xx (110523.1)

CINVESTAV

Centro de Investigación y de Estudios Avanzados del IPN
Unidad Guadalajara

ARQUITECTURA DE SOFTWARE PARA UN SISTEMA DE COMERCIO ELECTRÓNICO B2C

Tesis que presenta:
Jaime Arturo Cedillo Navarro

Para obtener el grado de:
MAESTRO EN CIENCIAS

En la especialidad de:
INGENIERÍA ELÉCTRICA

Guadalajara, Jal., Marzo del 2003

ARQUITECTURA DE SOFTWARE PARA UN SISTEMA DE COMERCIO ELECTRÓNICO B2C

Tesis de Maestría en Ciencias
Ingeniería Eléctrica

Por:

Jaime Arturo Cedillo Navarro

Licenciado en Informática
Universidad de Guadalajara, 1995-1999

Becario del CONACYT, expediente No. 143822

Director de Tesis:
Dr. Manuel Edgardo Guzmán Rentería

CINVESTAV del IPN Unidad Guadalajara, Marzo del 2003

AGRADECIMIENTOS:

A mis padres.

A mis hermanos.

A mi asesor.

A mis maestros.

A mis compañeros.

Al CINVESTAV.

Índice General

Lista de Figuras

Capítulo 1 **Introducción y Objetivos**

1.1 **Introducción**

Presentamos una panorámica de Internet y de WWW (World Wide Web), estas tecnologías han hecho posible el comercio electrónico como lo conocemos hoy y son fundamentales en esta tesis.

Internet evolucionó de la red militar ARPANet creada en la década de los sesenta cuya idea principal era crear una red que continuara trabajando aún si alguna de sus partes fuera dañada. Internet es una infraestructura de red que esta construida sobre ciertos estándares que son utilizados por los participantes para conectarse unos con otros. La especificación del protocolo Internet (Internet Protocol) no especifica que tipo de información, servicios o productos pueden ser intercambiados, sólo define como se organiza el flujo de información.

Para el intercambio de información se han creado varios protocolos, uno de ellos es el protocolo HTTP (Hyper-text Transfer Protocol) que da soporte al World Wide Web, comúnmente llamado Web. El Web ofrece el intercambio de documentos vía HTTP, estos documentos se encuentran principalmente en formato HTML y su contenido es desplegado mediante navegadores o visualizadores Web.

El Web es sólo uno de los numerosos servicios ofrecidos en Internet que proporciona una interfaz simple de usar y permite que personas con pocos conocimientos de computación accedan a los servicios Web. Estos servicios incluyen contenido, productos y servicios que pueden ser vistos y ordenados a través de un navegador. El navegador es una parte principal del auge comercial en Internet ya que permite a los clientes adquirir por si mismos productos o servicios ofrecidos en el Web.

Sin embargo, los fines comerciales no fueron el origen del desarrollo del Web. Los primeros sitios o sistemas Web fueron creados por Tim Berners-Lee en el CERN[1] a principios de los noventa para facilitar a los investigadores el acceso directo a documentos e información publicados por otros investigadores. A pesar de que el comienzo del Web se dio en la comunidad científica, a medida que el número de sitios Web se ha incrementado, el porcentaje de los sitios comerciales ha ido en aumento.

La industria divide al comercio electrónico o e-commerce en dos categorías o modelos de negocio principales: business-to-business (B2B) y business-to-consumer (B2C) [Korper, 2000]. Estos modelos dependen de los participantes en el proceso de negocio y la manera en que se comunican.

El modelo B2B implica la venta de productos y servicios entre corporaciones mediante la integración y automatización de sistemas. Esta categoría de comercio típicamente involucra a proveedores, distribuidores, fabricantes y almacenes. La mayoría de las transacciones ocurren directamente entre dos sistemas. Por ejemplo, una compañía automotriz que requiere partes de muchos proveedores para construir un automóvil puede implementar una solución de este tipo. Cada organización cuenta con una Intranet[2] que se conecta a otra a través de Internet, las organizaciones pueden ver sólo la información necesaria para realizar el negocio. Aunque pueden utilizarse redes privadas, el mantenimiento de estas resulta muy caro, el uso de Internet reduce los costos dramáticamente. El estándar OBI [3] proporciona un diseño flexible para soluciones B2B en Internet.

El modelo B2C comprende la interacción y transacciones entre una compañía y sus consumidores directos a través de la venta de bienes y servicios. Este modelo de comercio electrónico se ha vuelto familiar para muchas personas. Compañías como Amazon.com, Dell o eBay se enfocan a la venta directa al consumidor a través de Internet. Tradicionalmente las compañías y negocios pequeños emplean catálogos, agentes de ventas, medios como el teléfono o correo para vender sus productos. El comercio electrónico es el nuevo medio para vender bienes utilizando la tecnología Web, de esta manera el consumidor tiene más oportunidades de encontrar productos a mejores precios visitando los sitios Web de distintas compañías y consultando sus catálogos electrónicos.

En años recientes se han presentado muchos cambios en la manera en que las compañías hacen negocios, venden bienes y servicios o se comunican con sus proveedores o compradores. Debido al extraordinario crecimiento de la tecnología Internet, el comercio

---

[1] European Laboratory for Particle Physics.
[2] Red que utiliza los estándares de Internet pero únicamente el personal dentro de una organización puede acceder a ella.
[3] Open Buying on the Internet, http://www.openbuy.org.

electrónico jugará un papel importante en los procesos de negocio de las empresas pequeñas, medianas y grandes.

1.2 **Motivación**

El comercio electrónico proporciona inmejorables oportunidades de ventas para la mayoría de las organizaciones. Las compañías tienen la oportunidad de vender sus productos y servicios las 24 horas del día, reducir los costos asociados al personal, espacio de ventas, transacciones, e incrementar su participación en el mercado.

Las ventajas más importantes de contar con una solución de comercio electrónico en una organización se basan en las fortalezas de la infraestructura de Internet. Internet esta disponible en todo el mundo, es fácil de usar y los costos de transacción son bajos para el usuario final. Comparado con los medios tradicionales de distribución, Internet ofrece un costo menor a las compañías. Además, Internet permite la comunicación interactiva mediante estándares abiertos bien establecidos. La interacción en la comunicación permite una retroalimentación directa de los consumidores y los estándares facilitan las operaciones entre compañías, sitios Web y servicios.

Las compañías medianas y pequeñas tienen la oportunidad de ganar presencia y alcanzar un mayor número de clientes expandiendo su alcance más allá de su localización geográfica. Los costos de inventario pueden reducirse acortando los ciclos de venta y reaccionando de manera más rápida a las necesidades de sus clientes. Mediante la distribución de información en forma electrónica se reducen los costos de material, papelería, impresión y distribución manual. Además, los clientes cuentan con un nuevo medio de comunicación para la solicitud de servicios y consulta personalizada de la información de sus compras. Estas son algunas de las razones por las que muchas organizaciones deciden ingresar al comercio electrónico y mantener la competitividad mediante el uso de la tecnología de la información.

El desarrollo de aplicaciones o sistemas que proporcionan soluciones para comercio electrónico es una de las principales áreas de trabajo de los sistemas de información en la actualidad. Es necesario que las compañías que desean tener éxito en el ámbito de los negocios en línea tengan en cuenta varios aspectos como la infraestructura tecnológica con que cuenta la empresa, la competencia, las estrategias y metas de mercado, los servicios que pretende ofrecer y el análisis de cómo se verán afectados sus procesos de negocios antes de seleccionar la solución tecnológica adecuada. Contar con un sistema de comercio electrónico adecuado permite recuperar más rápidamente la inversión

económica y proporciona un medio eficaz para lograr las metas y objetivos de la organización.

1.3 Definición del Problema

Las compañías que desean participar en el comercio electrónico tienen que decidir entre adquirir una solución o producto de software en el mercado, rentar espacio con un proveedor que ofrezca servicios de compra en línea o desarrollar completamente un sistema con componentes y partes que se adapten exactamente a sus especificaciones.

Es difícil encontrar un producto o sistema para comercio electrónico que se adapte completamente a las necesidades de una organización. Este tipo de solución requiere que la compañía cuente con los recursos y conocimientos para instalar y dar mantenimiento a la aplicación. Para ello necesita contar con personal con conocimientos de programación para configurar y mantener el sistema. Estos sistemas tienen las funciones requeridas para un sistema de comercio electrónico e incluyen algunas reglas de negocio integradas en el producto. Si las necesidades del negocio son similares a lo que el producto ofrece, esta puede ser una buena opción a corto plazo. Un problema que se presenta con muchos de estos productos es que necesitan el establecimiento de una infraestructura de tecnología de información específica, muchas veces es necesario adquirir otros productos del proveedor que sólo funcionan para una plataforma de software, como es el caso de las soluciones para comercio electrónico que ofrece Microsoft[4] El costo de adquirir una determinada infraestructura puede ser prohibitivo para muchas empresas; sin embargo, esto es necesario cuando se requiere un control total sobre la disponibilidad y el contenido del sistema. También deben considerarse los costos asociados al mantenimiento y la adición de características o funciones que eviten que el sistema se vuelva obsoleto.

Las compañías pequeñas frecuentemente no cuentan con los recursos financieros para implementar y mantener una solución compleja para comercio electrónico, estas compañías pueden recurrir a los servicios que ofrecen algunos proveedores para rentar espacio[5] y mantener una tienda en línea. El proveedor del servicio cuenta con la infraestructura necesaria para hacer funcionar la tienda y la compañía sólo le proporciona la presentación y los productos que desea comercializar. Aunque es posible configurar algunas propiedades, esta solución presenta problemas de integración con sistemas ya existentes en la empresa. Este tipo de servicios puede parecer accesible económicamente; sin embargo, es difícil personalizar y agregar nuevas funciones al sistema, ya que no se cuenta con el control total sobre el software y hardware que administra el proveedor del servicio.

---

[4] http://www.microsoft.com/

[5] Este servicio se denomina comúnmente como *hosting*.

La tercer opción comprende el desarrollo completo de un sistema para comercio electrónico por parte de la empresa interesada. Para esto es necesario que la empresa cuente con personal que tenga conocimientos de programación e ingeniería de software para hacerse cargo de todo el proceso de desarrollo de software que un sistema de este tipo requiere. Los costos en tiempo y esfuerzo dependen de la complejidad del sistema y requieren un análisis cuidadoso de los recursos con que cuenta la empresa antes de elegir esta solución. La mayoría de las empresas no cuenta con el personal capacitado ni los recursos para el desarrollo de un sistema a la medida de sus necesidades. Aunque existen herramientas que facilitan la implementación de este tipo de sistemas, esto puede llevar un tiempo excesivo o la utilización de un grupo numeroso de desarrolladores.

1.4 **Objetivos**

Uno de los objetivos del presente trabajo es la definición de la arquitectura de software de un sistema para comercio electrónico que comprenda las necesidades básicas del modelo de negocio business-to-consumer (B2C). Para la definición de esta arquitectura deben tenerse en consideración los siguientes objetivos particulares:

- Utilizar componentes de software que permitan la reutilización de código de una manera eficiente.

- Obtener una arquitectura de software fácil de transportar para ser independiente de la plataforma de software y hardware.

- Proporcionar la facilidad de extensión que permita añadir las funciones necesarias para nuevos requerimientos del sistema.

- Utilizar un enfoque modular para la división del sistema y para la organización del trabajo de desarrollo.

- Establecer una base para el desarrollo de una línea de productos que cuente con facilidad de integración y se adapte fácilmente a las necesidades de diferentes tipos de empresas.

Mediante el uso de componentes reutilizables debe ser posible acortar los tiempos de desarrollo de aplicaciones de manera significativa utilizando grupos pequeños de desarrolladores especializados en diferentes áreas del sistema.

Otro de los objetivos de la tesis es el diseño e implementación de un prototipo del sistema utilizando la arquitectura de software propuesta. El trabajo de diseño e

implementación de esta tesis comprende solamente la interfaz de usuario o cliente del sistema. Algunas de las metas de diseño del cliente del sistema son las siguientes:

- Construir la interfaz de usuario mediante el diseño de páginas Web.
- Minimizar el procesamiento o lógica de negocio en el cliente.
- Mantener un mínimo de control sobre la configuración del cliente.
- Separar la lógica de la aplicación de la apariencia de las páginas

1.5 Estructura de la Tesis

Esta tesis está dividida en 6 capítulos; como ya se vio, el capítulo 1 presenta una introducción a los sistemas de comercio electrónico y sus modelos de negocio, además se presenta la motivación, la definición del problema y los objetivos de este trabajo. A continuación se describe el contenido de los demás capítulos.

En el capítulo 2 se presentan los conceptos fundamentales de la arquitectura de software de sistemas de una manera general, se parte de una definición de lo que es la arquitectura de software, su ciclo de vida y sus atributos de calidad. Se describen algunos de los estilos de arquitectura de software más comunes, así como el análisis y revisión de arquitecturas. También se describe lo que es el desarrollo basado en la arquitectura y por último se presentan las características de una arquitectura para líneas de productos.

El capítulo 3 trata sobre la interfaz de usuario y el intercambio de datos en el contexto de las aplicaciones Web, se describen los estilos de arquitectura más comunes para aplicaciones Web y algunas de las arquitecturas para interfaz de usuario más conocidas. Se presenta una descripción de los tipos de documentos que pueden encontrarse en el Web, así como algunas de las tecnologías para la presentación y el intercambio de datos más recientes.

El capítulo 4 presenta una descripción de la arquitectura de software del sistema para comercio electrónico. Se describen las cualidades de diseño de esta arquitectura de software y los estilos o patrones de arquitectura que fueron utilizados para obtener estas cualidades. Además, se muestra como se realizó la división del sistema de comercio electrónico; primero se describen las diferentes capas que conforman la arquitectura de software y luego se presenta las descripción de la división en módulos del sistema.

El diseño del cliente del sistema de comercio electrónico es tratado en el capítulo 5. Primero se hace una descripción de la tecnología utilizada para el diseño de la interfaz

de usuario del sistema, se discuten algunas de sus ventajas y los modelos de aplicación disponibles para esta tecnología. Luego se presentan algunas consideraciones sobre el diseño del cliente, se describe la notación para el modelo de diseño y la manera como se realiza la interacción del usuario con el sistema.

Las conclusiones y el trabajo futuro se discuten en el capítulo 6. En este capítulo se presentan las conclusiones obtenidas en base al trabajo de tesis y los objetivos alcanzados. También se mencionan las áreas de trabajo futuro relacionadas con esta tesis en el campo de las aplicaciones para comercio electrónico.

El apéndice A presenta una descripción de los casos de uso significativos para la arquitectura de software del sistema para comercio electrónico. El apéndice B muestra un extracto del modelo de análisis del sistema. El apéndice C contiene una descripción de la arquitectura de software del sistema. En el apéndice D se presenta un extracto del modelo de diseño del cliente del sistema. Por último, el apéndice E contiene una lista de los acrónimos más utilizados en esta tesis.

Capítulo 2 **Arquitectura de Software**

2.1 **Introducción**

En este capítulo se introducen los conceptos de arquitectura de software y su importancia en el desarrollo de sistemas complejos. Los temas aquí tratados son estudiados extensamente en el libro por [Bass et al, 1998] y en los siguientes artículos: [Abowd, 1996], [Garlan, 1995], [IEEE, 1995], [Parnas, 1972], [Parnas, 1974], [Parnas, 1976], [Parnas, 1979], [Shaw, 1997].

Este capítulo es un intento de síntesis de los temas tratados en las referencias anteriores.

2.2 **Definición de Arquitectura de Software**

Existen muchas definiciones de lo que es una arquitectura de software, casi tantas como los autores que han escrito sobre la materia, para nuestro caso adoptaremos la definición propuesta por [Bass et al, 1998]:

> La arquitectura de un programa o sistema de cómputo es la estructura o estructuras del sistema, que constan de componentes de software, de las propiedades visibles externamente de estos componentes, y las relaciones entre ellos.

Por *propiedades visibles externamente* entendemos aquellas características acerca de las cuales tienen conocimiento otros componentes del sistema, como pueden ser los servicios que ofrece, sus características de desempeño, tolerancia a fallas, recursos compartidos, etc. Esta definición pretende abstraer las propiedades relevantes arquitectónicamente y dejar de lado detalles que no son necesarios para el análisis, la toma de decisiones y la reducción de riesgos en el diseño de sistemas complejos.

La arquitectura define *componentes* y la forma en como estos interactúan entre sí. La arquitectura es una *abstracción* del sistema que suprime los detalles de los componentes que no afectan la manera como usan, son usados, o interactúan con otros componentes. Normalmente la estructura de un componente se divide en una parte pública y una privada, la parte pública es relevante arquitectónicamente, mientras la parte privada incluye detalles de implementación específicos del componente.

La definición menciona que *un sistema puede constar de más de una estructura* y que ninguna de ellas puede considerarse como la arquitectura del sistema. Frecuentemente los proyectos grandes son divididos en componentes que se utilizan como unidades de asignación de trabajo entre los equipos de programación. Este tipo de componente es llamado *módulo* y es utilizado para describir sistemas. Sin embargo, un sistema puede ser construido como un conjunto de procesos que interactúan entre ellos, comparten recursos y deben ser sincronizados; este conjunto de procesos define otro tipo de estructura que describe al sistema. Ninguna de las estructuras anteriores define completamente la arquitectura del sistema sino que son complementarias. También existe más de un tipo de interacción entre componentes como pueden ser la subdivisión y la sincronización.

La definición implica que *todo sistema tiene una arquitectura*, ya que se puede mostrar que el sistema está compuesto por componentes y las relaciones entre ellos. En el caso trivial, un sistema, es en sí mismo, un componente. Aunque todo sistema tiene una arquitectura, no necesariamente siempre es conocida por todos los involucrados, de ahí se deriva otro aspecto importante de una arquitectura: como base para la comunicación de todos los interesados e involucrados en el sistema.

El *comportamiento de cada componente es parte de la arquitectura* en la medida en que este comportamiento puede ser observado desde el punto de vista de otros componentes. Este comportamiento es lo que permite a los componentes tener interacción con los demás; por lo tanto, es parte de la arquitectura y debe existir una especificación de interfaz del componente.

Ya que es posible que existan, tanto buenas como malas arquitecturas, es necesario contar con un mecanismo de *evaluación de arquitecturas* que nos permita prever que el sistema cumplirá con los requerimientos de comportamiento y desempeño. La evaluación de la arquitectura elimina la necesidad de seleccionar una arquitectura mediante prueba y error.

Existen otros conceptos comunes que es importante destacar:

1. Un ***estilo de arquitectura*** es una descripción de los tipos de componentes y un patrón que describe su interacción. El estilo define una serie de restricciones sobre la arquitectura, lo cual da origen a familias de arquitecturas que satisfacen estas restricciones. Por ejemplo, existe un estilo común de arquitectura llamado cliente-servidor; en este estilo, tanto el cliente como el servidor identifican a los tipos de componentes, y su coordinación se describe en términos del protocolo de comunicación utilizado.
2. Un ***modelo de referencia*** es una división de la funcionalidad junto con el flujo de datos entre las partes. Consiste en la descomposición de un problema conocido en las partes que cooperan para resolver un problema. Por ejemplo, un compilador generalmente puede descomponerse en módulos que realizan funciones específicas, como pueden ser: análisis léxico, sintáctico, semántico y la generación de código. Los modelos de referencia son producto de un análisis de dominio y de la experiencia adquirida en el mismo.
3. Una ***arquitectura de referencia*** es la asignación de un modelo de referencia a los componentes que implementan la funcionalidad definida en el modelo así como el flujo de datos entre los componentes. Es decir, ya que el modelo de referencia divide la funcionalidad de un sistema, la arquitectura de referencia asigna esta funcionalidad mediante la descomposición del sistema. Esta asignación no debe ser necesariamente uno a uno, sino que un componente puede realizar varias funciones o parte de ellas.

Los modelos de referencia, los estilos de arquitectura y las arquitecturas de referencia no son arquitecturas completas; sino pasos intermedios para el desarrollo de la misma, como puede apreciarse en la figura 2.1. Los modelos de referencia, estilos de arquitectura y arquitecturas de referencia existen para sistemas que aparecen muy frecuentemente en la práctica y para los cuales se conocen parcialmente aspectos de la arquitectura final.

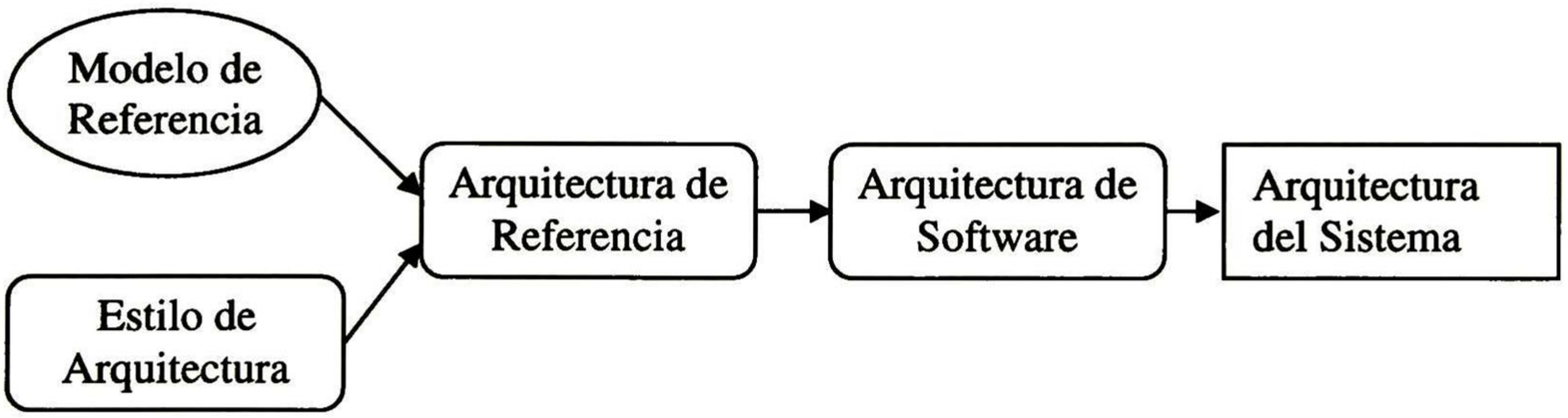


Figura 2.1 Relación entre los conceptos de arquitectura de software.

Hay tres razones principales por las cuales es importante la arquitectura de software:

1. ***Como medio de comunicación entre los participantes***. La arquitectura de software representa una abstracción de alto nivel del sistema que la mayoría, sino es que todos los participantes, pueden utilizar como base común de entendimiento, formación de consenso, y comunicación entre ellos.
2. ***Apoyo en la toma de decisiones de diseño.*** La arquitectura de software representa las decisiones de alto nivel en el diseño, que influyen de manera importante en el resto del trabajo de desarrollo, su emplazamiento, y su mantenimiento. También proporciona el primer mecanismo de evaluación del sistema que se planea construir.
3. ***Como abstracción del sistema que puede ser reutilizada.*** La arquitectura de software constituye un modelo relativamente pequeño y comprensible de cómo se encuentra estructurado un sistema, así como la manera en que trabajan sus componentes; este modelo puede servir como base para otros sistemas con requerimientos similares.

