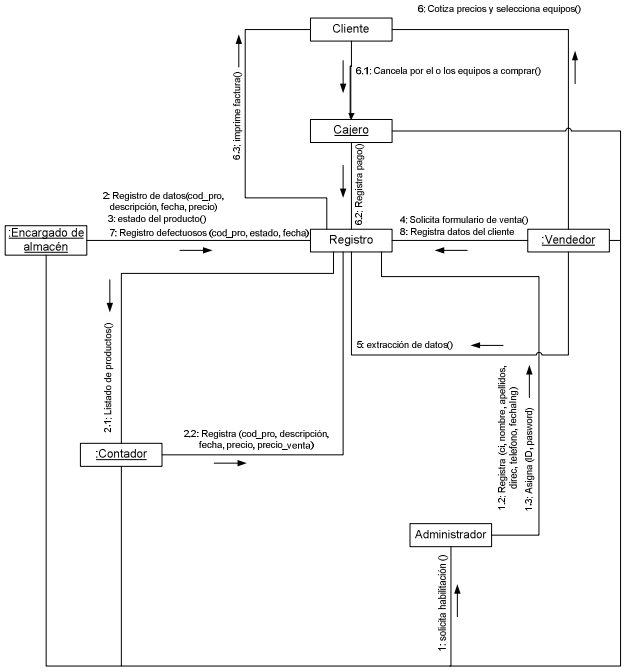
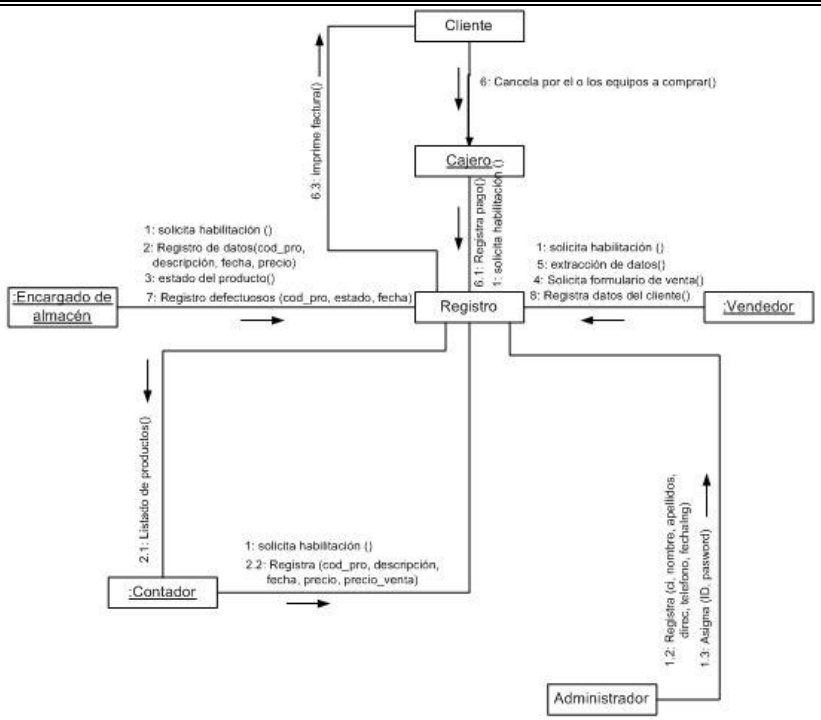


Diagrama de colaboración:



Patrones de diseño (Alta cohesión y bajo acoplamiento):



BIBLIOGRAFÍA

- http://es.wordpress.com/tag/patrones-grasp/
- http://www.adictosaltrabajo.com/tutoriales/pdfs/grasp.pdf
- http://pub.ufasta.edu.ar/fim28/files2005/FIM28%20AyD%20I%20Dominio%20y%20GRASP.pdf
- http://es.wikipedia.org/wiki/Grasp/
- http://migueljaque.com/
- Booch, G., Jacobson, I. y Rumbaugh, J. UML *Manual de referencia.* Addison Wesley. 1997.

ANEXOS (PREGUNTAS)

- **¿Porque debe usar Patrones de Diseño?**

- Son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces.

- **¿Qué diferencias existe entre alta cohesión y bajo acoplamiento?**

  - Alta cohesión, es una medida de cuán relacionadas y enfocadas están las responsabilidades de una clase.

  - Bajo acoplamiento, asigna la responsabilidad de controlar el flujo de eventos del sistema.

- **¿Que patrón es el que asigna responsabilidades de control del flujo de eventos?**

  - El patrón de Bajo acoplamiento.