2.2.1 **Estructuras Arquitectónicas**

Como se ha mencionado anteriormente, la arquitectura de software consta de varias estructuras que nos muestran diferentes puntos de vista acerca del sistema. Algunas de las estructuras de software más comunes y útiles son:

- ***Estructura de módulos***
- ***Estructura lógica o conceptual***
- ***Estructura de procesos o de coordinación***
- ***Estructura física***
- ***Estructura de uso***
- ***Estructura de llamadas***
- ***Estructura de flujo de datos***
- ***Estructura de flujo de control***
- ***Estructura de clases***

2.3 Ciclo de Vida de la Arquitectura de Software

La arquitectura de software es el resultado de influencias técnicas, sociales y de negocio. De la misma manera, la existencia de la arquitectura de software afecta el ambiente técnico, social y de negocio que tiene influencia en las arquitecturas futuras. Este ciclo de influencias mutuas, del ambiente a la arquitectura y de la arquitectura al ambiente, es llamado el ciclo de vida de la arquitectura.

Muchas personas y organizaciones están interesadas en la construcción de sistemas de software, estos *participantes* incluyen al cliente, usuarios finales, desarrolladores, la organización del desarrollador, y las personas encargadas del mantenimiento, entre otras. Estos participantes tienen diferentes intereses y perspectivas acerca de lo que el sistema debe garantizar y optimizar; el comportamiento en tiempo de ejecución, facilidad de uso, tiempos y costos de desarrollo cortos y habilidades de los desarrolladores.

La arquitectura de software del sistema es el primer producto del trabajo o entregable que hace posible asignar prioridades a las características que deben ser analizadas y las describe como cualidades del sistema. Las negociaciones entre desempeño y seguridad, entre facilidad de mantenimiento y fiabilidad, entre el costo del esfuerzo de desarrollo actual y los desarrollos futuros son manifestadas en la arquitectura.

Una arquitectura es el resultado de un conjunto de decisiones técnicas y de negocio. Existen muchos factores que influyen en el diseño de una arquitectura, esto depende del ambiente donde se pretende que funcione la misma. Por ejemplo, el arquitecto que trabaja en un sistema en tiempo real diseñará una arquitectura diferente a aquel que lo hace para un sistema que realiza trabajo por lotes, ambos arquitectos toman decisiones diferentes. Aún si ambos sistemas tienen restricciones parecidas, es muy probable que existan marcadas diferencias en las arquitecturas desarrolladas por arquitectos diferentes.

El problema principal se presenta cuando los participantes tienen metas e intereses diferentes e incluso contradictorios. Además de los requerimientos, la arquitectura es influenciada por la estructura o naturaleza de la organización de desarrollo. Los arquitectos son influenciados por los requerimientos del producto, la estructura y las metas de la organización de desarrollo, la tecnología disponible, y su propia experiencia.

Las relaciones entre las metas del negocio, los requerimientos del producto, la experiencia de los participantes, arquitecturas y sistemas, forman un ciclo de retroalimentación para la toma de decisiones y evaluación de operación del negocio. La

retroalimentación proviene de la arquitectura y del sistema construido a partir de ella, de la siguiente manera:

1. ***La arquitectura afecta a la estructura de la organización de desarrollo.*** Como la arquitectura proporciona la base para la planeación y la asignación de trabajo a los equipos de desarrollo, esto influye de manera directa en la manera como debe estructurarse la organización de desarrollo.
2. ***La arquitectura puede afectar las metas de la organización de desarrollo.*** Un sistema exitoso puede hacer ganar a la empresa una importante posición en un área de mercado en particular.
3. ***La arquitectura puede influenciar los requerimientos del cliente.*** Las próximas versiones de un sistema pueden añadir funciones, y ofrecer una mayor fiabilidad a partir de la arquitectura desarrollada anteriormente, proporcionándole al cliente una solución económica a cambio de un mayor grado de flexibilidad en los requerimientos.
4. ***El proceso de construcción del sistema incrementa la experiencia del arquitecto.*** Un desarrollo exitoso de la arquitectura de un sistema puede servir como base para desarrollos subsecuentes de sistemas con características similares, reduciendo así el tiempo de desarrollo debido a la experiencia adquirida por los arquitectos.
5. ***Algunos sistemas influyen en la cultura de ingeniería de software y el ambiente técnico en el cual los desarrolladores de sistemas se desempeñan.*** Algunos ejemplos de estos sistemas son las primeras bases de datos relacionales, generadores de compiladores y sistemas operativos de la década de los sesenta y principios de los setenta. Las primeras hojas de cálculo y los sistemas de administración de ventanas de la década de los ochenta. CORBA, Java y el World Wide Web son ejemplos de la década de los noventa. Cuando estos sistemas innovadores son construidos, los sistemas subsecuentes son afectados por ellos.

Estos y otros mecanismos son los que conforman el ciclo de vida de la arquitectura de software como puede observarse en la Figura 2.2.

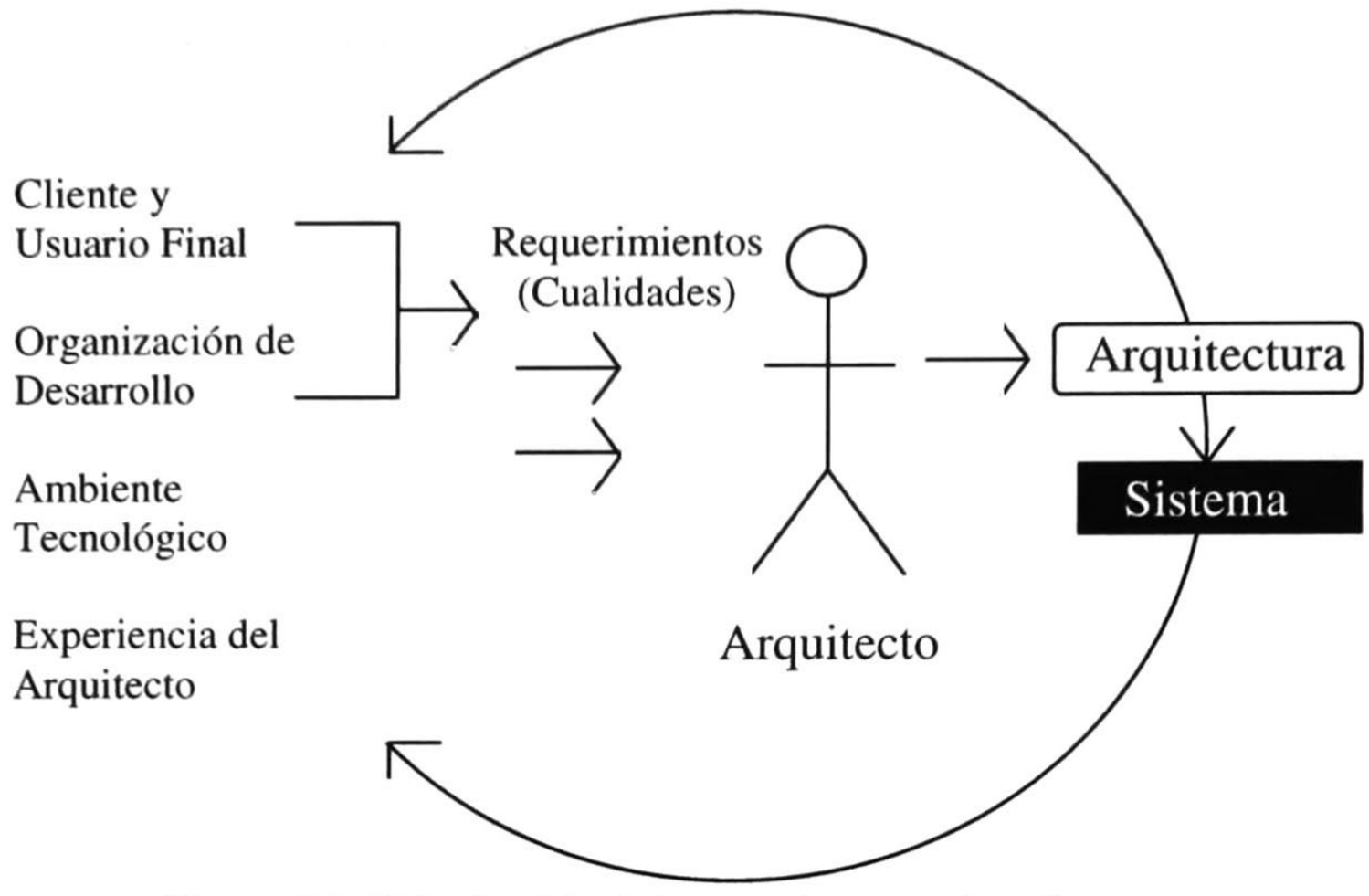


Figura 2.2 Ciclo de vida de la arquitectura de software.

2.3.1 Atributos de Calidad

Los requerimientos del negocio determinan las cualidades que deben ser incluidas en la arquitectura del sistema. Estas cualidades se refieren a la funcionalidad, la cual trata de las capacidades del sistema, los servicios, y el comportamiento. Frecuentemente los sistemas son rediseñados debido a que son difíciles de mantener, extender, son lentos o tienen fallas de seguridad; pero en ocasiones ofrecen una funcionalidad aceptable. Existe una relación directa entre la funcionalidad del sistema sobre las estructuras de software que determinan el soporte de la arquitectura para las cualidades del sistema. Una relación importante entre las cualidades de un sistema es que ninguna de ellas puede ser maximizada sin sacrificar de alguna manera alguna otra cualidad o cualidades.

Existen dos grandes categorías de atributos de calidad que sirven para medir un sistema desde el punto de vista arquitectónico:

1. ***Observables vía ejecución.*** Requerimientos de comportamiento, exactitud de los resultados, rapidez, funcionamiento.
2. ***No observables vía ejecución.*** Facilidad de modificación, integración y pruebas.

La arquitectura es crítica para la realización de muchas de las cualidades de interés en un sistema, estas cualidades deben ser diseñadas y evaluadas a un nivel arquitectónico.

Algunas otras cualidades no son arquitectónicamente relevantes, y no es útil analizarlas desde el punto de vista de la arquitectura.

Atributos de calidad del sistema apreciables en tiempo de ejecución

1. Desempeño
2. Seguridad
3. Disponibilidad
4. Funcionalidad
5. Facilidad de uso

Atributos de calidad del sistema no apreciables en tiempo de ejecución

1. Facilidad de modificación
2. Facilidad de transportación o independencia de plataforma
3. Facilidad de reutilización
4. Facilidad de integración
5. Facilidad de verificación

Atributos de calidad de negocio

1. Tiempo de desarrollo
2. Costo
3. Tiempo de vida del sistema
4. Mercado
5. Facilidad de configuración
6. Integración con sistemas existentes

Atributos de calidad de la arquitectura

1. Integridad conceptual
2. Correcta y completa
3. Facilidad de construcción

2.4 **Estilos de Arquitectura de Software**

Un *estilo de arquitectura* consiste de algunas características clave y de reglas para combinarlas de manera que la integridad sea preservada. Un estilo de arquitectura de software es determinado por:

- Un conjunto de tipos de componentes (almacenes de datos, procesos, procedimientos) que desempeñan una función en tiempo de ejecución.
- Una topología de estos componentes que indica sus relaciones en tiempo de ejecución.
- Un conjunto de restricciones semánticas (por ejemplo, un almacén de datos no puede cambiar los valores que contiene).
- Un conjunto de conectores (llamadas a subrutinas, llamadas remotas a procedimientos, flujos de datos, sockets) que permiten la comunicación, coordinación, y cooperación entre componentes.

Un estilo no es una arquitectura, sino que define una clase de arquitectura, es decir, es una abstracción para un conjunto de arquitecturas que cumplen con las propiedades del estilo.

2.4.1 **Catálogo de Estilos de Arquitectura de Software**

2.4.1.1 Arquitecturas Centradas en Datos

Estas arquitecturas tienen como meta la facilidad de integración de datos. El término de *arquitecturas centradas en datos* se refiere a aquellos sistemas en los cuales el acceso y la actualización de un almacén de datos es la característica principal. Los datos compartidos que todos los clientes actualizan pueden ser almacenes pasivos (como un archivo) o un almacén activo (como un pizarrón). Los medios de comunicación o *modelo de coordinación* distingue dos subtipos: almacén y pizarrón. Un pizarrón envía notificación a sus suscriptores cuando algún dato de interés cambia, por esta razón es activo. Los estilos centrados en datos han incrementado su importancia debido a que ofrecen una solución estructural para la facilidad de integración. Además este estilo es fácil de escalar, es decir, se pueden agregar clientes fácilmente.

2.4.1.2 Arquitecturas de Flujo de Datos

La meta de este estilo de arquitectura es la obtención de la facilidad de reutilización y la facilidad de modificación. Se caracteriza por realizar una serie de transformaciones sobre partes sucesivas de datos de entrada. Los datos entran al sistema y siguen un flujo a través de los componentes hasta que llegan a su destino final (salida o almacén de datos). Existen dos subtipos de este estilo, secuencial por lotes y el de conducto-filtro (pipe-and-filter).

En el *estilo secuencial por lotes*, los pasos de procesamiento, o componentes, son programas independientes y cada uno de ellos termina antes de que comience el siguiente paso.

El *estilo conducto-filtro* enfatiza la transformación incremental de los datos mediante una sucesión de componentes. Este estilo es típico en la familia de los sistemas operativos UNIX. Los filtros transforman los datos de manera incremental utilizando muy poco la información del contexto. No retienen información de estado entre instancias. Los conductos no tienen estados y solo mueven los datos entre los filtros. Su principal ventaja es la simplicidad. No existen complejas interacciones que manejar, simplifica el mantenimiento y fomenta la reutilización. Su principales desventajas son la dificultad para crear aplicaciones interactivas, la restricción de una representación común de los datos (regularmente ASCII) y la utilización de memoria intermedia para evitar bloqueos mutuos.

2.4.1.3 Arquitecturas de Máquina Virtual

Este estilo de arquitectura tiene como principal objetivo la obtención de sistemas de software portátiles. Las máquinas virtuales son estilos de software que simulan la funcionalidad que no es nativa al hardware o software sobre los cuales están implementados. Son útiles para la simulación y prueba de sistemas.

Algunos ejemplos de máquinas virtuales son los intérpretes, sistemas basados en reglas y procesadores de lenguajes de comandos. El lenguaje Java está construido para ejecutarse sobre la Máquina Virtual de Java, lo que permite que este lenguaje sea independiente de la plataforma.

La ejecución de un programa por medio de un intérprete añade flexibilidad a través de la habilidad para interrumpir o consultar un programa e introducir modificaciones en tiempo de ejecución. La principal desventaja es que existe un costo en el desempeño debido al cómputo adicional que es necesario para la ejecución.

2.4.1.4 Arquitecturas de Llamada-retorno (call-and-return)

Este estilo de arquitectura tiene como objetivo la facilidad de modificación y la facilidad de extensión. Las arquitecturas de llamada y retorno han sido el estilo de arquitectura dominante en sistemas grandes de software en los últimos 30 años. Existen varios subtipos del estilo que a continuación se describen.

Las *arquitecturas de programa principal y subrutina* son el modelo clásico de programación. Su meta es descomponer un programa en partes pequeñas que proporcionen la facilidad de modificación. Esta descomposición es jerárquica.

Los sistemas de *llamadas a procedimientos remotos* son sistemas de programa principal y subrutina que son descompuestos en partes que están en computadoras conectadas por medio de una red. La meta es aumentar el desempeño mediante la distribución del cómputo y tomar ventaja de múltiples procesadores.

Los sistemas *orientados a objetos* o de *tipos de datos abstractos* son las versiones modernas de las arquitecturas de llamada y retorno. El paradigma orientado a objetos se caracteriza por conjuntar los datos con el conocimiento acerca de cómo manipular y acceder a los mismos. Una de sus metas es la facilidad de modificación. Los accesos al objeto son realizados mediante operaciones llamadas métodos, los cuales son versiones restringidas de llamadas a procedimientos. Las principales características que distinguen al paradigma orientado a objetos son el encapsulamiento (el usuario de un servicio no tiene que conocer como este es implementado), la herencia (definiciones y datos compartidos jerárquicamente) y el polimorfismo (la habilidad para determinar la semántica de una operación en tiempo de ejecución). Cuando los objetos forman componentes que proporcionan servicios de caja negra a otros componentes que solicitan estos servicios este estilo es llamado *cliente servidor basado en llamadas.*

Los s*istemas por capas* son aquellos cuyos componentes están asignados a capas para controlar la interacción entre componentes. Las metas de este estilo son la facilidad de modificación y la facilidad de transportación. La capa más baja proporciona funciones

básicas, como puede ser el hardware o el núcleo de un sistema operativo. Las capas sucesivas son construidas sobre sus predecesoras, ocultando la capa inferior y proporcionando servicios a las capas superiores.

2.4.1.5 Arquitecturas de Componentes Independientes

Consisten de un número de procesos independientes u objetos que se comunican a través de mensajes. Todas estas arquitecturas tienen como meta la facilidad de modificación mediante el desacoplamiento de porciones de cómputo. Los componentes envían datos a otros pero no se controlan directamente unos a otros.

Los *sistemas de eventos* son un subtipo del estilo en el cual el control es una parte del modelo. Los componentes publican la información que desean compartir y otros componentes pueden registrar su interés en esta clase de datos, es decir, se suscriben. Estos sistemas hacen uso de un administrador de mensajes para la comunicación entre los componentes. Los componentes registran los tipos de información que desean proporcionar y aquellos que desean recibir. La información es publicada enviando un mensaje al administrador de mensajes, el cual reenvía el mensaje a todos los componentes interesados.

El otro subtipo del estilo de componentes independientes es el de *procesos comunicantes*, el cual es utilizado en los sistemas multiprocesos clásicos. Un estilo bien conocido es el de cliente-servidor. La meta de este estilo es la facilidad de extensión. Un servidor proporciona datos a uno o más clientes, los cuales están localizados en distintos nodos de una red. El cliente origina una llamada al servidor, el cual responde de manera síncrona o asíncrona para prestar un servicio al cliente. Si el servidor trabaja de manera síncrona, este regresa el control al cliente al mismo tiempo que regresa los datos. Si el servidor trabaja de manera asíncrona, este regresa sólo los datos al cliente (el cual tiene su propio hilo de control).

2.5 **Análisis de Arquitecturas de Software**

El *método de análisis de arquitecturas de software* (MAAS) comprende un conjunto de técnicas de validación para determinar que las metas de las cualidades del sistema sean cumplidas mediante su arquitectura. Este método puede ser utilizado en dos contextos diferentes: como fase de validación durante el desarrollo de la arquitectura o como un paso requerido en la adquisición de un sistema de software.

La evaluación de las cualidades del software en las primeras etapas del ciclo de vida del software es mucho más eficiente en costo y es una de las principales razones para llevar a cabo el análisis de la arquitectura; un cambio en los requerimientos o en el diseño es mucho menos costoso que un cambio en la implementación del sistema.

Cuando una empresa desea adquirir un nuevo sistema de software que es ofrecido por varios vendedores, es de gran utilidad llevar a cabo un análisis de las arquitecturas propuestas para escoger aquella que cumpla con los requerimientos; si esto no es posible debe seleccionarse aquella que represente el menor costo y esfuerzo en su adaptación. Aún así, la arquitectura del sistema no puede garantizar la funcionalidad y las cualidades que se requieren. Un diseño o implementación deficientes pueden contrarrestar los beneficios de una arquitectura aceptable. El refinamiento de la arquitectura en una implementación que conserve las cualidades es necesario para obtener un producto aceptable.

2.5.1 **Entradas y Salidas del Proceso de Evaluación**

Obviamente se requiere una descripción o especificación de la arquitectura como base para el análisis. Como ya se ha mencionado anteriormente una arquitectura puede estar compuesta por varias estructuras o perspectivas que se enfocan en diferentes aspectos de un sistema. Por ejemplo, si la evaluación se enfocara en aspectos como el desempeño o paralelismo, se requerirá una descripción de la estructura de tareas y comunicación de la arquitectura. Si la facilidad de modificación es importante, entonces una descripción de la descomposición de la arquitectura en sus unidades de asignación de trabajo o módulos será necesaria. Si dos o más arquitecturas candidatas son comparadas para ver cual de ellas cumple mejor los requerimientos, MAAS producirá una clasificación de estas arquitecturas.

2.5.2 **Análisis de Cualidades Mediante Escenarios**

Los atributos de calidad no existen de manera aislada sino que adquieren su significado de acuerdo a su contexto. Un sistema es fácil de modificar o no, con respecto a ciertas clases de cambios; seguro o no, con respecto a ciertas clases de amenazas; fácil de usar o no, con respecto a tipos de usuarios específicos. Esta noción de evaluación basada en el contexto de los atributos de calidad ha llevado a la adopción de escenarios como medios de descripción, especificación y evaluación de los atributos de calidad. Un *escenario* es una breve descripción de una interacción sencilla entre el usuario o desarrollador y el sistema. Este concepto es similar al caso de uso de Jacobson en las metodologías

orientadas a objetos [Jacobson, 1999]. Sin embargo, los casos de uso se enfocan al comportamiento en tiempo de ejecución entre el usuario y el sistema solamente, mientras los escenarios de MAAS comprenden además otro tipo de interacción, como la que lleva a cabo el desarrollador para dar mantenimiento al sistema.

Un escenario sirve como representante de una clase de escenarios. Esta clase consiste en todos los escenarios para los cuales el sistema en consideración responde de la misma manera. Una de las métricas de MAAS es el número de escenarios de evaluación que afectan al mismo componente de la arquitectura.

En el caso de los escenarios, los participantes en él pueden ser:

- La persona responsable de la ejecución del software- el usuario final.
- La persona responsable de administrar los almacenes de datos utilizados por el sistema- el administrador del sistema.
- La persona responsable de modificar las funciones del sistema – el desarrollador.
- La persona responsable de aprobar nuevos requerimientos para el sistema.

En resumen, MAAS utiliza los escenarios, o descripciones concretas de los usos o funciones del sistema, como medio de comparación para las arquitecturas candidatas. El propósito de MAAS es proporcionar una manera de validar las cualidades del sistema a partir de la descripción de su arquitectura. Estas descripciones son comparadas con un conjunto de escenarios para determinar el grado de soporte que ofrece la arquitectura para cada escenario o si esta debe ser modificada para soportarlo. Después, las arquitecturas son comparadas entre sí, enfocándose en el desempeño que tienen para escenarios similares.

2.5.3 El Método de Análisis para Arquitecturas de Software

A continuación se definen los pasos del método de análisis de arquitecturas de software.

1. ***Desarrollo de escenarios.*** Los escenarios deben ilustrar los tipos de actividades que el sistema debe soportar y los tipos de cambios que pueden anticiparse en el sistema. Al desarrollar estos escenarios, es importante capturar todos los principales usos del sistema y las cualidades que el sistema debe satisfacer.
2. ***Descripción de las arquitecturas candidatas.*** La arquitectura o arquitecturas candidatas deben ser descritas en una notación que es bien entendida por todas las personas involucradas en el análisis. Estas descripciones arquitectónicas

deben indicar los componentes de control y datos del sistema, así como sus conexiones relevantes.

3. ***Clasificación de escenarios***. El sistema puede soportar un escenario directamente; es decir, si no requiere modificaciones en el sistema para que el escenario pueda ser ejecutado. Estos escenarios son llamados *directos*. En cambio, cuando un escenario no es soportado directamente, debe hacerse un cambio al sistema que puede representarse en su arquitectura. Este cambio puede hacerse a uno o más componentes, la adición de uno o más componentes, o la adición de conexiones entre los componentes existentes. Esta clase de escenarios es llamado *indirecto*.

4. ***Evaluación de escenarios***. Para cada escenario indirecto, los cambios necesarios para que la arquitectura soporte el escenario deben ser listados, y el costo de desempeñar el cambio debe ser estimado. Una tabla resumen es especialmente útil cuando se comparan diferentes arquitecturas candidatas, ya que esto proporciona una manera fácil para determinar cual arquitectura soporta mejor un conjunto de escenarios.

5. ***Identificar la interacción de escenarios***. Cuando dos o más escenarios requieren cambios en un solo componente del sistema, se dice que estos interactúan en ese componente. La interacción de escenarios es importante porque muestra como esta localizada la funcionalidad en el diseño del producto. Las áreas con un alto grado de interacción revelan una potencial falta de separación de funciones. El número de interacciones de escenarios está relacionado a métricas tales como complejidad estructural, acoplamiento y cohesión.

6. ***Evaluación general.*** Si diferentes arquitecturas están siendo comparadas, se debe asignar un peso a cada escenario o interacción de escenarios en términos de su importancia relativa, y esta medida debe ser utilizada para determinar una clasificación general de las diferentes arquitecturas.

El desarrollo de escenarios y la representación de arquitecturas son pasos independientes. El grado apropiado de granularidad de una arquitectura depende del tipo de escenarios que desean evaluarse. La determinación de un conjunto razonable de escenarios depende del tipo de actividades que se espera que desempeñe el sistema, y esto es reflejado también en la arquitectura.

2.6 **Desarrollo Basado en Arquitectura**

Aquí se trata sobre el proceso de creación de un sistema a partir de su arquitectura. Las partes del proceso de creación específicas a la arquitectura son las siguientes:

- Formación de la estructura de equipos y su relación con la arquitectura.
- Utilización de la arquitectura para crear una versión esqueleto del sistema.
- Utilización de patrones de diseño y codificación dentro de la estructura de la arquitectura.
- Comprobación del sistema final con respecto a la arquitectura.

2.6.1 **Formación de la Estructura de los Equipos**

Una vez que la arquitectura del sistema a ser construido ha sido aprobada, se asignan equipos para trabajar en los componentes principales y se crea una estructura desglosada de trabajo que refleja a estos equipos. Cada equipo puede crear sus prácticas de trabajo internas.

Los equipos dentro de una organización de desarrollo trabajan sobre componentes. Dentro de un equipo debe haber una gran comunicación, información detallada sobre las decisiones de diseño es intercambiada entre los miembros del equipo. Entre equipos, un menor grado de comunicación es requerido. Lo anterior es válido siempre y cuando exista una adecuada separación de funciones. En los equipos, como en los sistemas de software, debe mantenerse un menor grado de acoplamiento y una gran cohesión.

La estructura lógica o de módulos es la más adecuada para llevar a cabo la conformación de los equipos. El principio de diseño del ocultamiento de información en la estructura de módulos, permite a los equipos trabajar de manera que los detalles de implementación queden ocultos a los demás módulos y sólo sea necesario presentar una interfaz que proporcione un conjunto bien conocido de servicios a los otros equipos que así lo requieran. Esto implica que cada módulo sea visto como un pequeño domino, es decir un área de conocimiento especializado.

El reconocimiento de módulos como pequeños dominios sugiere inmediatamente una adecuada asignación de personal de acuerdo al grado de experiencia en el área en cuestión. A medida que una organización continúa trabajando en un dominio particular, desarrolla un conjunto de activos (productos del trabajo) que se utilizan como medios de comunicación, y existen grupos en la organización cuyo propósito es darle mantenimiento a estos activos.

2.6.2 **Creación del Esqueleto del Sistema**

La arquitectura de software sirve como base para la primera tarea de implementación: la creación de un *esqueleto del sistema.* Es decir, la arquitectura forma los huesos del sistema. La codificación, los subsistemas y componentes, y el diseño en general forman los músculos del sistema, sin embargo, la estructura básica y las relaciones entre las partes ya están determinadas. La idea es implementar los aspectos principales del comportamiento de la arquitectura, es decir los componentes descritos en ella a un nivel de funcionalidad menor pero ejecutable.

Mediante el uso de la arquitectura como guía, la secuencia de la implementación es clara. Primero se implementa el software que trata sobre la ejecución e interacción de los componentes de la arquitectura, por ejemplo, implementar la coordinación cliente-servidor en un sistema de este tipo. Este software es el único que existe realmente, todo lo demás es definición de interfaces. Entonces puede hacerse una selección de los componentes delineados que deben ser implementados con mayor prioridad. La selección puede hacerse basada en la reducción de riesgos, atacando las áreas más problemáticas primero o basándose en el nivel y tipo de personal disponible.

El sistema esqueleto que resulta de este enfoque sirve como base para la integración. En esta versión del sistema, las partes que faltan son inicialmente cubiertas con implementaciones sin funcionalidad. Estas implementaciones cumplen con las mismas interfaces que la versión final del sistema requiere, de manera que son útiles para entender y probar la interacción entre los componentes.

El desarrollo del esqueleto se caracteriza por la construcción general pero superficial del sistema: cada componente es parte de un prototipo operativo, pero la funcionalidad es menor. Una versión alternativa de este esquema puede usarse cuando se espera que un aspecto particular del sistema sea problemático o cuando se dispone de poco personal. La idea es construir de manera más detallada un pequeño subconjunto del sistema. Este subconjunto puede ser el más pequeño que pueda ser ejecutado y su trabajo pueda ser reconocido, la estructura de uso es de utilidad en la definición del subconjunto. Los subconjuntos subsecuentes pueden concentrarse en áreas de incertidumbre o en áreas sugeridas por el nivel de experiencia del personal.

Tener una versión esqueleto del sistema desde las primeras etapas del proceso de desarrollo es importante por varias razones:

- Tiene un gran impacto en la productividad y la moral del equipo, ya que el sistema existe y es operativo desde las primeras etapas.
- Permite la concentración en aquellos aspectos del sistema que se espera sean más problemáticos de implementar o sobre los que existe mayor incertidumbre.
- Reduce los tiempos y costos de integración.
- Ayuda a los procesos de revisión y pruebas ya que estas pueden realizarse siempre en el sistema existente y de manera incremental. El sistema es esencialmente su propia simulación.
- Las dependencias complejas son encontradas más rápidamente, debido a que la planeación de la integración y el trabajo de desarrollo se presenta al inicio del ciclo de vida del producto.

2.6.3 **Utilización de Patrones en la Arquitectura de Software**

Ser capaz de analizar y construir un sistema con un conjunto regular de bloques de construcción nos proporciona beneficios sustanciales. Estos incluyen una mejor comprensión de sistemas complejos, en provecho de su desarrollo y mantenimiento. Este conjunto de bloques de construcción representa patrones de uso común. Los patrones permiten a los humanos entender sistemas complejos por medio de partes conceptuales que disminuyen el esfuerzo de comprensión. Los patrones pueden ser de una mayor granularidad que los procedimientos individuales y de esta manera incrementar el tamaño de las partes que un humano puede comprender como unidad atómica.

Un *patrón* es una colección pequeña de unidades atómicas y una descripción de sus relaciones. En los patrones de diseño orientados a objetos, un patrón es una colección de clases, relaciones de herencia y de uso (llamadas a métodos) entre ellas.

Más frecuentemente los patrones están siendo documentados y comunicados como una forma de transmitir experiencia de los expertos a los principiantes. Los estilos de arquitectura vistos anteriormente son un tipo especial de patrones que representan soluciones a un nivel arquitectónico. Algunas de las ventajas del uso de patrones son las siguientes:

- Sirven como ejemplos para los programadores, diseñadores y arquitectos, los cuales son adaptados rápidamente para utilizarlos en sus proyectos.
- Ayuda en la comunicación entre diseñadores y programadores.

- El uso de patrones proporciona beneficios en el tiempo de desarrollo, mediante el uso de plantillas (templates). Las implementaciones pueden acelerarse utilizando plantillas de código que conforman una estructura común.
- Conforman una base de experiencia documentada y reutilizable que de otra manera tendría que ser aprendida de manera oral o a través de prueba y error.

2.6.3.1 Patrones de Diseño

Como se ha dicho antes un patrón de diseño es una pequeña colección de objetos o clases de objetos que cooperan para lograr una meta de diseño. Los diseñadores expertos analizan un problema en términos de sus requerimientos de calidad, y cuando es posible, adoptan una solución comprobada que satisface estos requerimientos. Para poder aprovechar estas soluciones, se realiza un esfuerzo sustancial en la documentación de los patrones de diseño orientados a objetos y de algunos metapatrones[6] El lector interesado en los patrones de diseño puede consultar [Gamma, 1995], donde podrá encontrar un catálogo de varios de ellos.

2.6.4 **Comprobación de Concordancia con la Arquitectura**

Una vez que los requerimientos de un sistema han sido analizados, sus patrones desarrollados, el esqueleto del sistema cubierto de componentes, la integración y las pruebas han terminado, contamos con el producto final. Es un hecho que los requerimientos cambiarán. Entonces el sistema entrará a la fase de mantenimiento que durará hasta el fin de su vida útil.

Ante los cambios sufridos en el sistema que pueden afectar a la arquitectura es necesario mantener actualizada su descripción. Frecuentemente las arquitecturas difieren de manera significativa de los documentos que la describen, y nada hace más inútil a un documento que ser inconsistente. Es crucial mantener la documentación de la arquitectura, regularmente el primer cambio es el más difícil y el más importante.

Cuando la arquitectura de un sistema no concuerda con su descripción o no se cuenta con una, es posible utilizar la llamada ingeniería inversa para determinar la arquitectura.

---

[6] Patrones que a su vez describen la estructura de otros patrones.

Esta disciplina tiene dos áreas principales, una basada en enfoques técnicos para la derivación de información a través de los entregables o productos del trabajo (código fuente, comentarios, documentación de usuario, módulos ejecutables, descripciones del sistema, etc.) y la otra basada en el conocimiento e inferencia del desarrollador (estrategias de diseño descendente, diseño ascendente, basadas en modelos, o una combinación de ellas). El resultado de estas técnicas es llamado *información de descubrimiento* y tiene dos principales usos:

1. Puede utilizarse para medir la concordancia entre la arquitectura planeada con respecto a la arquitectura en ejecución.
2. Puede utilizarse para hacer cambios de ingeniería al sistema (reingeniería).

2.7 Arquitecturas para Líneas de Productos de Software

Para una organización una arquitectura representa una inversión significativa de tiempo y esfuerzo de desarrollo. Es natural querer maximizar esta inversión mediante la reutilización de la arquitectura en múltiples sistemas, esto reduce los costos de construcción y el tiempo de desarrollo. Una *línea de productos de software* es una colección de sistemas que comparten un conjunto de características construidas a partir de un conjunto común de activos de software. Estos activos incluyen una arquitectura base y el conjunto de componentes adaptables que la conforman. Algunos otros activos que pueden reutilizarse en una línea de productos son:

- ***Componentes***. Incluye la reutilización de código, diseño, interfaces, documentación, planes y procedimientos de prueba, y modelos para predecir o medir el comportamiento del sistema.
- ***Personal***. El personal de desarrollo puede transferirse entre diferentes proyectos debido a las características que estos comparten.
- ***Procesos, métodos y herramientas***. Procedimientos de control de la configuración, planes de documentación, procedimientos de aprobación, entornos de desarrollo, estándares de codificación.
- ***Planeación del proyecto***. Los presupuestos y calendarios pueden anticiparse con mayor exactitud debido a la experiencia adquirida en anteriores proyectos.

El desarrollo de una línea de productos no es una tarea simple. Una de las razones por las que la reutilización de software no se ha desarrollado completamente es que el enfoque requerido para ello implica una reestructuración de la organización. Algunos elementos que deben tomarse en consideración en una línea de productos son: la forma en que son creados los productos individuales, la forma en que evoluciona la

arquitectura de línea de base de la línea de productos, la inclusión de componentes externos en el sistema base, el impacto en la organización de la línea de productos.

2.7.1 **Creación de Productos y Evolución de una Línea de Productos**

Una organización que posee una línea de productos eventualmente crea un nuevo producto que es un miembro de ella. Este producto tendrá características en común con los otros miembros de la línea de productos además de sus características particulares. Uno de los problemas asociados con una línea de productos es el manejo de su evolución. Esta evolución tiene principalmente tres fuentes:

1. Nuevas versiones de componentes dentro de la línea de productos.
2. Adición de nuevos componentes externos en la línea de productos.
3. Nuevas características agregadas a la línea de productos para mantener la competitividad.

Un segundo problema es el manejo de la evolución individual de los productos después de su creación. Debe tomarse la decisión de actualizar la arquitectura base cuando una función es añadida a un producto particular. Esta actualización puede ser una buena decisión cuando es probable que la funcionalidad agregada sea utilizada en productos futuros. Un tercer problema es la administración de productos entregados cuando la arquitectura base evoluciona, es decir, el mantenimiento de la compatibilidad de los productos toma tiempo y esfuerzo cuando los componentes anteriores deben ser reemplazados.

2.7.2 **Impactos en la Organización de una Línea de Productos**

Una línea de productos afecta a una organización en su estructura, en sus relaciones con sus clientes, y en el entrenamiento del personal. Esto lleva a que una parte de la organización administre los elementos de la línea de productos – la arquitectura, los componentes, los casos de prueba, etc. , y otra parte se encargue del desarrollo de productos para el cliente. El equipo de mercadotecnia será responsable de comunicar las necesidades del cliente a ambas partes, y la administración es responsable de la decisión de asignar que necesidades serán satisfechas por una u otra parte de la organización.

La figura 2.3 muestra la estructura de una organización que administra una línea de productos de software.

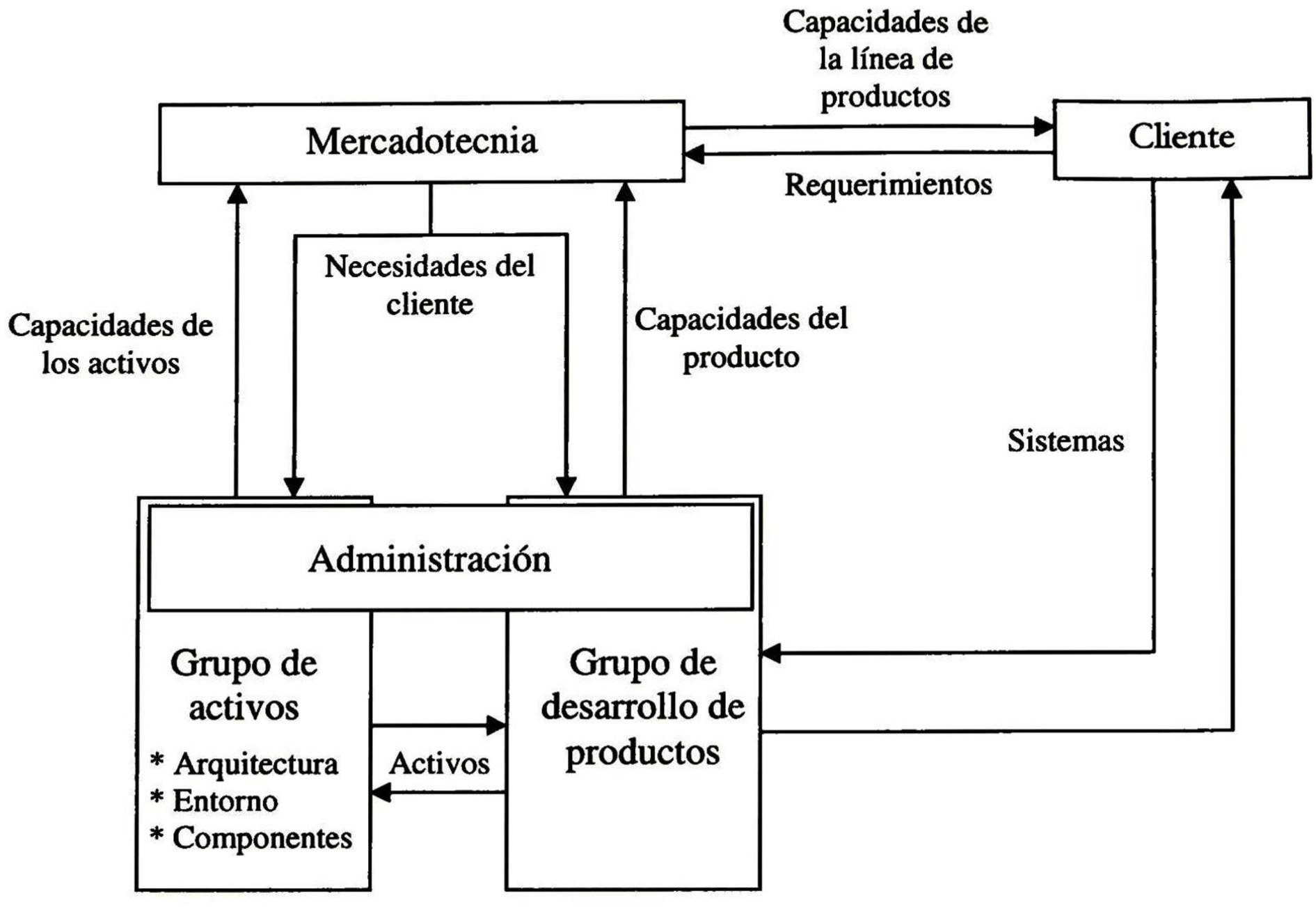


Figura 2.3 Organización para el desarrollo de líneas de productos.

2.7.3 **Sistemas Basados en Componentes**

El porcentaje de sistemas construidos con componentes adquiridos es mucho mayor que hace unos pocos años. "Comprar, no construir" es la tendencia en el desarrollo de software. Sin embargo, comprar el software significa menos control sobre cada aspecto de desarrollo del sistema. Debido a que en los sistemas de tamaño considerable la arquitectura es determinante para las cualidades del sistema, la manera de contrarrestar la desventaja anterior es llevar a cabo la integración de los componentes de manera que no se comprometa a la arquitectura subyacente. Algunos de los beneficios de la construcción de sistemas basados en componentes son los siguientes:

- Capacidad de tomar ventaja de nuevos productos y tecnologías.
- Reducción significativa del tiempo de desarrollo.
- Mayor productividad, con énfasis en la reutilización e integración.
- Mayor fiabilidad debido a componentes comprobados.
- Sistemas más flexibles debido a la facilidad de reemplazo de componentes.

- Sistemas fáciles de extender debido a la facilidad para agregar componentes.

Sin embargo, no todos los componentes trabajan juntos como es esperado, los errores pueden ser sutiles, especialmente sobre los supuestos acerca del tiempo u orden de operaciones. Los componentes que no son desarrollados específicamente para un sistema pueden no cumplir todos los requerimientos. Por desgracia, en ocasiones no es posible saber si un componente es el adecuado o no hasta que no es comprado y probado. El término *inconsistencia arquitectónica (architectural mismatch)* describe los problemas que surgen en la integración de sistemas basados en componentes. La inconsistencia arquitectónica es un caso especial de la *inconsistencia de interfaces*, donde una interfaz es definida como los supuestos que los componentes pueden hacer acerca de los demás. El concepto normal en la práctica es el de interfaz de programación de aplicación (API). Una API nombra los programas y sus parámetros y nos dice algo acerca de su comportamiento, pero esto es sólo una parte de la información necesaria para usar correctamente un componente. El consumo de recursos globales y los requerimientos de sincronización son partes de la interfaz del componente y deben ser incluidos en su especificación.

Capítulo 3 Interfaz de Usuario e Intercambio de Datos

3.1 Introducción

En este capítulo tratamos los siguientes temas: estilos de arquitectura para aplicaciones Web, arquitecturas de interfaz de usuario, representación de documentos y métodos de intercambio de datos entre dos sitios geográficamente distantes. Discutimos las ventajas y desventajas de las arquitecturas, las diferentes representaciones de documentos y los métodos de intercambio de datos más recientes. Nuestro interés en la arquitectura de interfaz de usuario es debido a que como parte de esta tesis desarrollaremos la interfaz de usuario de un sistema de comercio electrónico y usaremos la arquitectura seleccionada como base para nuestra implementación. También estamos interesados en la comunicación entre el cliente y el servidor que es el paradigma que se utilizará para la implementación del sistema de comercio electrónico y en los estilos de arquitectura para aplicaciones Web aplicables. En otro capítulo presentamos las implementaciones de las arquitecturas que aquí describimos.

3.2 Estilos de Arquitectura para Aplicaciones Web

3.2.1 Introducción

Existen varios patrones o estilos de arquitectura para desarrollar aplicaciones Web, estos estilos nos describen los tipos de componentes que contiene el estilo y la forma en que estos interactúan. Una aplicación Web permite utilizar lógica del negocio que cambie el estado del sistema en el servidor; por ejemplo, al acceder y modificar las bases de datos. Una aplicación Web debe tener tres tipos de componentes como mínimo: un cliente ejecutando un navegador, el servidor Web, y el servidor de aplicaciones.

Una aplicación Web definida como un sistema cliente/servidor consta de:

- Un navegador que soporta HTML o XML[7] ejecutándose en una o más máquinas cliente que se conectan a un servidor Web mediante el protocolo HTTP.
- Un servidor de aplicaciones que se encarga de la lógica del negocio.

Los estilos de arquitectura de cliente delgado, cliente grueso y de entrega para aplicaciones Web son descritos en las siguientes secciones.

3.2.2 Estilo de Arquitectura de Cliente Delgado

El estilo de arquitectura de cliente delgado tiene poco control de la configuración del cliente, el navegador debe ser capaz de soportar formularios HTML y todo el procesamiento de la aplicación se realiza en el servidor. La siguiente figura muestra la configuración del estilo de arquitectura de cliente delgado en un modelo de tres capas.

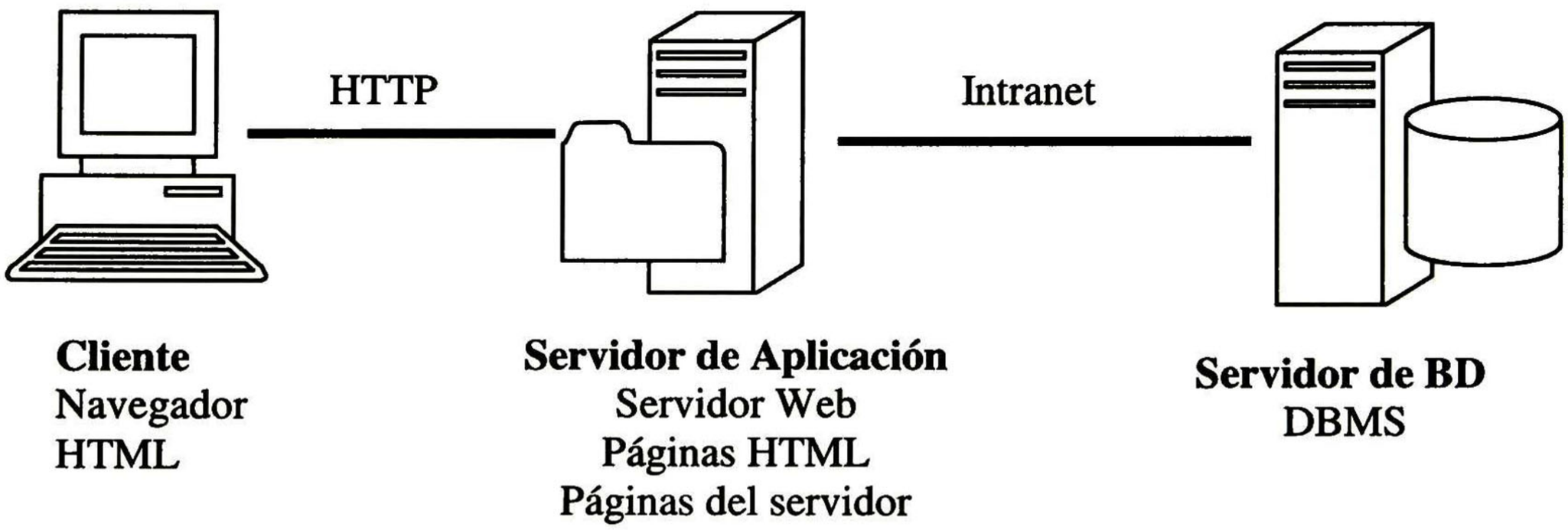


Figura 3.1 Estilo de arquitectura de cliente delgado.

3.2.3 Estilo de Arquitectura de Cliente Grueso

En el estilo de arquitectura de cliente grueso una cantidad considerable de procesamiento de la aplicación se lleva a cabo en la máquina cliente. El navegador puede utilizar HTML Dinámico, Applets de Java, o controles ActiveX para ejecutar la lógica del negocio de la aplicación. Mientras que la comunicación con el servidor sigue realizándose mediante el protocolo HTTP. La siguiente figura muestra la estructura de este estilo.

---

[7] Lenguaje de Marcado Extensible.

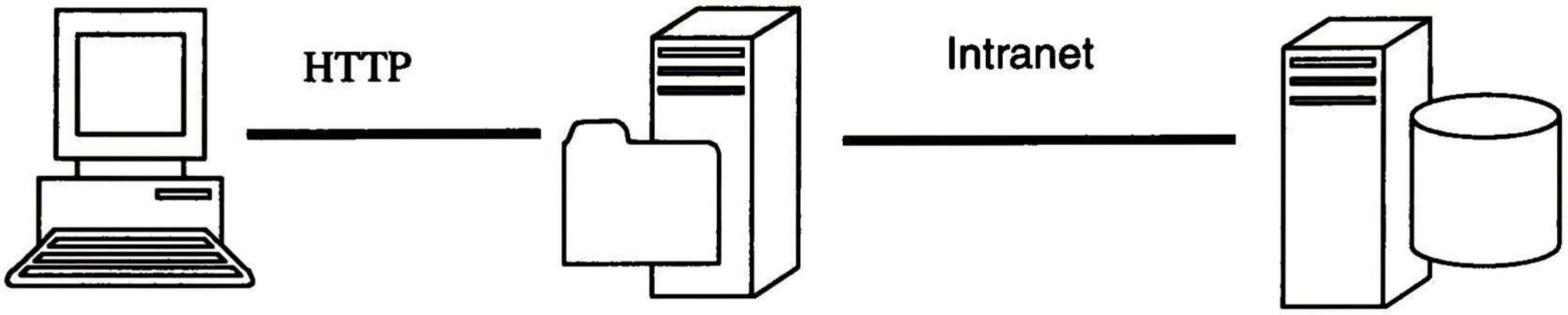


Figura 3.2 Estilo de arquitectura de cliente grueso.

3.2.4 Estilo de Arquitectura de Entrega para Web

En el estilo de arquitectura de entrega para Web además del protocolo de comunicación HTTP se utilizan protocolos como IIOP o DCOM para el soporte de objetos distribuidos, el navegador Web es principalmente utilizado como contenedor para la interfaz de usuario y algunos objetos del negocio que se comunican con otros objetos del lado del servidor mediante los protocolos antes mencionados. La estructura del estilo de arquitectura de entrega para Web se muestra en la siguiente figura.

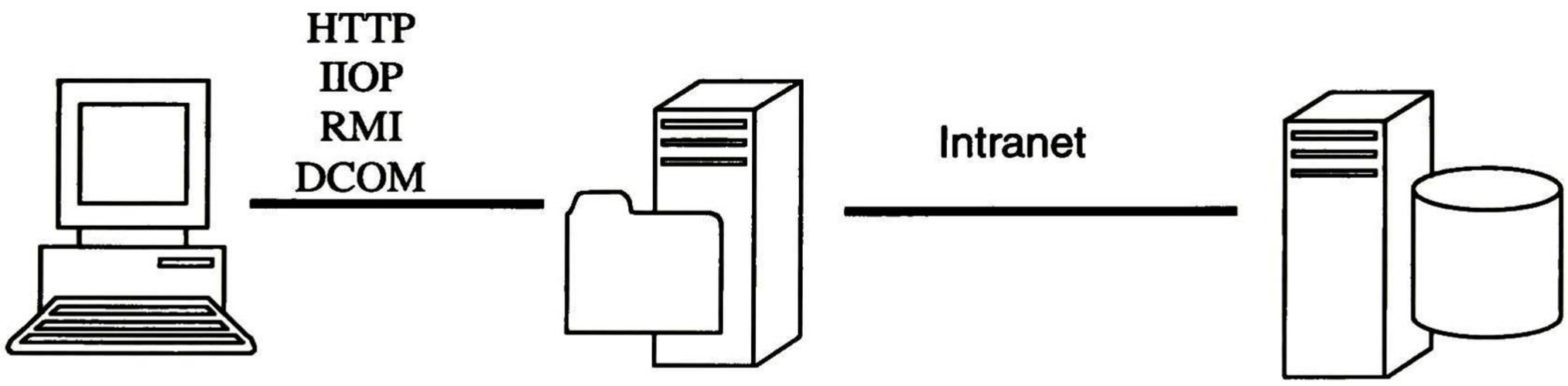


Figura 3.3 Estilo de arquitectura de Entrega para Web.

La mayoría de las aplicaciones para comercio electrónico utilizan el estilo de arquitectura de cliente delgado con el objetivo de alcanzar el mayor número posible de clientes.

3.3 Arquitecturas de Interfaz de Usuario

3.3.1 Introducción

En esta sección presentamos algunos modelos de interacción de usuario con la aplicación, estos modelos son de aplicación general y son tomados de [Baas et al, 1998].

3.3.2 Arquitectura Monolítica

Existen al menos dos funciones distintas que cualquier sistema con una interfaz de usuario debe proveer: *presentación* (interacción con el usuario) y *aplicación* (la principal función del sistema). También existen aspectos temporales y jerárquicos, la tarea principal a ser desarrollada por el usuario es descompuesta en subtareas, que son a su vez descompuestas en acciones físicas que el usuario desempeña en la presentación. Esta descomposición y secuencia puede ser tomada como un diálogo entre el usuario y la aplicación. Este diálogo es un tercer tipo de función que todo sistema interactivo debe soportar.

En el modelo monolítico todas las funciones del sistema están agrupadas en un solo módulo. El modelo monolítico es simplemente un modelo de implementación. Es ampliamente utilizado ya que requiere de poco análisis y proporciona un buen grado de eficiencia, las funciones de aplicación e interfaz están directamente conectadas y no incurren en la sobrecarga de las capas de comunicación entre componentes.

Figura 3.4 Asignación de la funcionalidad en el modelo monolítico.

3.3.3 El Modelo Seeheim para Software de Interfaz de Usuario

Desde el punto de vista de facilidad de modificación y facilidad de transportación, cualidades que son ampliamente requeridas por la parte del sistema encargada de la interfaz de usuario, el modelo Seeheim ofrece varias ventajas.

Esta arquitectura hace uso principalmente de la operación de abstracción ya que esta cumple con las cualidades de facilidad de modificación y facilidad de transportación. En el caso del dominio de las interfaces de usuario, pueden identificarse dos tipos de modificación: reemplazar la presentación (que es muy común) y reemplazar la aplicación (lo cual es menos frecuente).

Tres de las operaciones unitarias pueden incrementar la facilidad de transportación y facilidad de modificación de un sistema: compartir los recursos, la descomposición y la abstracción. La descomposición no es aplicable en este caso ya que su dominio de aplicación es un solo componente. En la mayoría de las arquitecturas de interfaz de usuario la presentación es un recurso compartido (como el servidor en el sistema X-Window).

Con el propósito de aislar las modificaciones en el diálogo (típicamente la parte más afectada de un sistema interactivo) o la aplicación, los diseñadores recurren a la operación de separación, aislando el diálogo en un componente separado. Este modelo fue presentado originalmente en [Pfaff, 1985].

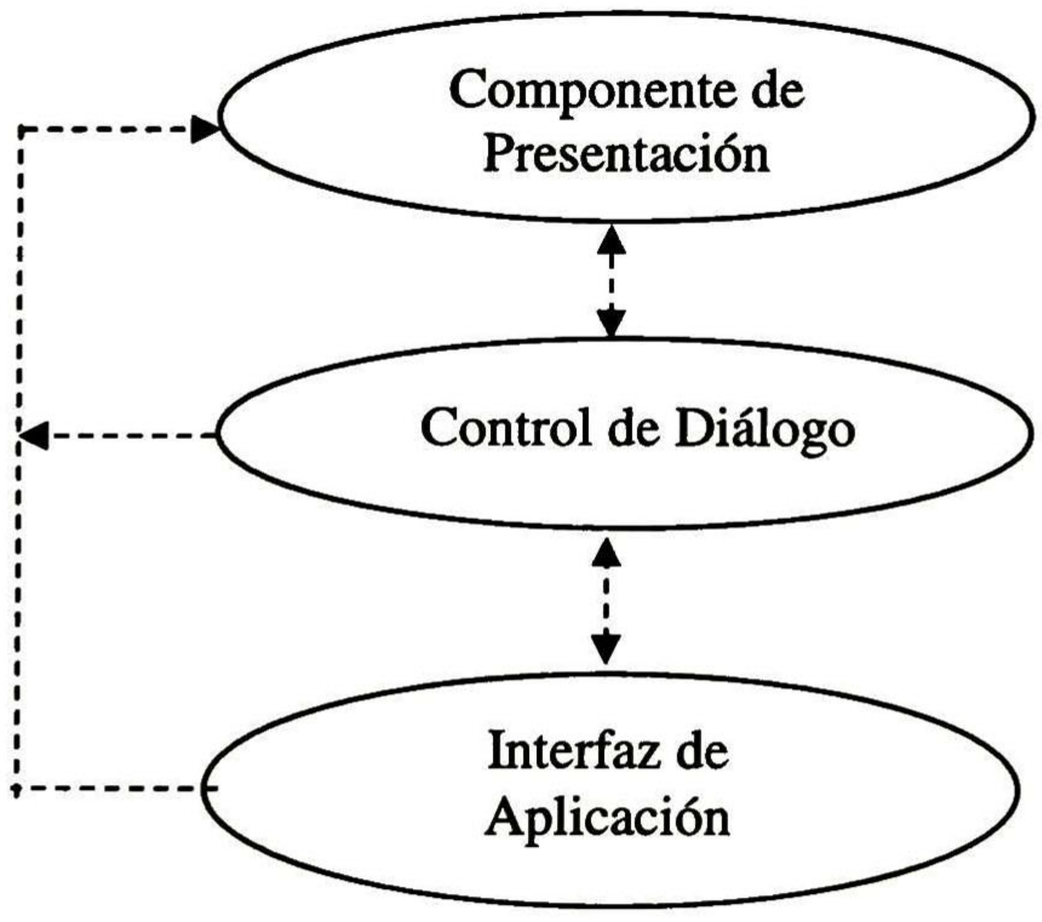


Figura 3.5 Modelo lógico Seeheim.

3.3.4 **Modelo-Vista-Controlador**

El paradigma Modelo-Vista-Controlador (MVC) obtiene el mismo conjunto de cualidades que el modelo Seeheim pero de manera diferente, estableciendo así un modelo de referencia distinto.

Partiendo del modelo monolítico y considerando la facilidad de modificación como cualidad del sistema, el Modelo-Vista-Controlador aplica la operación de descomposición, lo cual limita el alcance de las modificaciones.

La siguiente consideración es que el sistema debe soportar diferentes dispositivos de entrada y salida. Para lograr este objetivo la operación de separación es aplicada a la parte de presentación de cada uno de los componentes identificados. La intención es separar los dispositivos de entrada y salida no sólo del diálogo y la aplicación en uno, sino en todos los componentes. El resultado de estas operaciones es la versión original del modelo, la vista (salida), y el controlador (entrada). Las ventajas de la arquitectura MVC se entiende del conjunto consistente de operaciones aplicadas para obtener las cualidades requeridas. Una descripción detallada de este modelo se encuentra en [Krasner, 1988].

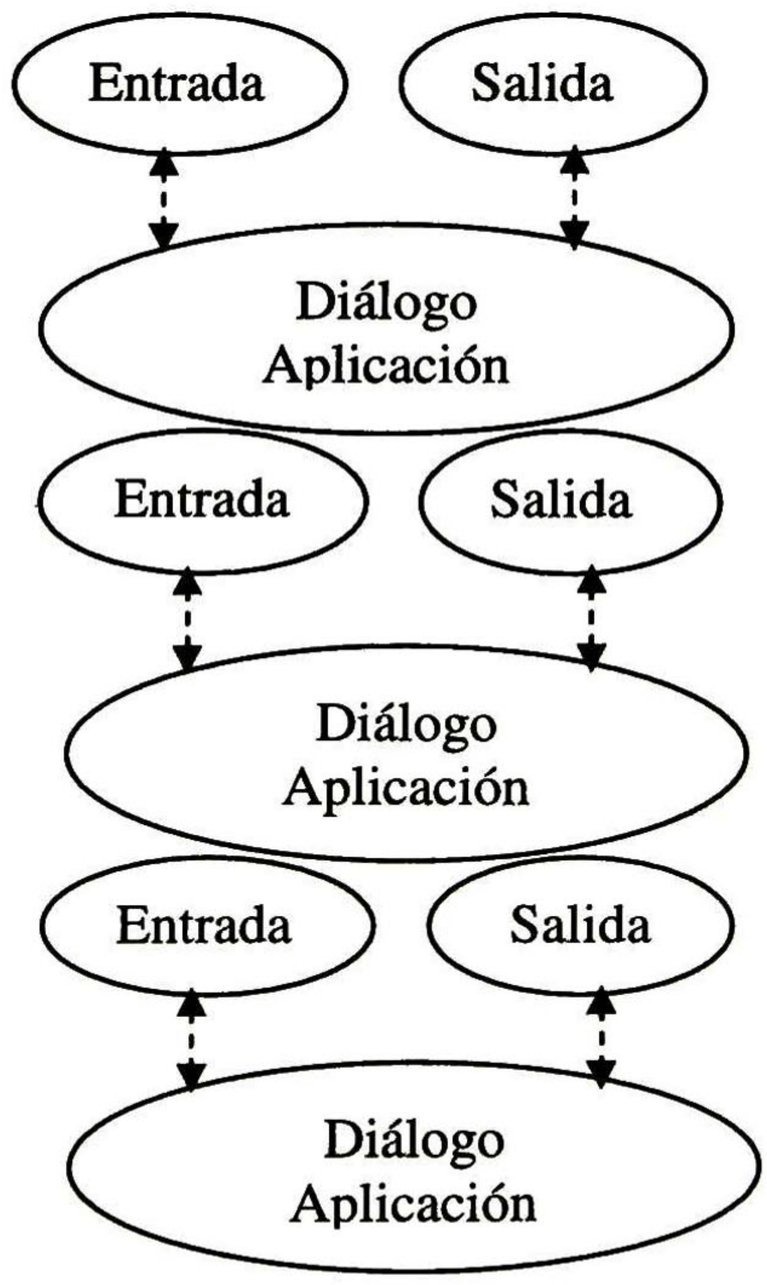


Figura 3.6 El paradigma Modelo-Vista-Controlador.

3.4 Documentos en Internet

3.4.1 Introducción

En las siguientes secciones presentamos las diferencias entre los tipos de documentos que pueden encontrarse en Internet y mencionamos algunas tecnologías que los soportan.

3.4.2 Documentos Estáticos

Los primeros documentos que aparecen en Internet tienen como característica principal que su información permanece constante si no son modificados directamente por el autor del documento. El despliegue de un documento se lleva a cabo en el navegador; mediante el Lenguaje de Marcado de Hiper Texto (HTML) se expresa el contenido y el formato visual de una página Web, HTML es un lenguaje de etiquetas basado en el Lenguaje de Marcado Generalizado Estándar (SGML), el cual es un lenguaje mucho más amplio utilizado para definir lenguajes de marcado de propósito particular. HTML es sólo una aplicación específica del SGML, adecuada para la presentación de documentos textuales. HTML contiene etiquetas que definen el formato al desplegarse el texto: fuente, tamaño, color, etc. HTML es un estándar que evoluciona administrado por el grupo de estándares del World Wide Web Consortium (W3C)[8]. Una de las desventajas de utilizar HTML es que los documentos son *estáticos*, es decir, una vez que los documento son almacenados en el servidor, este los localiza en el disco duro y los distribuye a los navegadores que se conectan a él sin ningún cambio. La mayor ventaja de este tipo de documentos es que no se requieren conocimientos especiales de programación y cualquier persona con un editor de páginas es capaz de crear sus propios documentos y ponerlos en línea. Además las páginas estáticas no requieren de tantos recursos como las páginas dinámicas.

3.4.3 Documentos Dinámicos

El tipo de documento visto en la sección anterior tiene algunas desventajas. Por lo general, la mayoría de las compañías eventualmente requieren que sus páginas se actualicen dinámicamente de acuerdo con la información de su base de datos. Al utilizar este enfoque, las *páginas dinámicas* permiten a los visitantes añadir, insertar y borrar datos, mientras que los datos están disponibles inmediatamente. Las páginas dinámicas son mucho más flexibles y útiles que las páginas estáticas; sin embargo, estas pueden consumir mucho más recursos del servidor. Por ejemplo, considere una tienda virtual

---

[8] La especificación completa de HTML puede encontrarse en http://www.w3c.org.

que cuenta con 5,000 artículos. Al utilizar páginas estáticas, es necesario crear una página por cada producto; es decir, serán necesarias 5,000 páginas, las cuales son muy difíciles de mantener. Al utilizar páginas estáticas la página completa necesita ser editada, aún si sólo una línea desea ser cambiada. La sobrecarga de crear páginas estáticas puede consumir más recursos que las páginas dinámicas. Para poder generar dinámicamente el contenido de una página, existen muchas tecnologías disponibles, una de ellas es utilizar CGI (Common Gateway Interface), que nos proporciona una interfaz para escribir programas del lado del servidor, mediante los cuales se obtiene la información actualizada de las bases de datos. Estos programas pueden ser escritos en diferentes lenguajes tanto compilados como interpretados (como C y PERL[9]). Otro de los lenguajes de programación utilizado comúnmente es Java, lenguaje independiente del hardware que es ejecutable en las principales plataformas. La mayoría de la gente conoce Java en la forma de applets ejecutándose en el navegador. La tecnología JavaServer Pages (JSP) proporciona una forma fácil y eficiente de crear páginas de contenido dinámico. Esto hace posible el desarrollo rápido de aplicaciones que son independientes de la plataforma del servidor. La lógica de la aplicación se localiza en los recursos del servidor que la página utiliza a través de etiquetas HTML. Debido a la separación del diseño de la página de la generación de contenido, además del soporte de diseño basado en componentes reutilizables, la tecnología JSP habilita la construcción de aplicaciones dinámicas e interactivas.

3.4.4 **Documentos Activos**

Actualmente es posible llevar a cabo procesamiento mediante el uso de componentes incluidos en los documentos o páginas Web, llamaremos *documentos activos* a este tipo de documentos. Para ello existen varias tecnologías, se puede realizar algún tipo de procesamiento en el cliente mediante scripts[10]; por ejemplo, varios documentos utilizan JavaScript para validar el contenido de los formularios HTML. También pueden utilizarse componentes como los applets o JavaBeans escritos en lenguaje Java, o los componentes ActiveX de Microsoft. Mediante estos componentes pueden construirse interfaces gráficas sofisticadas o servir como medio de comunicación con otros componentes del lado del servidor utilizando diferentes protocolos de comunicación. Para poder hacer uso de estos tipos de componentes es necesario tener mayor capacidad de procesamiento en la máquina cliente y navegadores que soporten la tecnología; por lo cual, es menor el número de documentos de este tipo, que los mencionados anteriormente.

---

[9] PERL (practical extraction and reporting language) .
[10] Fragmentos de código incrustados en páginas HTML.

3.5 Intercambio de Datos

3.5.1 Introducción

En esta sección presentaremos algunas tecnologías para el intercambio de datos entre aplicaciones y presentación de documentos, enfocándonos principalmente en XML.

3.5.2 Lenguaje de Marcado Extensible (XML)

3.5.2.1 Introducción a XML

La principal razón por la que fue creado XML es proporcionar una manera flexible para el intercambio de datos entre sistemas heterogéneos. También puede ser utilizado para separar el contenido y la presentación de documentos, así como para la definición de archivos o formatos de configuración de aplicaciones. En el caso de la interfaz de usuario, el Lenguaje de Marcado Extensible (XML) ofrece a los desarrolladores de aplicaciones Web otra forma de expresar el contenido de sus páginas. Uno de los problemas principales con HTML es que los detalles del contenido y la presentación están incluidos en el mismo documento. Los documentos XML no contienen información de presentación; en cambio hacen referencia a un documento XSL[11] o CSS[12] con el propósito de desplegar el documento en el navegador. En la mayoría de los casos, los documentos XML son utilizados para la administración y comunicación de datos entre sistemas de distintas plataformas. XSL y CSS son tratados en las siguientes secciones.

3.5.2.2 El Lenguaje de Hoja de Estilo Extensible (XSL) y la Hoja de Estilo de Cascada (CSS)

El Lenguaje de Hoja de Etilo Extensible (XSL) y la Hoja de Estilo de Cascada (CSS) son utilizados principalmente para la presentación y definición de estilo de documentos. Una de las funciones del Lenguaje de Hoja de Estilo Extensible es transformar y traducir datos de un formato XML a otro. Por ejemplo, el mismo documento en XML puede ser desplegado en un formato HTML, PDF, o Postscript. Sin XML el documento tendría que ser duplicado manualmente y después convertido a cada uno de los tres formatos. En cambio, XSL proporciona un mecanismo para definir hojas de estilo para cumplir con

---

[11] Extensible Stylesheet Language.
[12] Cascading Style Sheet.

este tipo de tareas. En lugar de tener que cambiar los datos para una diferente presentación, XSL proporciona una separación completa de datos o contenido y la presentación. Si un documento XML necesita ser desplegado en otra presentación, entonces XSL proporciona una buena solución.

El uso principal de las Hojas de Estilo de Cascada es la definición de estilos para la presentación de documentos HTML. CSS define como desplegar elementos HTML. Los estilos están normalmente almacenados en hojas de estilo y fueron añadidos a HTML 4.0. Las hojas de estilo externas pueden ahorrar mucho trabajo, estas se almacenan en archivos CSS. CSS es un gran avance en el diseño Web ya que permite a los desarrolladores controlar el estilo y la distribución de múltiples páginas Web. Es posible definir un estilo para cada elemento HTML y aplicarlo a cuantas páginas Web se requiera. Para hacer un cambio global, simplemente se cambia el estilo, y todos los elementos de la página Web serán actualizados automáticamente.

3.5.2.3 Lenguaje de Ligado XML (XLL)

Esta sección, como la siguiente, trata sobre elementos XML definidos específicamente para la descripción de ligas o enlaces. El Lenguaje de Ligado XML (XLL[13] o XLink) especifica elementos que pueden ser insertados en recursos XML para describir enlaces entre objetos de datos. Utiliza sintaxis XML para crear las estructuras que pueden describir no sólo enlaces unidireccionales sino también enlaces múltiples y enlaces con tipo.

Un enlace es una relación explícita entre dos o más objetos de datos o porciones de ellos. Un elemento de enlace es utilizado para describir las características del enlace. Los elementos del enlace son reconocidos basándose en el uso de un atributo designado como *xml:link*, los posibles valores son *simple* y *extended*. Un elemento que incluye tal atributo debe ser tratado como un elemento de enlace del tipo indicado. El siguiente es un ejemplo similar a un enlace <*A*> en HTML:

<A xml:link="simple" href="http://www.w3.org/">The W3C</A>

La especificación de XLink puede encontrarse en *http://www.w3.org/TR/xlink*.

---

[13] Extensible Linking Language.

3.5.2.4 Lenguaje Apuntador XML(XPL)

Otra de las extensiones para la descripción de elementos o atributos internos en documentos XML es XPL[14] o XPointer. XPointer especifica elementos que dan soporte al direccionamiento interno de los documentos XML. Este proporciona referencias específicas a elementos, atributos, cadenas de caracteres, y otras partes de documentos XML. Los XPointer pueden ser utilizados como identificadores de fragmentos que especifican un recurso más preciso. Cualquier identificador de fragmento que apunte a un recurso XML debe ser un XPointer. Los XPointer trabajan sobre el árbol definido por los elementos de un documento XML. Un XPointer consiste de una serie de términos de localización, cada término de localización tiene una clave o identificador, que tiene argumentos como número de instancia, tipo de elemento, o atributo. Muchos de estos términos localizan nodos individuales en un árbol. De cualquier manera, algunos términos de localización pueden localizar conjuntos más complejos de datos. Los términos de localización son clasificados en términos absolutos, términos relativos, términos de amplitud, términos de atributos y términos de datos.
La especificación de XPointer puede encontrarse en: *http://www.w3.org/TR/xptr/*.

3.5.2.5 XML como Metalenguaje (XMI)

Un *modelo de metadatos* define la estructura y el significado de los datos, es decir, nos proporciona información acerca de cómo interpretar los datos y la forma en que están estructurados. Regularmente el intercambio de información entre aplicaciones o herramientas de diferentes proveedores es difícil debido a que manejan diferentes formatos de entrada que requieren ser traducidos o introducidos manualmente para ser utilizados por otras aplicaciones. XML para el Intercambio de Metadatos (XMI[15]) es una propuesta de la OMG[16], cuyo propósito principal es facilitar el intercambio de datos entre herramientas y entre almacenes de datos en ambientes heterogéneos distribuidos. XML define un mecanismo para validar un documento XML, un documento de definición de tipos (DTD[17]) especifica la estructura lógica de un documento XML mediante una gramática. La especificación XMI consiste principalmente en:

1 Un conjunto de reglas de producción XML DTD para transformar metamodelos.
2 Un conjunto de reglas de producción de Documentos XML para codificar y transferir metadatos.
3 Principios de diseño para DTD's que cumplen con XMI.

---

[14] Extensible Pointer Language.
[15] XML Metadata Interchange.
[16] Object Management Group.
[17] Document Type Definition.

4 Principios de generación para documentos XML que cumplen con XMI.

El objetivo de la especificación es el desarrollo de herramientas de importación y exportación que permitan el intercambio de datos entre aplicaciones.

3.5.2.6 Modelo de Objetos de Documentos (DOM) y la API para XML (SAX)

El Modelo de Objetos de Documento (DOM[18]) y la Interfaz de Programación de Aplicación (API) para XML de Java (SAX[19]), son utilizadas para procesar documentos XML. DOM representa un documento XML en forma de árbol, en donde los nodos del árbol representan los elementos del documento. Mientras que la API SAX está basada en eventos. SAX sólo proporciona acceso a los datos en un documento XML, DOM está diseñado como un medio para la manipulación de estos datos. DOM proporciona una presentación de un documento de XML como un árbol. Dado que un árbol es una estructura de datos conocida, los recorridos y la manipulación de estas estructuras son fáciles de implementar en los lenguajes de programación, Java no es la excepción. DOM lee un documento XML completo en la memoria, almacena todos los datos en nodos, por lo cual el acceso a ellos es muy rápido; ya que permanece en la memoria mientras existe el árbol. Cada nodo representa una parte de los datos extraída del documento original.

Sin embargo, existe una importante desventaja de DOM. Como consecuencia de que DOM lee un documento entero en memoria, puede consumir grandes recursos, en ocasiones deteriorando el desempeño y ocasionando aún la terminación anormal de una aplicación. Entre más largo y más complejo es un documento, la degradación del desempeño es mayor. Teniendo en cuenta que mientras DOM es el medio prevaleciente para la modificación de datos XML, no es el único medio para lograr esto. La especificación de DOM puede encontrarse en *http://www.w3.org/DOM*.

SAX proporciona una estructura basada en eventos para el análisis sintáctico de datos XML, mediante el proceso de leer a través del documento y descomponerlo en partes útiles; en cada paso SAX define los eventos que pueden ocurrir. Por ejemplo, SAX proporciona una interfaz *org.xml.sax.ContentHandler* que define los métodos *startDocument()* y *endElement()*; la implementación de esta interfaz permite un control completo sobre estas partes del proceso de análisis de XML. Existe una interfaz similar para el manejo de errores y elementos léxicos. También se define un conjunto de errores y advertencias que hacen posible el manejo de las situaciones que pueden presentarse durante el análisis, tal como un documento no válido, o uno que no está bien formado. Se puede añadir comportamiento para personalizar el proceso de análisis que da como

---

[18] Document Object Model.
[19] Simple API for XML

resultado tareas específicas a una aplicación, utilizando una interfaz estándar en documentos XML. La documentación sobre la API puede encontrarse en *http://www.megginson.com/SAX*.

Capítulo 4 **Arquitectura de Software del Sistema para Comercio Electrónico**

4.1 **Introducción**

Una contribución importante de esta tesis es el uso de los conceptos de arquitectura de software antes expuestos para implementar parte de un sistema de comercio electrónico. El sistema de comercio electrónico está basado en los siguientes documentos:

i) Documento de Especificación de Requerimientos.
ii) Documento de Análisis.
iii) Documento de Diseño.
iv) Documento de Descripción de la Arquitectura de Software.

Los documentos de Especificación de Requerimientos y Análisis fueron elaborados de manera conjunta por: Janet Reyes, Alberto M. Cortés, Rogelio André y Jaime A. Cedillo. El documento de diseño trata específicamente sobre la interfaz de usuario de la aplicación o cliente del sistema, el diseño de los componentes del negocio que se encuentran en el servidor es tema de otra tesis de maestría [Reyes, 2003]. Estos documentos son apéndices a esta tesis, se usan sólo como referencia y no forman parte de la misma, sin embargo, forman la definición sobre la cual se basa la arquitectura, diseño e implementación que aquí se presentan.

Comentarios generales acerca del modelo de casos de uso. En el documento de especificación de requerimientos se encuentra el modelo de casos de uso. En cada caso de uso se describe el flujo básico de una interacción típica con el usuario, flujos alternativos, precondiciones, poscondiciones y requerimientos especiales específicos al caso de uso.

Comentarios generales acerca de modelo de análisis. En el documento de análisis se encuentra el modelo de análisis que incluye una descripción de las clases de análisis, la realización de los casos de uso en la etapa de análisis (diagrama de clases, diagrama de colaboración, flujo de eventos, requerimientos especiales), la descripción de la división de paquetes de análisis y una descripción de la arquitectura para esta etapa, en particular la perspectiva lógica de los componentes de software que conforman el sistema.

Comentarios generales acerca de modelo de diseño. En el documento de diseño se encuentra el modelo de diseño específicamente para la interfaz de usuario o cliente del sistema que incluye una descripción de las clases de diseño, la realización de los casos de uso en la etapa de diseño (diagrama de clases, diagramas de secuencia), la descripción de la división de subsistemas de diseño y las interfaces.

Comentarios generales acerca de la descripción de arquitectura. Cada uno de los modelos de casos de uso, análisis, diseño, emplazamiento e implementación incluyen una descripción o perspectiva de la arquitectura para el modelo en particular. También existe un documento de especificación de arquitectura que reúne las perspectivas mencionadas y una descripción de las cualidades de la misma.

Para una descripción detallada de los casos de uso mencionados a través de los diferentes modelos del proceso de desarrollo, refiérase al documento correspondiente. En las siguientes secciones nos enfocamos a la descripción de la arquitectura general del sistema.

4.2 Arquitectura de Alto Nivel

En las siguientes secciones se examina la arquitectura de software de la aplicación desde una perspectiva de alto nivel, enfocándose primero en los objetivos o metas de diseño, seguida por una descripción de los módulos funcionales y la división en capas de la aplicación. También se describen los estilos de arquitectura de software utilizados en la implementación del prototipo del sistema.

4.3 Cualidades de Diseño

Esta aplicación de comercio electrónico le permite al usuario consultar productos en venta y realizar pedidos de los mismos, así como registrarse en el sistema, administrar la información de su perfil de usuario, identificarse para tener acceso a información personal, entre otras funciones típicas de este tipo de aplicaciones. Todo esto es llevado a cabo por el usuario mediante un navegador o visualizador Web.

Para soportar estas funciones, se tuvieron en mente varias metas o cualidades de diseño:

Reutilización de código. La reutilización de código disminuye el costo de nuevos productos, permite mejoras de la calidad y el establecimiento de buenas prácticas de programación en la organización de desarrollo.

Descomposición funcional. Cada clase en el sistema tiene un rol bien definido en el sistema. Lo cual facilita el mantenimiento, la extensión y entendimiento del sistema.

Facilidad de extensión. La funcionalidad de la aplicación puede extenderse de acuerdo a los cambios tecnológicos y de organización que se presenten.

Diseño modular. La descomposición del diseño en módulos que tienen interacción a través de interfaces bien definidas, permite a los desarrolladores trabajar independientemente, facilita el mantenimiento y las pruebas, y la adquisición de componentes desarrollados externamente.

Seguridad. Dado que la aplicación realiza transacciones financieras, y como instrumento para la privacidad y seguridad de los clientes, la seguridad de la información es una parte importante del sistema.

Facilidad de uso. Las aplicaciones son más fáciles de usar, si el usuario puede intuir donde encontrar la información que desea.

Minimizar el tráfico de datos. La aplicación debe evitar transmitir datos no necesarios o redundantes.

Persistencia. La persistencia de los datos debe ser consistente siempre.

4.4 Estilos de Arquitectura Aplicados

El estilo o patrón de arquitectura de cliente delgado fue seleccionado para nuestra aplicación debido a que es el más apropiado para aplicaciones basadas en Internet en donde se requiere un mínimo de poder de procesamiento en el cliente o no se tiene control sobre su configuración, obsérvese que éste es nuestro caso.

Los principales componentes del patrón de arquitectura de cliente delgado son los siguientes:

- ***Navegador o cliente.*** Se supone aquí cualquier navegador que soporte formularios HTML. El navegador actúa como medio generalizado para la interfaz de usuario. El usuario de la aplicación utiliza el navegador para solicitar

las páginas al servidor Web. La página que regresa el servidor contiene la interfaz de usuario completa – texto y controles de edición - que se despliega en el navegador ejecutándose en la máquina cliente. Toda interacción con el sistema es llevada a cabo por el usuario a través del navegador.

- ***Servidor Web.*** Es el principal punto de acceso para los navegadores o clientes. Los navegadores tienen acceso al sistema solamente a través del servidor Web; este acepta solicitudes de páginas – ya sean estáticas (HTML) o páginas de servidor[20] Dependiendo de la solicitud, el servidor Web puede iniciar el procesamiento, como en el caso de las solicitudes de páginas de servidor.
- ***Conexión HTTP[21].*** Representa el medio de comunicación entre el cliente y el servidor, este tipo de conexión no mantiene el estado de la sesión. Cada vez que el cliente o el servidor envía información, una conexión nueva es establecida. Una variación de la conexión HTTP es la conexión segura HTTP implementada mediante el protocolo SSL[22]. Este tipo de conexión codifica la información que se transmite entre el cliente y el servidor.
- ***Página HTML.*** Contiene la interfaz de usuario y el contenido que no necesita un procesamiento previo en el servidor. Cuando el servidor Web recibe una solicitud de la página, este simplemente localiza el archivo y lo envía al cliente.
- ***Página de servidor.*** Este tipo de páginas requiere un procesamiento previo por parte del servidor para la generación de contenido dinámico, estas páginas tienen acceso a los recursos del servidor, incluyendo los componentes de la lógica de negocio, bases de datos y sistemas externos.
- ***Servidor de Aplicación.*** El servidor de aplicación es el responsable de ejecutar el código de las páginas de servidor, puede estar localizado en la misma máquina que el servidor Web o incluso en el mismo proceso. Debido a que su función principal es la ejecución de la lógica del negocio puede usarse una tecnología independiente del servidor Web.

Algunos componentes adicionales de este estilo de arquitectura tienen que ver con la persistencia de los datos y los componentes que encapsulan la lógica del negocio en el servidor. Debido a que la mayoría de navegadores en el mercado soportan los lenguajes de script en el cliente para el mejoramiento de las interfaces de usuario, también se ha incluido un componente para ello.

El estilo de arquitectura de cliente delgado es el más apropiado para las aplicaciones que requieren tiempos de respuesta aceptables para las solicitudes de los clientes. La figura 4.1 muestra un diagrama de la perspectiva lógica de la arquitectura de cliente delgado.

---

[20] Server Pages, utilizadas para la generación de contenido dinámico.
[21] HyperText Transfer Protocol.
[22] Secure Sockets Layer.

Otro estilo de arquitectura aplicado es el estilo Modelo-Vista-Controlador, en donde se separa la interfaz del usuario, de los elementos de control y las entidades que representan el modelo de la aplicación. La aplicación de este estilo de arquitectura se adapta bien al tipo de aplicaciones divididas en capas. Entre los beneficios de este estilo de arquitectura se encuentra un desarrollo y mantenimiento más fácil de los componentes, así como el soporte para nuevos tipos de clientes.

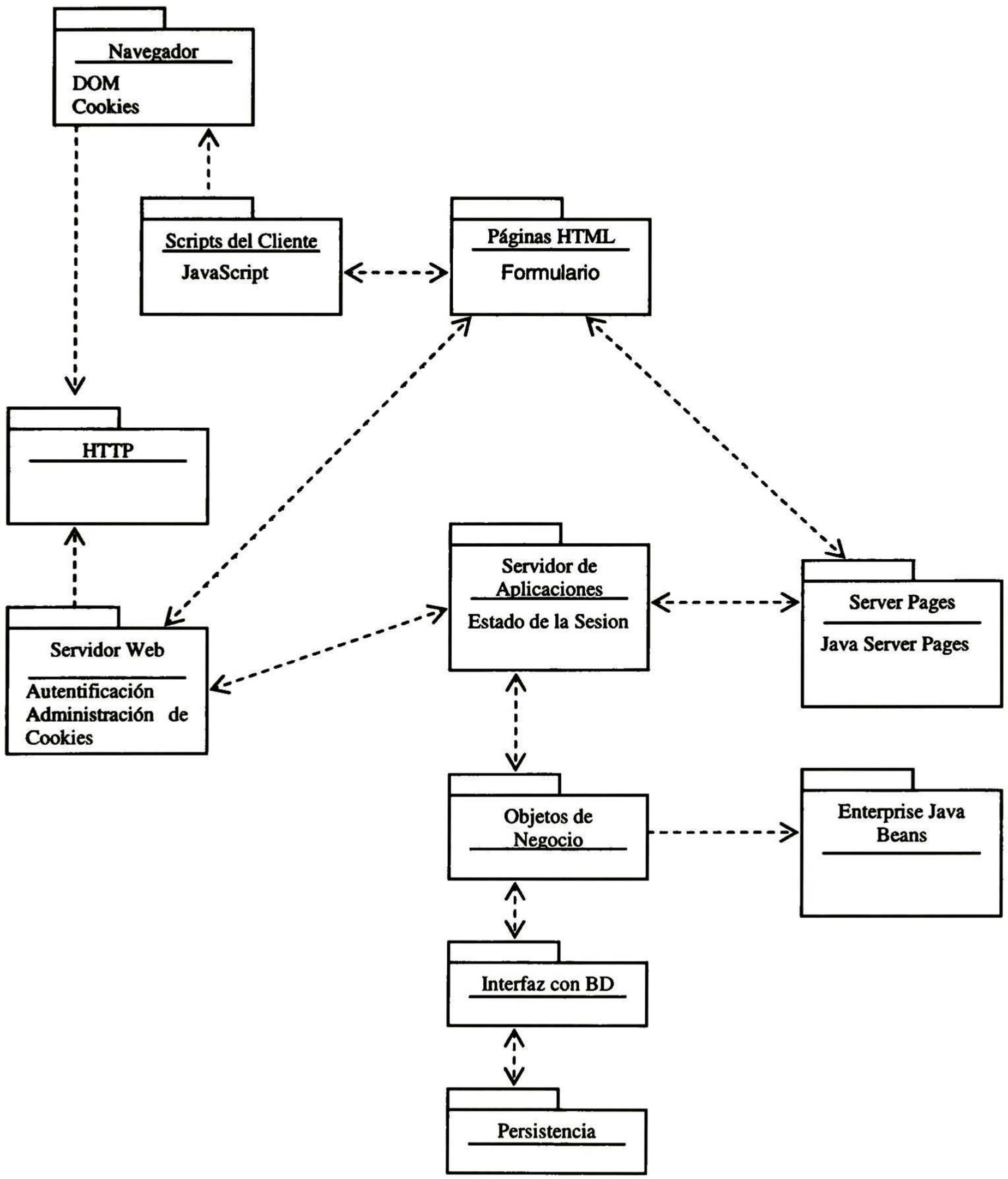


Figura 4.1 Perspectiva lógica del estilo de arquitectura de cliente delgado.

4.5 División de la aplicación

La división de la aplicación se realiza de dos formas. La primera partición divide la aplicación en múltiples capas, separando la funcionalidad de los clientes de la aplicación, la funcionalidad del servidor de la aplicación y la funcionalidad de la persistencia de datos y sistemas externos. La segunda forma de partición es la tradicional descomposición modular de las funciones de la aplicación.

4.5.1 División en Capas

El diseño de la aplicación se encuentra dividido en varias capas: Cliente, Servidor, Datos. Estas capas no están ordenadas jerárquicamente, cada capa se comunica directamente con las demás o indirectamente a través de una capa intermedia. La división del diseño en capas permite a los diseñadores escoger la tecnología adecuada para cada situación. Múltiples tecnologías pueden utilizarse para proporcionar el mismo servicio en diferentes situaciones.

Capa del Cliente. La capa del Cliente es responsable de presentar los datos al usuario, tener interacción con el usuario y comunicarse con otras capas de la aplicación. Frecuentemente la capa del Cliente es la única que el usuario de la aplicación puede ver. La capa del Cliente de la aplicación consiste principalmente en un navegador que despliega páginas Web generadas por Java Server Pages en el servidor. Esta capa se comunica con otras capas a través de interfaces bien definidas y proporciona las cualidades de flexibilidad y facilidad de extensión. Debido a que la capa del Cliente principalmente ofrece funciones de presentación, se utilizó un estilo de arquitectura de cliente delgado.

Capa del Servidor. La capa del Servidor puede dividirse en dos: servidor Web y servidor de aplicación. El servidor Web es responsable de realizar todo el procesamiento relacionado con el servicio de páginas HTML y JSP para su presentación en el navegador. El servidor de aplicación es responsable de cualquier procesamiento o lógica de negocio utilizando Enterprise JavaBeans (EJB). Enterprise JavaBeans son componentes de software de negocio los cuales extienden la funcionalidad específica de la aplicación. La interfaz entre estos componentes y sus contenedores está definida en la especificación de EJB[23]. Típicamente los servidores implementan contenedores para EJB, estos contenedores proporcionan servicios como: control de transacciones, administración de seguridad y manejo de persistencia. Los desarrolladores de la

[23] http://java.sun.com/products/ejb.

aplicación diseñan componentes EJB enfocándose en la funcionalidad del negocio o lógica de la aplicación.

Capa de Datos. La capa de Datos conforma la infraestructura de información. En esta capa están incluidos sistemas de administración de bases de datos relacionales, sistemas de planeación de recursos (ERP), y otros sistemas externos. El acceso a esta capa se realiza a través de transacciones que mantienen la consistencia de los datos. Esta capa también debe garantizar la seguridad de la información y ofrecer cualidades de facilidad de extensión.

La figura 4.2 muestra la división en capas de la aplicación y las relaciones de dependencia que existen entre ellas.

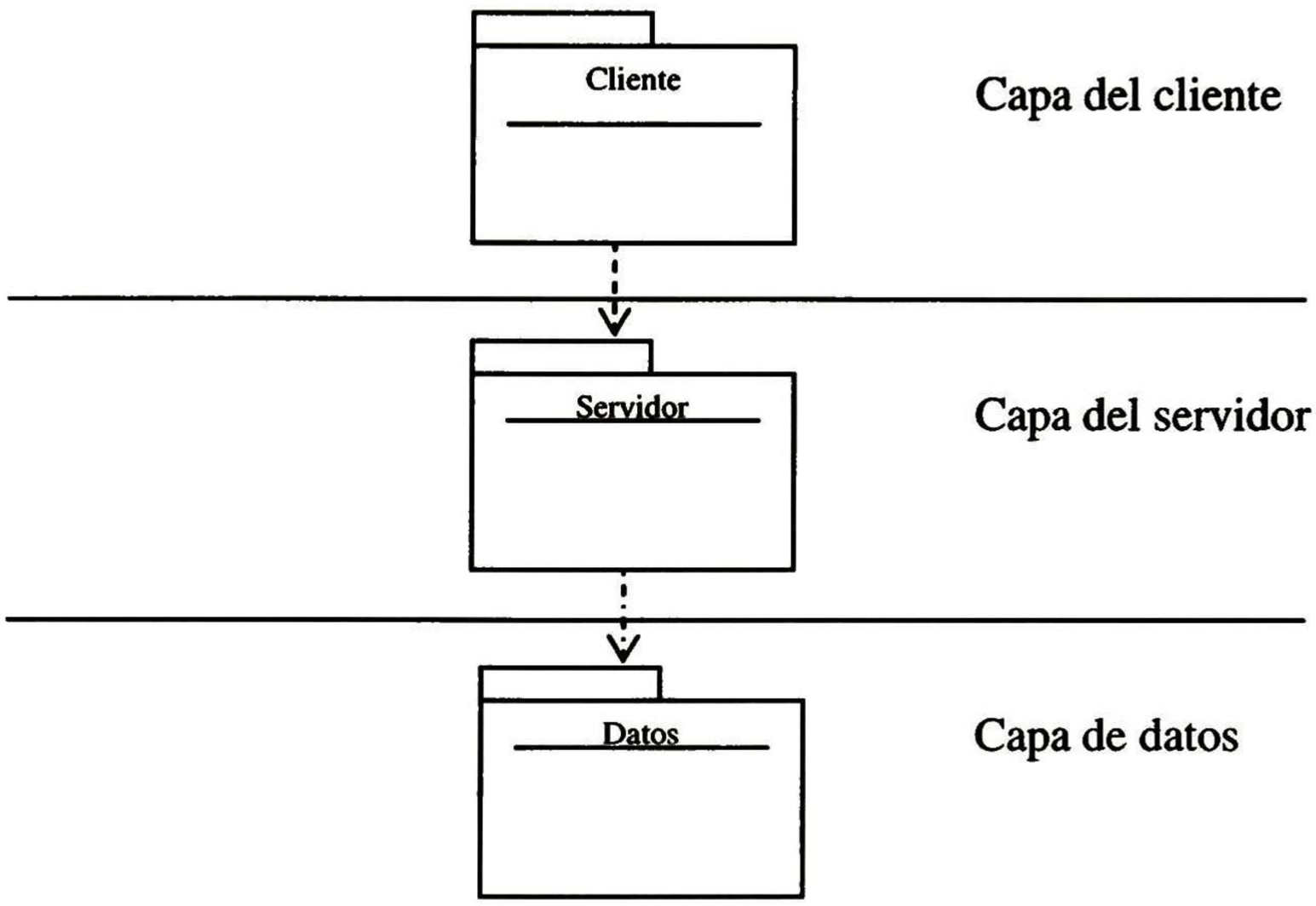


Figura 4.2 División en capas de la aplicación.

4.5.2 División Modular

La aplicación está diseñada como un conjunto de módulos, cada uno de los cuales presenta cohesión interna y un acoplamiento débil entre módulos. Estos módulos fueron identificados en la fase de análisis, donde grupos con similares datos y funcionalidad fueron relacionados en varios escenarios. Por ejemplo, varios escenarios incluyen el concepto de "catálogo de productos", de ahí que exista un módulo de *Catálogo* en la aplicación. La agrupación de funcionalidad en módulos proporciona integración entre las clases que cooperan constantemente, y ofrece poco acoplamiento entre clases que se comunican sólo ocasionalmente. De esta manera los cambios en los módulos pueden

realizarse de manera independiente sin afectar a un gran número de ellos. Cada módulo tiene una interfaz que define los requerimientos funcionales del módulo y proporciona un medio para la comunicación entre ellos. El diseño modular proporciona las cualidades de reutilización y descomposición funcional dado que cada módulo tiene un rol bien definido.

La figura 4.3 muestra los módulos de alto nivel de la aplicación y las relaciones de dependencias entre ellos.

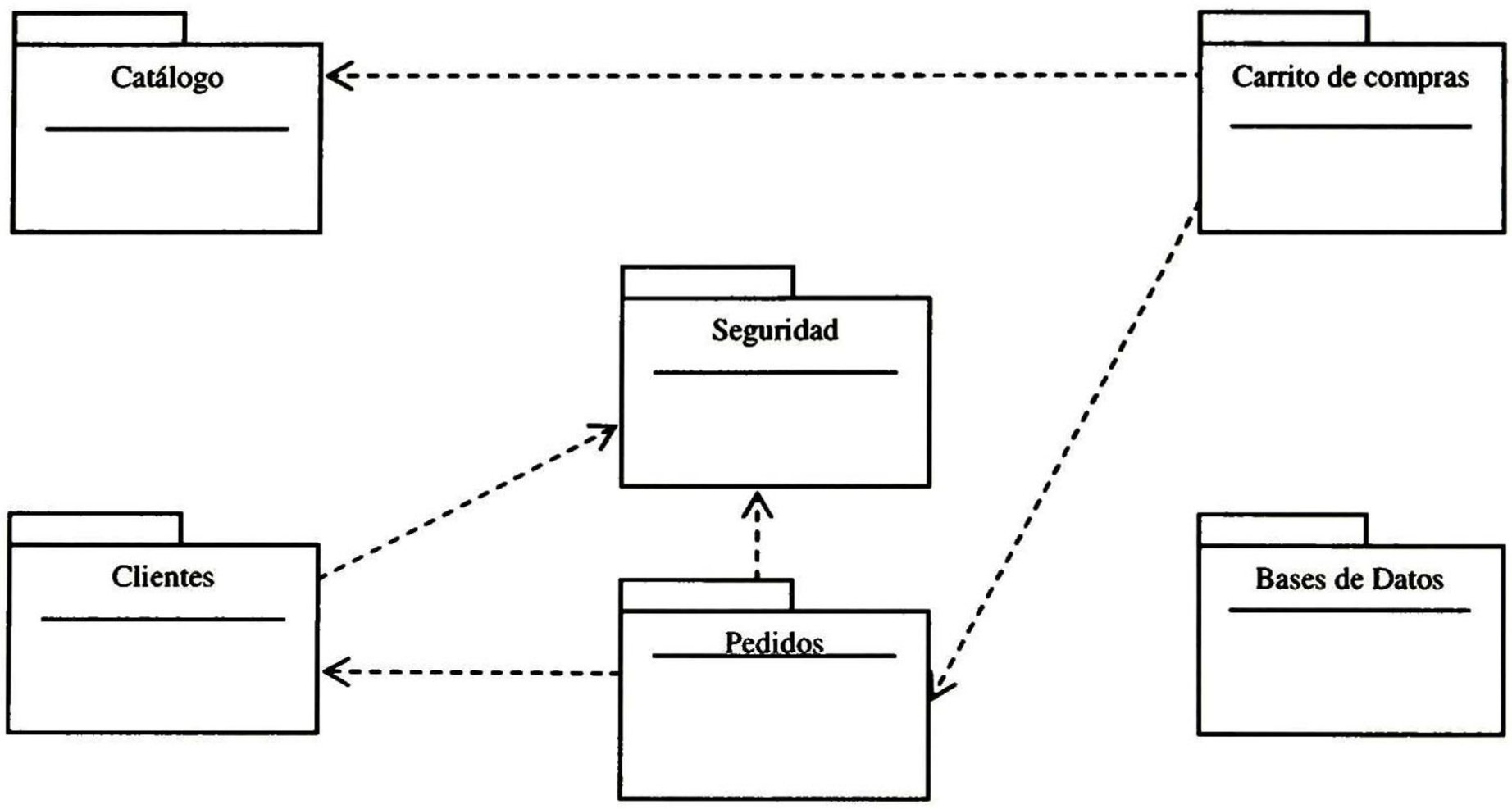


Figura 4.3 División modular de la aplicación.

La asignación de casos de uso a cada uno de los módulos se encuentra en el documento de análisis del sistema.

Capítulo 5 **Diseño del Cliente del Sistema**

5.1 **Introducción**

En las siguientes secciones se presentará la tecnología utilizada para la capa del cliente del sistema para comercio electrónico, así como las consideraciones de diseño de esta parte de la aplicación. Es necesario aclarar que el diseño mostrado en este capítulo se centra en la capa del cliente del sistema, específicamente en el diseño de la interfaz de usuario. El diseño de los componentes del servidor que encapsulan la mayor parte de la lógica de negocio de la aplicación, así como los aspectos de seguridad y el diseño de bases de datos del sistema no corresponden a esta tesis y son tratados en otras tesis de maestría [Reyes, 2003], [Cortés, 2003], [André, 2003].

5.2 **Tecnología Utilizada en el Cliente**

5.2.1 **JavaServer Pages (JSP)**

La tecnología JavaServer Pages (JSP) [JSP, 2000] proporciona una manera simple y rápida para crear páginas Web que despliegan contenido generado dinámicamente. La tecnología JSP fue diseñada para facilitar el desarrollo de aplicaciones Web que trabajan con una amplia variedad de servidores de aplicación, navegadores y herramientas de desarrollo.

Esta tecnología tiene varias ventajas sobre los problemas que presenta la interfaz CGI-BIN, donde los desarrolladores escriben programas individuales y las aplicaciones llaman a estos programas a través del servidor Web. Esta solución tiene problemas de consumo de recursos ya que cada solicitud CGI inicia un nuevo proceso en el servidor.

Si varios usuarios tienen acceso al sistema concurrentemente, estos procesos consumen la mayoría de los recursos disponibles en el servidor y el desempeño de la aplicación disminuye.

Otras soluciones tienen la desventaja de ser específicas a un servidor Web, como es el caso de la tecnología Active Server Pages (ASP) de Microsoft que facilita la creación de contenido dinámico, pero solamente trabaja con servidores de la misma empresa.

La tecnología Java Servlet facilita la creación de código para aplicaciones interactivas utilizando el lenguaje de programación Java. Un servlet es un programa que es ejecutado en el servidor, a diferencia de los applets que se ejecutan en el navegador. Los desarrolladores escriben servlets que toman una solicitud HTTP hecha por un navegador, generan la respuesta de manera dinámica (posiblemente consultando bases de datos) y regresan un documento HTML o XML. Utilizando este enfoque, la página debe escribirse completamente en el servlet. Si un desarrollador o diseñador quiere modificar la apariencia de la página, este tiene que editar y volver a compilar el servlet, aún si la lógica del programa funciona correctamente.

Las ventajas que proporciona trabajar con la tecnología JSP son las siguientes:

- Funciona con la mayoría de servidores Web o de aplicación.
- Separa la lógica de la aplicación de la apariencia o presentación de la página.
- Permite un desarrollo rápido y facilita las pruebas.
- Simplifica el proceso de desarrollo de aplicaciones interactivas basadas en Internet.

5.2.1.1 Desarrollo de Aplicaciones Web con Tecnología JavaServer Pages

La tecnología JSP facilita el desarrollo de páginas dinámicas en diferentes formas:

- ***Separación de la generación de contenido y la presentación.*** Con la tecnología JSP, los desarrolladores utilizan etiquetas HTML o XML para el diseño de páginas. También utilizan etiquetas JSP o scriptlets[24] para generar el contenido dinámico de la página. La lógica que genera el contenido es encapsulada en etiquetas y componentes JavaBean utilizados en scriptlets, los cuales son ejecutados en el servidor. Como la lógica se encuentra en etiquetas y componentes, entonces los diseñadores de páginas pueden editar y modificar la presentación sin afectar la generación de contenido. En el servidor, el procesador

[24] Fragmentos de código insertados en la página JSP.

JSP interpreta y ejecuta las etiquetas y scriptlets generando el contenido requerido utilizando componentes JavaBean o consultando bases de datos con tecnología JDBC[25] y envía el resultado en forma de un documento HTML o XML al navegador.

- ***Utilización de componentes reutilizables.*** Las páginas JSP pueden utilizar componentes JavaBeans o Enterprise JavaBeans para realizar el procesamiento de la aplicación. Los desarrolladores pueden compartir e intercambiar componentes para ejecutar operaciones comunes o ponerlos en disposición de otros desarrolladores. El enfoque basado en componentes agiliza el proceso de desarrollo y permite a las organizaciones optimizar la experiencia y los esfuerzos de desarrollo.

- ***Simplificación del desarrollo de páginas con etiquetas.*** La tecnología JavaServer Pages encapsula la mayor parte de la funcionalidad requerida para la generación de contenido en etiquetas JSP. Las etiquetas JSP estándar pueden acceder e instanciar componentes JavaBean, establecer o consultar atributos del componente, descargar applets, y desempeñar otras funciones que de otra manera sería más difícil y lento codificar. La tecnología JSP es fácil de extender a través del desarrollo de bibliotecas de etiquetas personalizadas. Esto permite a los desarrolladores trabajar con construcciones familiares, como las etiquetas, para realizar funciones complejas.

La tecnología JSP se integra fácilmente en una amplia variedad de arquitecturas de aplicación, y es parte de la arquitectura J2EE[26] para el soporte de aplicaciones empresariales.

5.2.1.2 Componentes de una JavaServer Page

Una página JSP tiene muchas características en común con una página HTML o XML estándar, con elementos adicionales que el procesador JSP se encarga de interpretar. Típicamente, los elementos JSP crean el texto que es insertado en la página resultante.

La siguiente página JSP imprime el día del mes y el año, y despliega un saludo dependiendo de la hora. La página combina elementos de HTML ordinarios con algunos elementos JSP:

- Llamadas a un componente JavaBean.

---

[25] Java Data Base Connectivity.
[26] Java 2 Enterprise Edition. http://java.sun.com/j2ee.

- La inclusión de un archivo externo.
- Expresiones y scriptlets JSP.

```
<HTML>
<%@ page language="java" imports="newstore.util.*" %>

<H1> Bienvenido </H1>

<P> Hoy es </P>
<jsp:useBean id="clock"  class="calendar.jspCalendar" />
<UL>
<LI>Day:  <%= clock.getDayOfMonth() %>
<LI>Year:  <%=clock.getYear() %>
</UL>

<% if (Calendar.getInstance().get(Calendar.AM_PM) == Calendar.AM)
{ %>
Buenos días
<% } else { %>
Buenas tardes
<% } %>
<%@ include file="encabezado.html" %>

</HTML>
```

La página incluye los siguientes componentes:

- ***Directiva JSP.*** Se encarga de pasar información al procesador JSP. En este caso, la primer línea indica la localización de algunas clases o extensiones al lenguaje de programación Java para que sean accesibles desde la página. Las directivas se encuentran entre los delimitadores `<%@` y `%>`.
- ***Elementos HTML o XML.*** Cualquier etiqueta que el procesador JSP no reconoce pasa directamente a la página resultante. Esto incluye etiquetas HTML o XML como las etiquetas `<UL>` y `<H1>` en el ejemplo anterior.
- ***Etiquetas JSP.*** Las etiquetas JSP son típicamente implementadas como etiquetas estándar o personalizadas, y tienen una sintaxis similar a las etiquetas XML. En el ejemplo, la etiqueta `jsp:useBean` crea una instancia del JavaBean `Clock` en el servidor.
- ***Expresiones.*** El procesador JSP evalúa cualquier texto que se encuentre entre los delimitadores `<%=` y `%>`. En los elementos de la lista del ejemplo anterior, los valores de los atributos `Day` y `Year` del componente `Clock` fueron obtenidos e insertados como texto en el archivo JSP; el día como primer elemento de la lista y el año como el segundo elemento.

- ***Scriptlets.*** Los scriptlets son pequeños fragmentos de código que desempeñan funciones no soportadas por etiquetas. El lenguaje de script nativo para JSP 1.0 está basado en el lenguaje de programación Java. El scriptlet en el ejemplo anterior determina la hora del día (AM o PM) y despliega el mensaje adecuado al usuario.

5.2.1.3 Modelos de Aplicación para Páginas JSP

Las páginas JSP son ejecutadas por un procesador JSP, el cual está instalado en el servidor Web o en algún servidor de aplicación que soporte páginas JSP. El procesador JSP recibe solicitudes desde un cliente, y genera la respuesta que es enviada al mismo.

Las páginas JSP son compiladas en Java Servlets, que son una extensión estándar de Java[27]. El desarrollador de las páginas tiene acceso al entorno de aplicación de Java, con las características de facilidad de extensión y facilidad de transportación de esta tecnología.

Cuando una página JSP es llamada por primera vez, es compilada en una clase de Java Servlet y es almacenada en la memoria del servidor. Esto hace posible respuestas más rápidas para las siguientes llamadas a la misma página, evitando así el problema de la interfaz CGI-BIN de generación de un nuevo proceso por cada solicitud HTTP, o la interpretación en tiempo de ejecución de otras soluciones.

Las páginas JSP pueden incluirse en diversos modelos de aplicación y pueden utilizarse en combinación con diferentes protocolos, componentes y formatos. A continuación se describen algunas posibilidades.

En una implementación simple, el navegador invoca directamente una página JSP, la cual genera el contenido solicitado. La página JSP puede llamar a componentes JDBC para generar los resultados, y crear HTML estándar que envía como resultado al navegador.

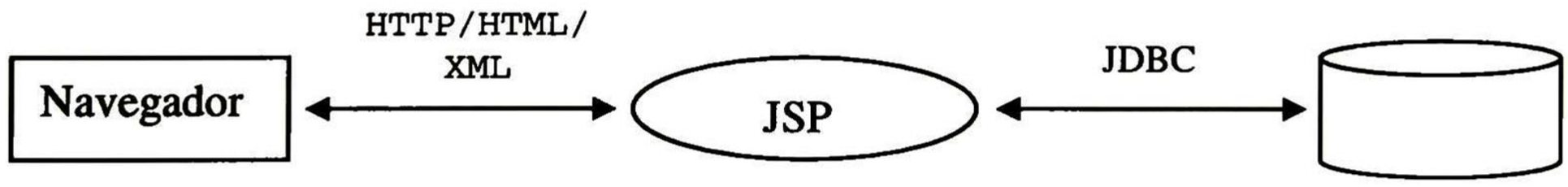


Figura 5.1 Modelo de aplicación simple.

[27] La tecnología Java Servlet es descrita con más detalle en http://www.java.sun.com.

Este modelo básicamente reemplaza el concepto de CGI-BIN mediante una página JSP (compilada como un Servlet de Java). Tiene la ventaja de ser simple de programar; sin embargo, no permite un número grande de clientes solicitando recursos al servidor, dado que cada uno debe establecer o compartir el recurso en cuestión. Por ejemplo, si la página JSP consulta una base de datos, esta puede generar muchas conexiones que pueden afectar el desempeño de la base de datos.

Otra posible configuración se tiene cuando el cliente hace una solicitud directamente a un servlet, el cual genera el contenido dinámico, encapsula el resultado en un JavaBean e invoca a la página JSP. La página JSP tiene acceso al contenido dinámico del componente y envía los resultados (como HTML) al cliente.

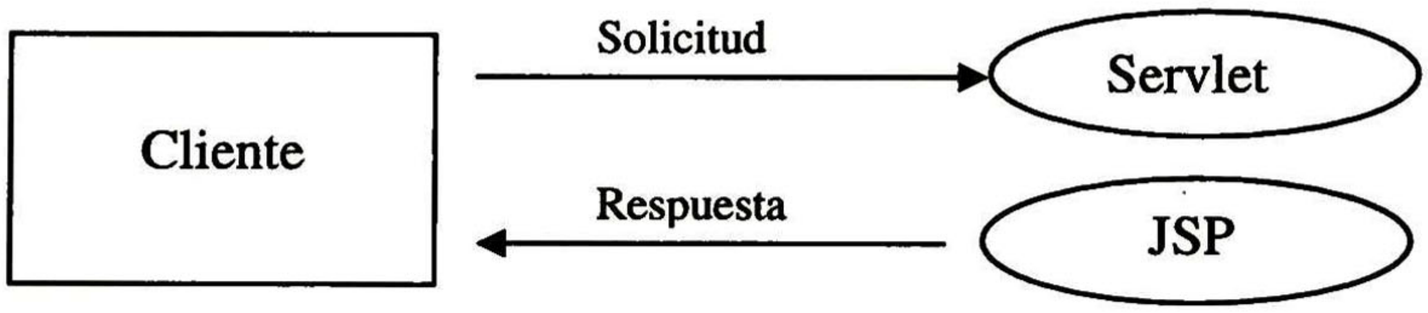


Figura 5.2 Modelo de aplicación con servlets.

Este enfoque crea componentes reutilizables que pueden compartirse entre varias aplicaciones, o como parte de una aplicación más grande. A pesar de esta ventaja, el modelo todavía tiene problemas de consumo de recursos, como puede ser el manejo de conexiones a las bases de datos.

Por último, una página JSP puede actuar como una capa intermedia dentro de un modelo que utiliza Enterprise JavaBeans (EJB). En este caso, la página interactúa con los recursos del servidor vía un componente EJB.

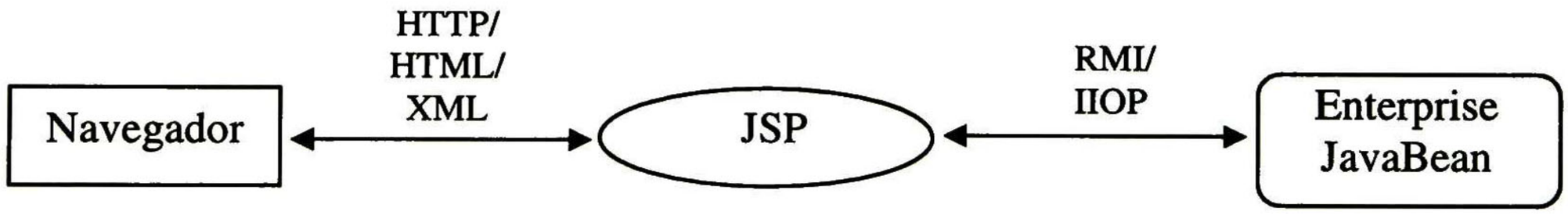


Figura 5.3 Modelo de aplicación con tecnología EJB.

Los componentes EJB administran el acceso a los recursos, lo cual proporciona un desempeño fácil de escalar para un alto número de usuarios concurrentes. Para las

aplicaciones de comercio electrónico, los componentes EJB manejan también las transacciones y la seguridad.

5.3 Consideraciones Sobre el Diseño

El diseño comienza con el modelo de análisis y la arquitectura como sus principales entradas. La principal actividad del diseño es refinar el modelo de análisis de tal manera que este pueda ser implementado con los componentes de la arquitectura. Las actividades de diseño se enfocan en los diagramas de clases e interacción. Las propiedades y operaciones de las clases son definidas con más detalle, de manera que pueda tenerse una relación directa con el código.

Durante el diseño, las interfaces entre los componentes son más específicas y son capaces de manejar la comunicación entre los actores del sistema, así como de soportar el flujo de actividades de los procesos de negocio. Además de la elaboración de clases y sus colaboraciones, las actividades de diseño incluyen la separación y definición de las interfaces de usuario o páginas Web.

Los estilos básicos de arquitecturas para aplicaciones Web involucran el uso de páginas Web, estas actúan como contenedores para la interfaz de usuario de manera general; y sirven para conectar a los navegadores con el resto del sistema. Por lo tanto, es muy importante representar a estos elementos del modelo.

Sin embargo la representación de páginas Web en el modelo tiene algunos problemas. Está claro que las páginas Web son objetos, como los son las interfaces de usuarios en cualquier otro sistema. El problema de modelado se presenta cuando una página Web tiene scripts que deben ser ejecutados en el servidor, este tipo de páginas interactúan con recursos del servidor antes de enviar al cliente la interfaz de usuario completa. La misma página puede contener scripts que se ejecutan en el cliente, de tal manera que cuando esta es procesada por el servidor tiene una funcionalidad distinta de cuando es procesada por el cliente. La siguiente sección muestra una solución para poder representar este tipo de comportamiento específico en el modelo de la aplicación.

5.3.1 Extensión de UML para Aplicaciones Web

Debido a la aceptación de UML[28] como lenguaje de modelado para sistemas de software y el respaldo de la OMG[29], cada vez más sistemas están siendo expresados en esta

---

[28] Unified Modeling Language.
[29] Object Management Group. http://www.omg.org

notación, como es nuestro caso. Sin embargo, para modelar aplicaciones Web, esta notación debe ser extendida, para tal efecto, los creadores de UML han definido un mecanismo de extensión que se basa principalmente en el concepto de *estereotipo,* una extensión al vocabulario del lenguaje que permite asignar un significado específico a un elemento del modelo. Los estereotipos pueden aplicarse a cualquier elemento del modelo y se representan regularmente con el nombre del estereotipo entre los delimitadores << >>, aunque también pueden utilizarse iconos para el elemento. Para una descripción detallada del mecanismo de extensión y de UML en general puede consultarse [Booch, 1999].

En las siguientes secciones se adoptará la extensión de UML para aplicaciones Web propuesta por [Conallen, 1999] y que ha sido aplicada en el diseño de la capa del cliente del sistema para comercio electrónico.

5.3.2 **Diagramas de Secuencia**

La actividad de diseño de páginas Web también incluye la identificación de relaciones con otras páginas y objetos del sistema. Una de las técnicas para identificar las páginas y modelar la interacción entre los actores y el resto del sistema son los diagramas de secuencia. Esto depende de manera significativa del estilo de arquitectura utilizado en la aplicación. El estilo de arquitectura de cliente delgado tiene el mayor número de restricciones sobre el uso de páginas Web admitiendo solamente los elementos especificados en la versión actual de HTML.

Como se ha visto en la sección 5.2 la tecnología seleccionada para la generación de contenido dinámico es la de JavaServer Pages. El entorno JSP proporciona objetos para el manejo de la sesión con el cliente y ofrece al diseñador un mecanismo para mantener el estado del cliente en el servidor. Los actores pueden tener interacción solamente con las páginas del cliente y las páginas del servidor tienen comunicación con los recursos del servidor. Por ejemplo, la figura 5.4 muestra el diagrama de secuencia que describe un escenario en donde el usuario agrega un producto al carrito de compras.

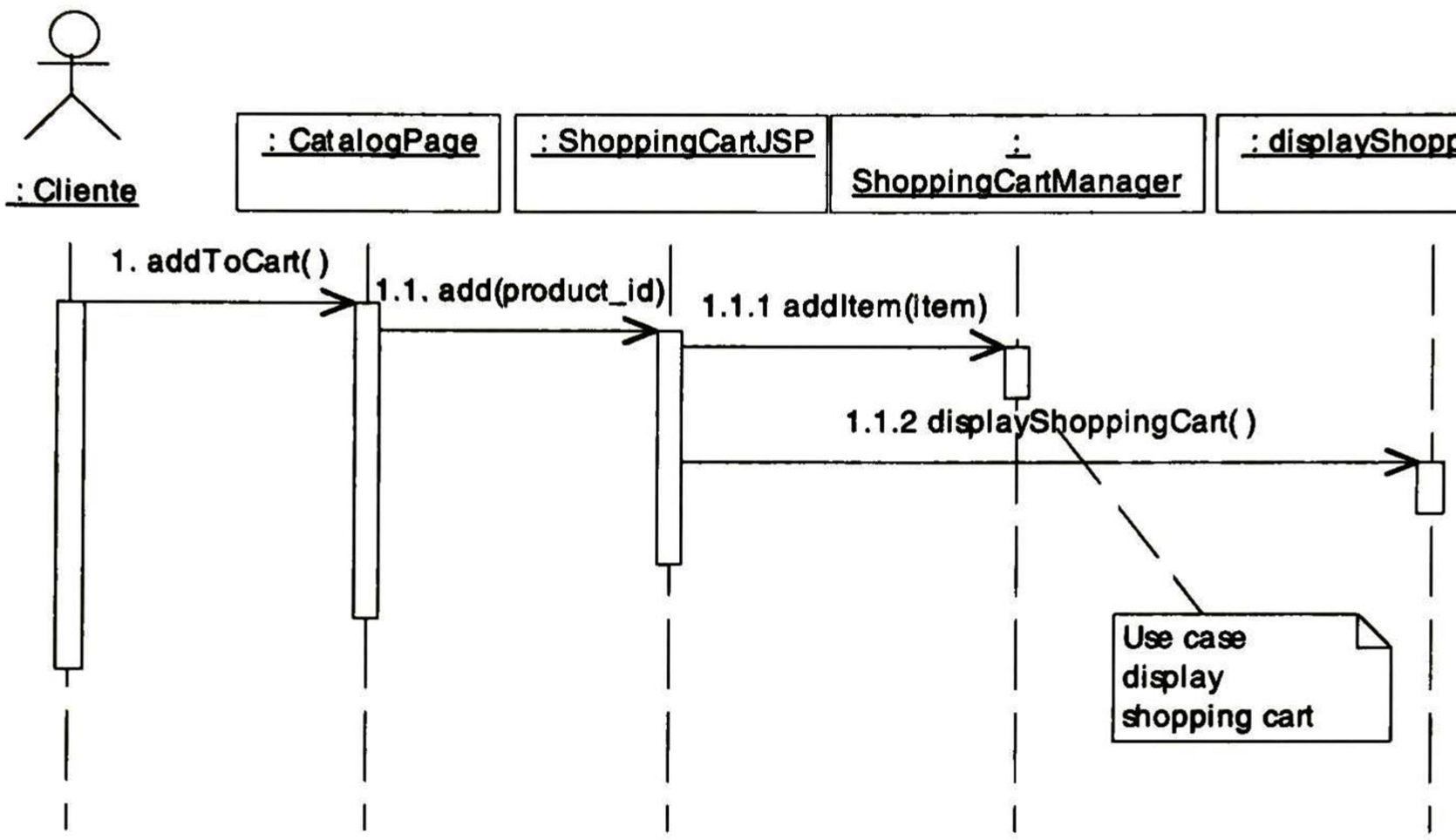


Figura 5.4 Diagrama de secuencia para un escenario del carrito de compras.

El diagrama de secuencia inicia cuando el actor envía un mensaje a la página del catálogo en el cliente, este mensaje es esencialmente una solicitud para la página de servidor (JSP) del carrito de compras y es implementada como una liga desde la página del catálogo. La página de servidor del carrito de compras es cargada por el servidor Web y es ejecutada por el procesador JSP. La lógica en la página envía un mensaje al administrador de carrito de compras para que el producto sea agregado al carrito de compras. El administrador del carrito de compras es un objeto en el servidor que sirve como interfaz entre la página de servidor y los componentes que implementan los objetos de negocio, como es el caso del carrito de compras. Una vez que el producto se ha agregado al carrito de compras, la página del servidor solicita a otra página que despliegue el contenido del carrito mediante una redirección; esta actividad es mostrada con más detalle en otro diagrama de secuencia y es una forma de separar responsabilidades entre las páginas del servidor. Agregar el producto al carrito de compras incluye la creación de objetos intermedios y ejecución de lógica de negocio que es encapsulada en componentes del servidor que pueden ser reutilizados. El diseño de los componentes del servidor es tratado en otra tesis de maestría [Reyes, 2003]; por lo tanto no se muestra aquí.

Una vez que las páginas Web, las principales colaboraciones, y las responsabilidades han sido identificadas, comienza el diseño de las páginas de servidor.

5.3.3 **Diseño de Páginas**

Se pueden modelar los aspectos de una página de servidor con una clase, y los aspectos del cliente con una clase diferente. Para distinguir estos dos tipos de clases utilizamos el mecanismo de extensión de UML para definir estereotipos para cada una de ellas; <<server page>> y <<client page>> respectivamente. Los estereotipos indican que la clase es una abstracción del comportamiento lógico de la clase, ya sea en el cliente o en el servidor. Las dos abstracciones se relacionan una con otra mediante una asociación entre las dos. Esta asociación tiene un estereotipo <<build>>, ya que puede decirse que la página de servidor construye una página de cliente. Cada página generada dinámicamente o cuyo contenido es determinado en tiempo de ejecución, es construida con una página de servidor.

Una relación común entre dos páginas es el hipervínculo. Un hipervínculo en una aplicación Web representa una ruta de navegación a través del sistema. Esta relación es expresada en el modelo mediante una asociación con el estereotipo <<link>>. Esta asociación siempre se origina desde una página de cliente y apunta ya sea a una página de cliente o de servidor. Otro tipo de relación se da cuando una página de servidor redirige una solicitud a otra página para que lleve a cabo procesamiento complementario o como medio para la separación de responsabilidades, como puede ser el procesamiento de errores o para realizar funciones comunes como desplegar el carrito de compras, este tipo de relación se representa mediante una asociación con el estereotipo <<redirect>>.

Utilizando estos estereotipos se facilita el modelado de las páginas y sus relaciones. Las operaciones de una clase con estereotipo <<server page>> son las funciones en los scripts de la página de servidor y sus atributos son las variables globales accesibles por las funciones. De la misma forma, las operaciones y atributos de una clase con estereotipo <<client page>> son las funciones y variables visibles en el cliente. La figura 5.5 muestra un diagrama de clases participantes en el caso de uso *Administrar Carrito de Compras*.

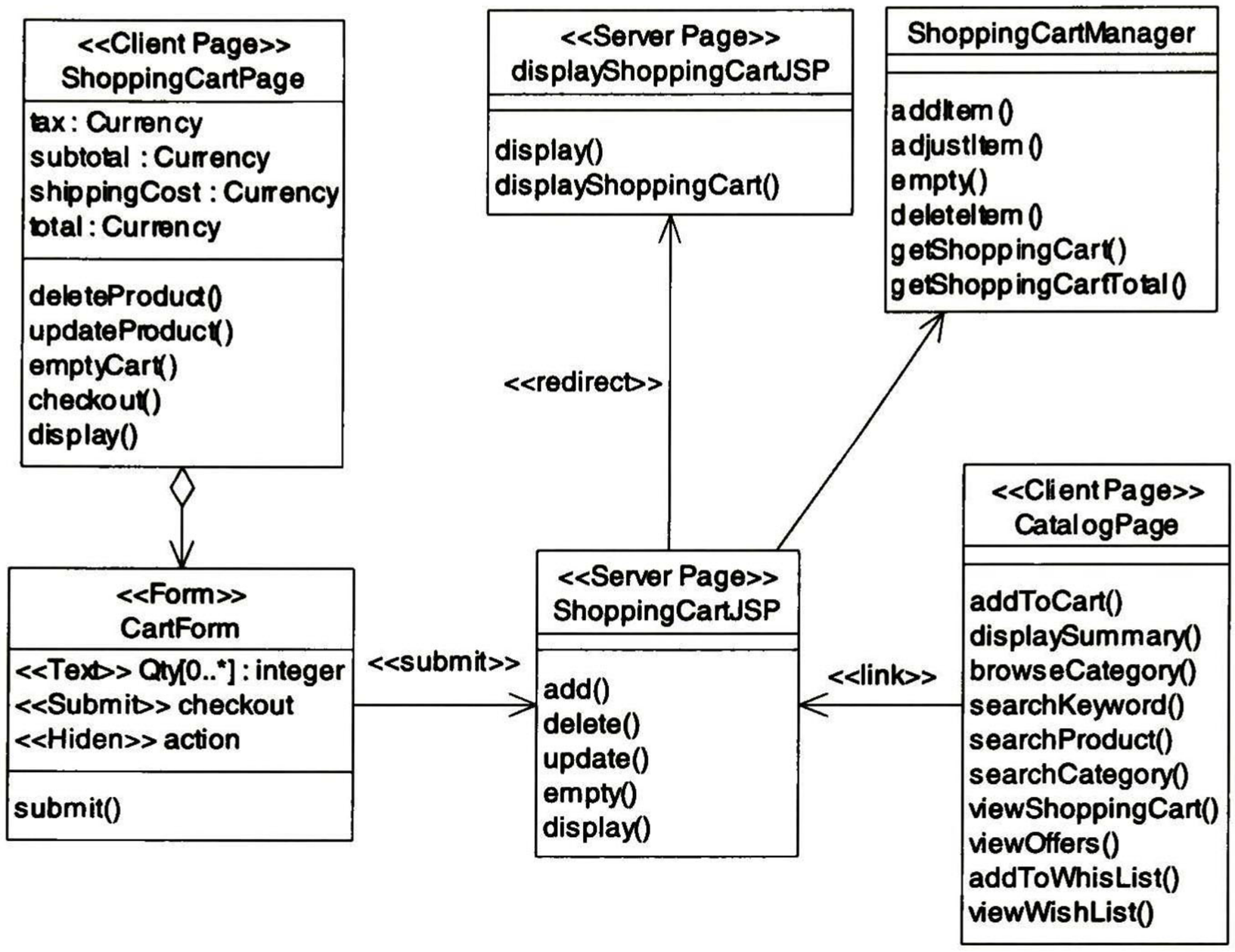


Figura 5.5 Diagrama de clases.

El principal mecanismo de entrada de datos en una página Web es el formulario. Los formularios son definidos en un documento HTML mediante la etiqueta <form>. Cada formulario especifica la página a la que será enviado, este contiene elementos de entrada expresados mediante etiquetas HTML. Los tipos más comunes son <input>, <select> y <textarea>, el tipo <input> a su vez puede ser un campo de edición, un botón, una imagen, un campo oculto, así como otros tipos menos comunes. Para el modelado de formularios se utiliza el estereotipo <<form>> y sus atributos son los elementos de entrada que contiene, el tipo del elemento se especifica con un estereotipo. Cada formulario tiene una relación con una página de servidor, la cual procesa los datos del formulario, esta relación se representa mediante una asociación con el estereotipo <<submit>>. Como los formularios están contenidos en una página de cliente, existe una relación de agregación entre ellos expresada mediante la notación de UML. La figura 5.5 muestra esta asociación entre la página del carrito de compras y el formulario que contiene.

Capítulo 6 **Conclusiones y Trabajo Futuro**

6.1 **Conclusiones**

El principal aporte de esta tesis consiste en la definición de la arquitectura de software para un sistema de comercio electrónico B2C (Business to Consumer) y en el diseño e implementación de los servicios de la capa del cliente descrita en esta arquitectura. Este tipo de sistemas utiliza la tecnología de Internet como infraestructura de comunicación para llegar a un número potencialmente grande de clientes.

Como se vio en el capítulo 2, la arquitectura de software nos da una perspectiva clara del sistema completo y es necesaria para el control del desarrollo de software. Esta perspectiva incluye elementos que nos describen los componentes fundamentales del sistema y la manera en que interactúan. La arquitectura debe contar con atributos de calidad que proporcionen las cualidades que son requeridas por el sistema, dependiendo de estas cualidades pueden utilizarse diferentes patrones o estilos de arquitectura; estos estilos pueden combinarse para obtener los resultados deseados.

El Proceso Unificado para Desarrollo de Software [Jacobson, 1999] fue la metodología que se utilizó para el desarrollo del sistema. Este proceso de desarrollo tiene como una de sus características principales el concepto de arquitectura, además de ser un proceso iterativo e incremental. Los casos de uso son utilizados como base para la definición y análisis de la arquitectura en el proceso de desarrollo. Para la selección de los casos de uso se analizaron varios sitios dedicados al comercio electrónico para encontrar un conjunto común que proporcionara la funcionalidad esperada para una aplicación de este tipo.

Para la definición de esta arquitectura se tomaron en cuenta varias metas o cualidades de diseño complementarias a los casos de uso; principalmente, que la arquitectura proporcionara la flexibilidad y facilidad de transportación necesarias para extender el

sistema, la capacidad de reutilizar el código y la modularidad. Estas cualidades, conllevan al desarrollo de una arquitectura para líneas de productos, donde una configuración de módulos o componentes hace posible la adición de funciones dependiendo de los requerimientos específicos de la aplicación.

Por la naturaleza de la aplicación, es claro que esta pertenece a la familia definida por el estilo general de arquitecturas cliente/servidor. Más específicamente, este tipo de aplicaciones es conocida como aplicación Web, debido a que utiliza como contenedor para la interfaz de usuario un navegador o visualizador Web y utiliza el protocolo HTTP para la comunicación con el servidor. En el capítulo 3, se describieron tres estilos de arquitectura para aplicaciones Web, el estilo de arquitectura de cliente delgado fue utilizado para la implementación del sistema debido a que supone una menor carga de procesamiento en el cliente; dado que no se tiene control sobre la configuración del cliente y se pretende que el mayor número posible de usuarios pueda utilizar el sistema, se consideró al estilo de arquitectura de cliente delgado como el más apropiado.

La arquitectura de la aplicación establece una división en capas, esta división proporciona una mayor facilidad de transportación y facilidad de modificación para el sistema. La capa del cliente se encarga de los servicios de presentación o interfaz de usuario, la capa del servidor se encarga de los servicios de la aplicación o lógica de negocio, mientras que la capa de datos se encarga de la lógica para almacenar, recuperar y modificar los datos, así como de mantener la integridad. El estilo de arquitectura Modelo-Vista-Controlador fue utilizado para separar los servicios mencionados anteriormente, delimitar el alcance de las modificaciones y proporcionar el soporte para el desarrollo de nuevos tipos de clientes.

También se realizó una división modular de la aplicación para obtener un conjunto de módulos o susbsistemas que describen la descomposición funcional del sistema; esta descomposición permite la identificación de componentes reutilizables y relacionarlos de manera que dentro de un módulo se encuentren solamente aquellos componentes que tengan una interacción frecuente debido a las funciones que desempeñan. La comunicación entre los módulos se logra mediante el uso de interfaces bien definidas que proporcionan una manera de limitar el alcance de las modificaciones y establecen los roles que cada subsistema tiene en la aplicación.

La segunda aportación de la tesis consiste en el diseño e implementación de la capa del cliente de la aplicación. En la etapa de diseño se utilizó la extensión de UML para aplicaciones Web propuesta en [Conallen, 1999], con el fin de modelar las páginas Web. Esta extensión propone estereotipos para los elementos del modelo de diseño para una aplicación Web como son las páginas del cliente, páginas del servidor, formularios, hipervínculos y otras asociaciones que se encuentran entre estos elementos. La interacción entre los elementos de diseño se modeló mediante diagramas de secuencia.

Estos diagramas de secuencia muestran la manera en que un usuario interactúa con el sistema en la realización de cada caso de uso.

Se eligió la tecnología JavaServer Pages para la implementación de la capa del cliente del sistema por sus características de facilidad de transportación, desempeño y utilización de recursos del servidor. Esta tecnología nos permite separar la generación de contenido dinámico de la presentación, permite la reutilización de componentes de Java conocidos como JavaBeans y simplifica el desarrollo de páginas Web mediante el uso de etiquetas. El modelo de aplicación seleccionado utiliza JavaBeans en las páginas de servidor para la comunicación con los componentes de negocio implementados con la tecnología Enterprise JavaBeans.

La implementación del sistema para comercio electrónico consiste en un prototipo que incluye la funcionalidad básica descrita en el documento de requerimientos y que sirve como base para la definición de la arquitectura de software. La descripción de los casos de uso fue utilizada para realizar las pruebas del prototipo de la aplicación. La funcionalidad del prototipo implementado puede ser ampliada dadas las cualidades de la arquitectura de software y la tecnología utilizadas.

6.2 Trabajo Futuro

La arquitectura de software aquí presentada puede extenderse para incluir módulos para la administración del catálogo de productos y pedidos, así como módulos para administración de ventas cruzadas, análisis y reportes de ventas. Esto incrementaría la funcionalidad del sistema y sería más adecuado para satisfacer los requerimientos de negocio de diferentes aplicaciones.

Debido al rápido avance tecnológico de Internet, cada vez más dispositivos se utilizan como clientes para aplicaciones de comercio electrónico, como es el caso de teléfonos celulares y agendas electrónicas. Estos dispositivos utilizan protocolos de comunicación inalámbricos como WAP[30] y lenguajes como WML[31] para el diseño de las interfaces de usuario. El diseño e implementación de clientes para los nuevos tipos de dispositivos aumentaría el número de usuarios potenciales de la aplicación adaptándose a los cambios tecnológicos de manera adecuada.

Para el soporte de los servicios de un sistema de comercio electrónico B2B (Business to Business) la comunicación con sistemas externos es esencial, los datos entre sistemas heterogéneos necesitan ser intercambiados y se requiere un formato de datos que pueda ser compartido entre las aplicaciones. Muchas compañías están utilizando XML para

---

[30] Wireless Application Protocol.
[31] Wireless Markup Languague.

integrar aplicaciones existentes e intercambiar datos entre diferentes plataformas y sistemas de bases de datos. Actualmente, una de las mayores áreas de trabajo futuro, es el diseño e implementación de servicios que proporcionen funcionalidad B2B.

Apéndice A Descripción de Casos de Uso

En este apéndice se describen los principales casos de uso que son la base para la definición y análisis de la arquitectura de software del sistema para comercio electrónico. Cada descripción de caso de uso contiene una breve descripción, el flujo de eventos, requerimientos especiales, precondiciones, poscondiciones y extensiones como se describe en [Jacobson, 1999]. El Documento de Especificación de Requerimientos para el Sistema de Comercio Electrónico contiene la especificación completa del sistema.

A.1 Administrar Carrito de Compras

Breve descripción

Este caso de uso permite al cliente utilizar el carrito de compras para agregar o eliminar un producto, modificar la cantidad de los productos agregados del catálogo para su compra posterior o vaciar el carrito de compras.

Flujo de eventos

Flujo básico

Este caso de uso comienza cuando el cliente despliega el carrito de compras:

1. El cliente solicita al sistema una de las siguientes acciones: agregar o eliminar un producto, modificar la cantidad de los productos agregados del catálogo o vaciar el carrito de compras.
2. Dependiendo de lo que el cliente haya solicitado al sistema, se ejecuta uno de los siguientes subflujos:
3. Si el cliente selecciona "agregar producto", el subflujo **Agregar producto** es ejecutado.
4. Si el cliente selecciona "eliminar producto", el subflujo **Eliminar producto** es ejecutado.
5. Si el cliente selecciona "modificar producto", el subflujo **Modificar producto** es ejecutado.
6. Si el cliente selecciona "vaciar carrito de compras", el subflujo **Vaciar carrito de compras** es ejecutado.

Agregar producto

1. El cliente selecciona un producto del catálogo y elige agregarlo al carrito de compras.
2. El producto es agregado al carrito de compras.
3. El sistema ejecuta el caso de uso **A.8 Mostrar Carrito de Compras**.

Eliminar producto

1. El cliente selecciona eliminar un producto del carrito de compras.
2. El sistema elimina el producto del carrito de compras.
3. El sistema ejecuta el caso de uso **A.8 Mostrar Carrito de Compras**.

Modificar producto

1. El cliente modifica la cantidad de un producto del carrito de compras.
2. El cliente selecciona actualizar carrito de compras.
3. El sistema actualiza el carrito de compras.
4. El sistema ejecuta el caso de uso **A.8 Mostrar Carrito de Compras**.

Vaciar carrito de compras

1. El cliente selecciona vaciar carrito de compras.
2. El sistema elimina todos los productos del carrito de compras.
3. El sistema ejecuta el caso de uso **A.8 Mostrar Carrito de Compras**.

Flujo alternativo

Ninguno.

Requerimientos especiales

Ninguno.

Precondiciones

Ninguna.

Poscondiciones

Ninguna.

Extensiones

Ninguna.

A.2 Administrar Perfil de Usuario

Breve descripción

Este caso de uso se presenta cuando un cliente desea modificar la información de su perfil almacenado en el sistema.

Flujo de eventos

Flujo básico

Este caso de uso inicia cuando el cliente desea modificar la información de su perfil.

1. Si el cliente selecciona modificar "nombre de usuario", el subflujo **Modificar nombre de usuario** es ejecutado.
2. Si el cliente selecciona modificar "contraseña", el subflujo **Modificar contraseña** es ejecutado.
3. Si el cliente selecciona modificar "información de facturación y envío", el subflujo **Modificar información de facturación y envío** es ejecutado.

Modificar nombre de usuario

1. El sistema solicita el nuevo nombre de usuario al cliente.
2. El sistema actualiza el perfil del cliente.

Modificar contraseña

1. El sistema solicita la nueva contraseña al cliente.
2. El sistema solicita al cliente que confirme la nueva contraseña.
3. El sistema actualiza el perfil del cliente.

Modificar información de facturación y envío

1. El sistema solicita al cliente la nueva información de facturación.
2. El sistema solicita al cliente la nueva información de envío.
3. El sistema actualiza el perfil del cliente.

Flujo alternativo

Información no válida

Cuando el cliente proporciona alguna información que no sea válida, el sistema le muestra un mensaje indicándole lo anterior, entonces el cliente puede proporcionarla de nuevo.

Requerimientos especiales

Ninguno.

Precondiciones

El usuario debe haber ingresado al sistema mediante el caso de uso **A.7 Ingresar**.

Poscondiciones

Si este caso de uso es exitoso el sistema almacena el nuevo perfil del usuario.

Extensiones

Ninguna.

A.3 Consultar Catálogo de Productos

Breve descripción

Este caso de uso se presenta cuando un cliente desea ver la información de un producto utilizando diferentes criterios de búsqueda.

Flujo de eventos

Flujo básico

Este caso de uso inicia cuando el cliente desea información acerca de un producto y para ello utiliza una búsqueda por categoría, por producto o por palabra clave.

1. El sistema recibe un criterio de búsqueda ya sea identificador de producto, categoría o palabra clave.
2. Si el criterio de búsqueda es un identificador de producto, el subflujo **Consultar productos por identificador** es ejecutado.
3. Si el criterio de búsqueda es un identificador de categoría, el subflujo **Consultar productos por categoría** es ejecutado.

4. Si el criterio de búsqueda es un identificador de producto, el subflujo **Consultar productos por palabra clave** es ejecutado.

Consultar productos por identificador

1. El sistema obtiene los datos del producto que corresponden al identificador de producto especificado.
2. El sistema muestra al usuario los datos del producto.

Consultar productos por categoría

1. El sistema obtiene la lista de productos que corresponden a la categoría de acuerdo al identificador de categoría especificado, o bien la lista de subcategorías si hubiera subcategorías para esa categoría.
2. El sistema muestra al usuario la lista de productos encontrados o en su defecto, la de las subcategorías.

Consultar productos por palabra clave

1. El sistema obtiene la lista de productos que corresponden a la palabra clave introducida por el usuario.
2. El sistema muestra al usuario la lista de productos encontrados.

Flujo alternativo

No hay resultado para el criterio de búsqueda

Si en cualquiera de los subflujos el sistema no obtiene resultados, se mostrará un mensaje informando de lo ocurrido. El cliente puede entonces realizar otra consulta, y comienza de nuevo el caso de uso.

Requerimientos especiales

Ninguno.

Precondiciones

Ninguna.

Poscondiciones

El estado del sistema no será afectado por este caso de uso.

Extensiones

Consultar ofertas

Esta extensión se presenta cuando el cliente se encuentra en el sitio Web y desea consultar las ofertas existentes

1. El cliente selecciona consultar ofertas
2. El sistema realiza una búsqueda de los productos que están en oferta
3. El sistema muestra los productos que se encuentran en oferta

A.4 Consultar Pedido

Breve descripción

Este caso de uso se presenta cuando un cliente desea consultar la lista de sus pedidos o los detalles de algún pedido en especial.

Flujo de eventos

Flujo básico

Este caso de uso inicia cuando el cliente selecciona la opción de consultar pedidos.

1. El cliente selecciona la opción de consultar pedidos
2. El sistema le muestra una lista de los pedidos realizados por el cliente.
3. El cliente selecciona un pedido para consultar los detalles del mismo.
4. El sistema muestra la información del pedido seleccionado.

Flujo alternativo

No existen pedidos realizados

Cuando el cliente no ha realizado ningún pedido, el sistema le muestra un mensaje indicándole que no tiene ningún pedido realizado.

Requerimientos especiales

El cliente solo puede ver la información de sus pedidos.

Precondiciones

Para que este caso se pueda ejecutar, el usuario debe de haber ingresado al sistema, ejecutando el caso de uso **A.7 Ingresar.**

Poscondiciones

El estado del sistema no será afectado por este caso de uso.

Extensiones

Cancelar pedido

Esta extensión se presenta cuando el cliente que se encuentra consultando sus pedidos desea realizar la cancelación de alguno o algunos de ellos que se encuentren en estado pendiente

1. El cliente selecciona el o los pedidos que desea cancelar
2. El cliente selecciona cancelar pedido
3. El sistema solicita la confirmación de la operación
4. El cliente confirma
5. El sistema realiza la cancelación del pedido

A.5 Comprar

Breve descripción

Este caso de uso se presenta cuando un cliente desea comprar los artículos de su carrito de compras.

Flujo de eventos

Flujo básico

Este caso de uso inicia cuando el cliente selecciona comprar los artículos contenidos en el carrito de compras.

1. El cliente realiza la petición para hacer la compra.
2. Si el cliente no ha ingresado, el sistema ejecuta el caso de uso **A.7 Ingresar**.

3. El sistema despliega la dirección de envío y la dirección de facturación correspondiente al perfil del cliente que ha ingresado.
4. El usuario selecciona un método de envío de una lista disponible.
5. El sistema muestra la información del envío y de la facturación, además de la información del pedido.
6. El sistema usa el caso de uso **A.6 Efectuar Pago**.
7. El sistema crea el pedido y le asigna un identificador.
8. El sistema muestra información del pedido y del pago además de un mensaje agradeciendo al cliente su compra.

Flujo alternativo

El pedido no se ha podido crear
En caso de que el pago no se haya podido realizar se le muestra al cliente un mensaje en el cual se le indica que no ha podido crearse el pedido.

Cancelar operación
El cliente puede cancelar la operación en cualquier momento, con lo que termina el caso de uso.

Requerimientos especiales

Ninguno.

Precondiciones

El sistema debe de haber ejecutado el caso de uso **A.8 Mostrar Carrito de Compras.**

Poscondiciones

Si la operación se realiza con éxito, el pedido del usuario es almacenado en el sistema para posteriores referencias.

Extensiones

Ninguna.

A.6 Efectuar Pago

Breve descripción

Cuando el cliente ha confirmado que los productos del carrito de compras son correctos se presenta este caso de uso, en el cual el mismo cliente tendrá que presentar la información correspondiente al pago que desea realizar para que sea procesado su pedido una vez que se haya efectuado el pago.

Flujo de eventos

Flujo básico

1. El sistema requiere que el cliente proporcione la información de la tarjeta de crédito con la cual va a realizar el pago.
2. El cliente proporciona la siguiente información personal para el pago en línea: Nombre y apellidos del cliente, número de tarjeta de crédito, fecha de vencimiento de la misma.
3. Se establece una comunicación con el proveedor de servicios bancarios.
4. El proveedor realiza la validación de la información relativa al pago.
5. El proveedor notifica al sistema que ha efectuado la transacción y le proporciona la información referente a la misma.

Flujo alternativo

Pago rechazado
Este subflujo se presenta cuando el pago en línea es rechazado. Se le da oportunidad al cliente de poder corregir los datos o cancelar la operación, en esta última opción el caso de uso termina.

Cancelar pago
En este caso cuando se está ejecutando el subflujo pago en línea el cliente cancela la operación, terminando el caso de uso.

Requerimientos especiales
Ninguno.

Precondiciones
El cliente debe haber ejecutado el caso de uso **A.5 Comprar**.

Poscondiciones
Si la operación se realiza con éxito, la transacción es realizada y la información referente a ella es almacenada, además el pedido del cliente es creado.

Extensiones
Ninguna.

A.7 Ingresar

Breve descripción
Este caso de uso describe cómo el usuario se identifica ante el sistema para ingresar al mismo.

Flujo de eventos
Flujo básico
Este caso de uso comienza cuando el sistema requiere que el cliente se identifique.

1. El sistema le pide al usuario su nombre y contraseña
2. El usuario ingresa su nombre y contraseña
3. El sistema valida el nombre y contraseña de usuario y establece que el usuario se ha identificado.

Flujo alternativo

Nombre/Contraseña no válido
Si en el flujo básico, el cliente proporciona un nombre o contraseña no válida, el sistema presentará un mensaje indicando lo ocurrido. El usuario puede escoger entre regresar al principio del flujo básico o cancelar el ingreso, con lo cual se termina el caso de uso

Requerimientos especiales

El sistema deberá permitir al usuario registrarse en el sistema mediante el caso de uso **A.9 Registrar Cliente**, en caso de no estar registrado.

Precondiciones

Para que el caso de uso tenga éxito el usuario debe de haber ejecutado el caso de uso **A.9 Registrar Cliente** en la actual o en cualquier sesión anterior.

Poscondiciones

Ninguna.

Extensiones

Ninguna.

A.8 Mostrar Carrito de Compras

Breve descripción

Este caso de uso se presenta cuando se desea consultar los productos del carrito de compras.

Flujo de eventos

Flujo básico

Este caso de uso inicia cuando el sistema va a mostrar la información acerca de los productos contenidos en el carrito de compras.

1. Para cada producto del carrito de compras, el sistema muestra la cantidad, precio unitario, total parcial y total general.

Flujo alternativo

El carrito de compras está vacío

Cuando el cliente no a agregado ningún producto o acaba de vaciar el carrito de compras, se mostrará un mensaje indicándole que el carrito de compras no contiene ningún producto.

Requerimientos especiales

Ninguno.

Precondiciones

Ninguna.

Poscondiciones

Ninguna.

Extensiones

Ninguna.

A.9 Registrar Cliente

Breve descripción

Cuando un cliente desea registrarse en el sistema, se presenta este caso en el cual se guarda la información referente al cliente.

Flujo de eventos

Flujo básico

Este caso de uso empieza cuando un cliente desea registrarse en el sistema o el sistema solicita al cliente registrarse.

1. El sistema solicita al cliente los datos necesarios para crear una cuenta en el sistema.
2. El sistema solicita al cliente los datos necesarios para la facturación y el envío de sus pedidos.
3. El cliente confirma la operación.

Flujo alternativo

Falta algún dato

Si el usuario no proporciona algún dato indispensable para el registro, aparece el mensaje correspondiente para que termine de proporcionar la información que se le solicita, o bien puede abandonar la acción con lo que termina este caso de uso.

Datos no válidos

Si el usuario proporciona algún dato no válido para el registro, aparece el mensaje correspondiente para que vuelva a proporcionar la información que fue no válida, o bien puede abandonar la operación con lo que termina este caso de uso.

Requerimientos especiales

Ninguno.

Precondiciones

Ninguna.

Poscondiciones

Si este caso de uso se realiza con éxito, el cliente es registrado en el sistema.

Extensiones

Ninguna.

A.10 Salir

Breve descripción

Este caso de uso se presenta cuando el cliente desea terminar su sesión actual.

Flujo de eventos

Flujo básico

Este caso de uso inicia cuando el cliente desea terminar la sesión y salir del sistema

1. El cliente elige terminar sesión.
2. El sistema termina la sesión del usuario.

Flujo alternativo

Ninguno.

Requerimientos especiales

Ninguno.

Precondiciones

El cliente debe haber ingresado al sistema.

Poscondiciones

Al ejecutarse con éxito este caso de uso, el sistema desconocerá la identidad del usuario.

Extensiones

Ninguna.

A.11 Ver Carrito de Compras

Breve descripción

Este caso de uso se presenta cuando el cliente desea consultar los productos del carrito de compras.

Flujo de eventos

Flujo básico

Este caso de uso inicia cuando el cliente desea consultar información acerca de los productos contenidos en el carrito

1. El cliente solicita ver el contenido del carrito de compras.
2. El sistema ejecuta el caso de uso **A.8 Mostrar Carrito de Compras**.

Flujo alternativo

Ninguno.

Requerimientos especiales

Ninguno.

Precondiciones

Ninguna.

Poscondiciones

Ninguna.

Extensiones

Ninguna.

Apéndice B Modelo de Análisis

En este apéndice se presentan las realizaciones de algunos de los principales casos de uso del modelo de análisis del Sistema para Comercio Electrónico. En cada caso de uso se incluye un diagrama de clases participantes, el diagrama de colaboración del caso de uso, así como el flujo de eventos del mismo. En estos diagramas se han utilizado los estereotipos para las clases de análisis propuestos en [Jacobson, 1999]. Estos tres estereotipos están estandarizados en UML y son utilizados para ayudar al desarrollador a distinguir la función de los diferentes tipos de clases, como se describe a continuación:

Clases de Interfaz[32]. Una clase de interfaz es utilizada para modelar la interacción entre el sistema y sus actores (por ejemplo, los usuarios y sistemas externos). Esta interacción frecuentemente involucra la recepción y presentación de información, así como las solicitudes de los usuarios y sistemas externos.

Clases de Entidad. Una clase de entidad es utilizada para modelar la información persistente en el sistema. Las clases de entidad modelan la información y el comportamiento asociado de un fenómeno o concepto, un objeto real o un evento.

Clases de Control. Las clases de control representan la coordinación, secuencia, transacciones y el control de otros objetos y son utilizadas principalmente para encapsular el control relacionado a un caso de uso específico. Las clases de control, también se utilizan para representar cálculos y reglas de negocio.

Cada una de las diferentes clases de análisis se representa con un icono particular como se muestra en la siguiente figura.

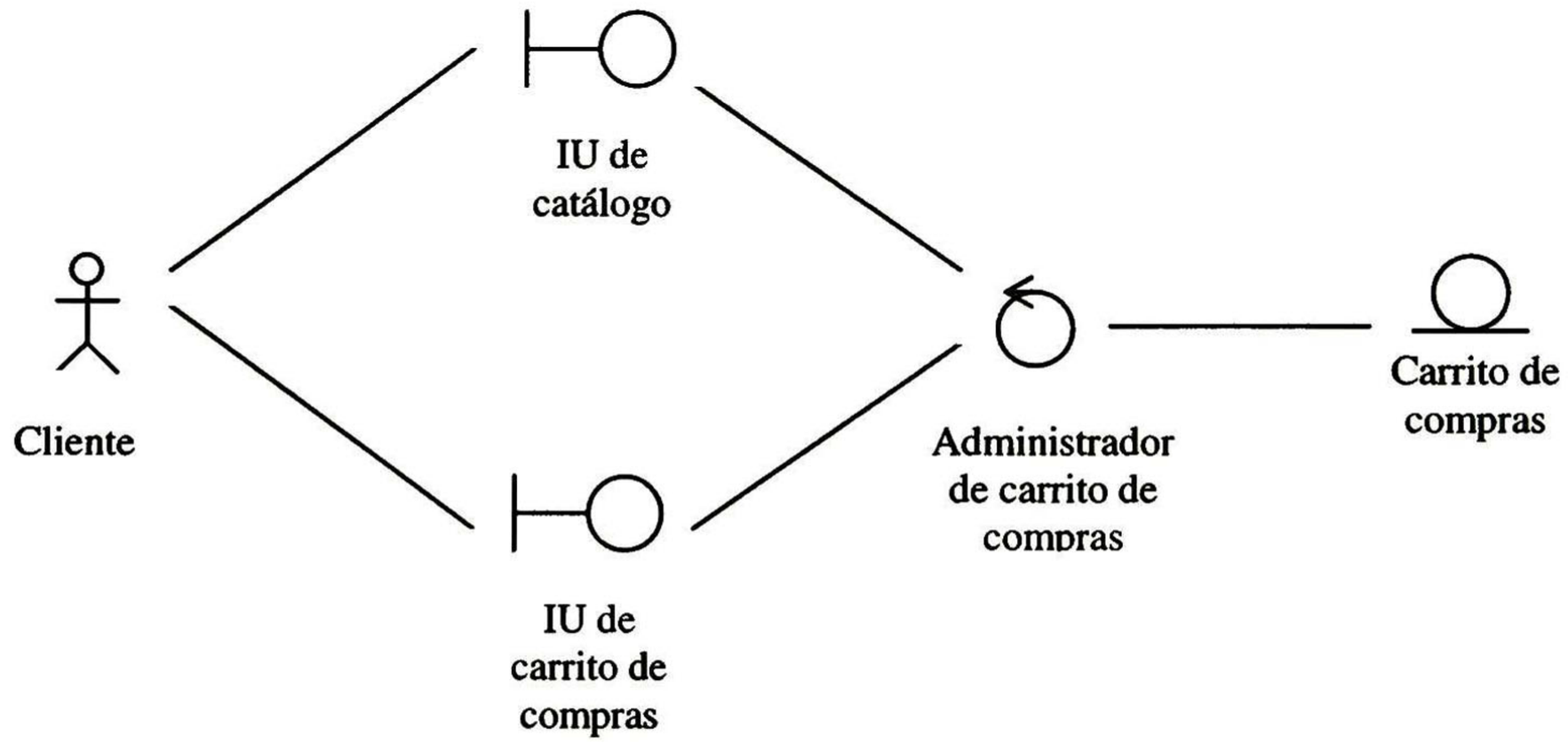
Clase de Interfaz	Clase de Control	Clase de Entidad

[32] Boundary Classes.

El modelo de análisis completo puede consultarse en el Documento de Análisis para el Sistema de Comercio Electrónico.

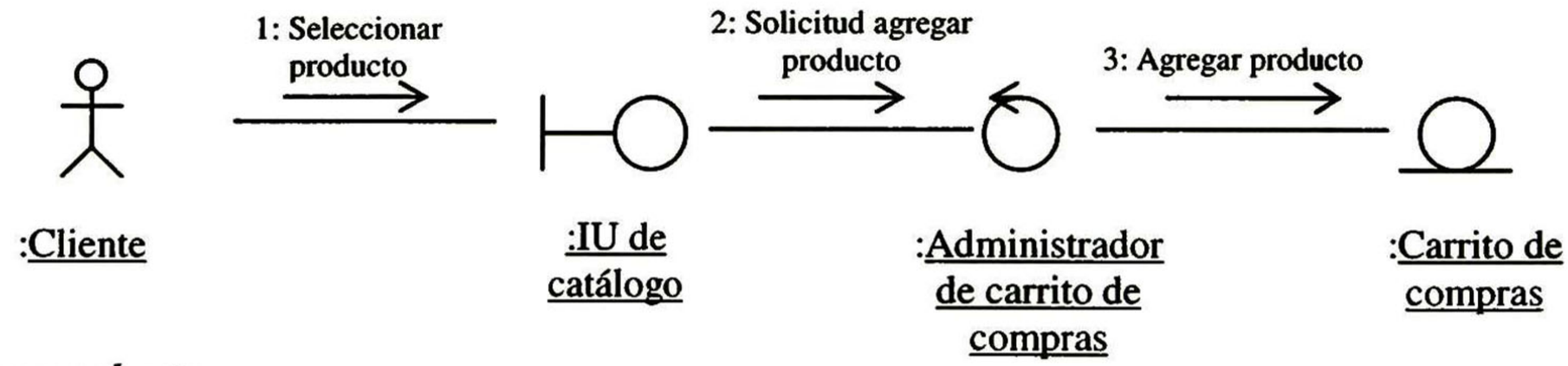
B.1 Administrar Carrito de Compras

Diagrama de clases

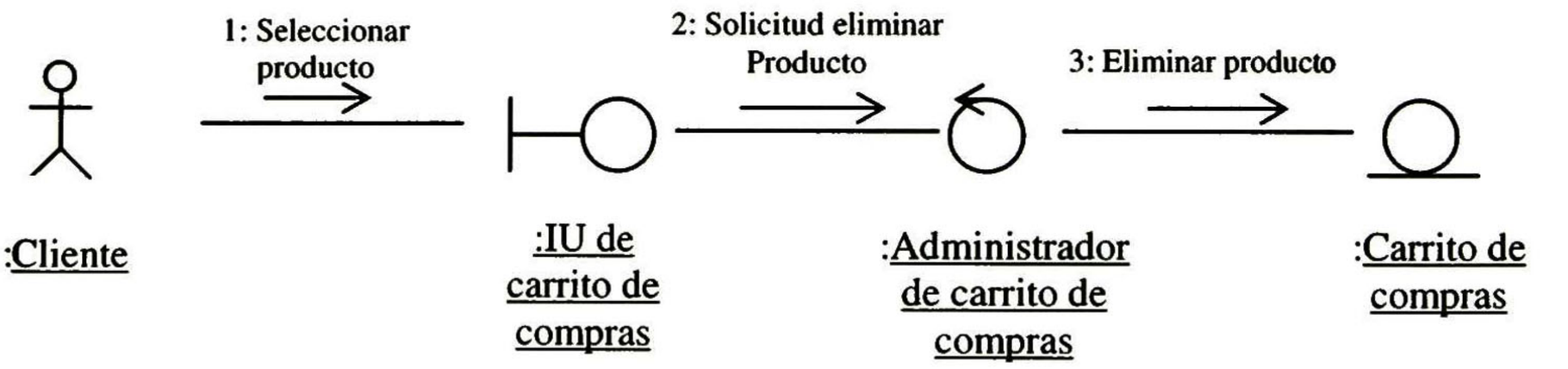


Diagramas de colaboración

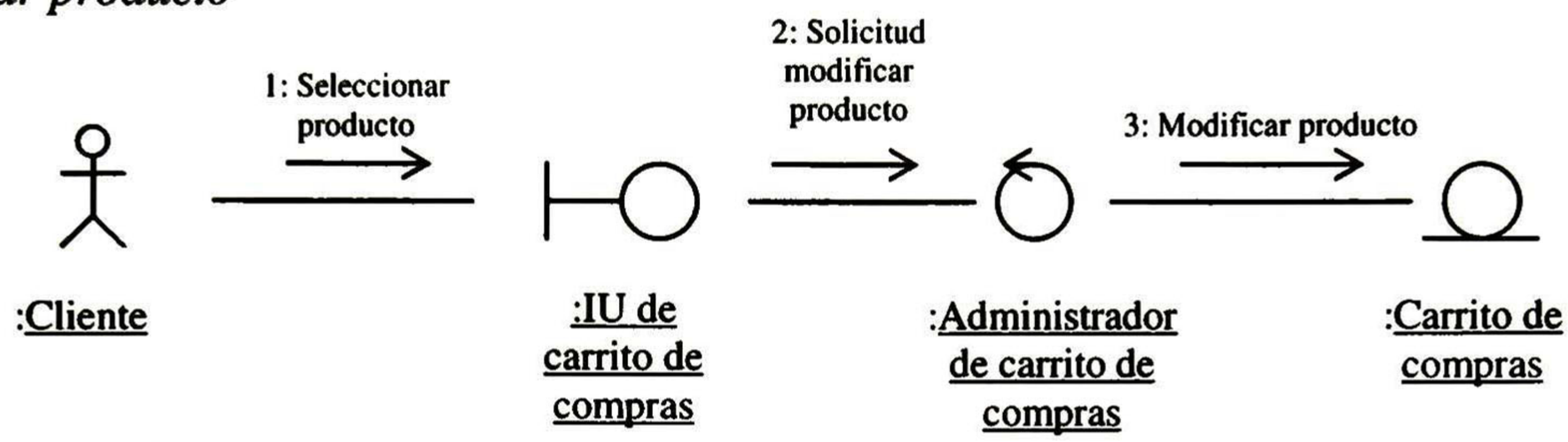
Agregar producto



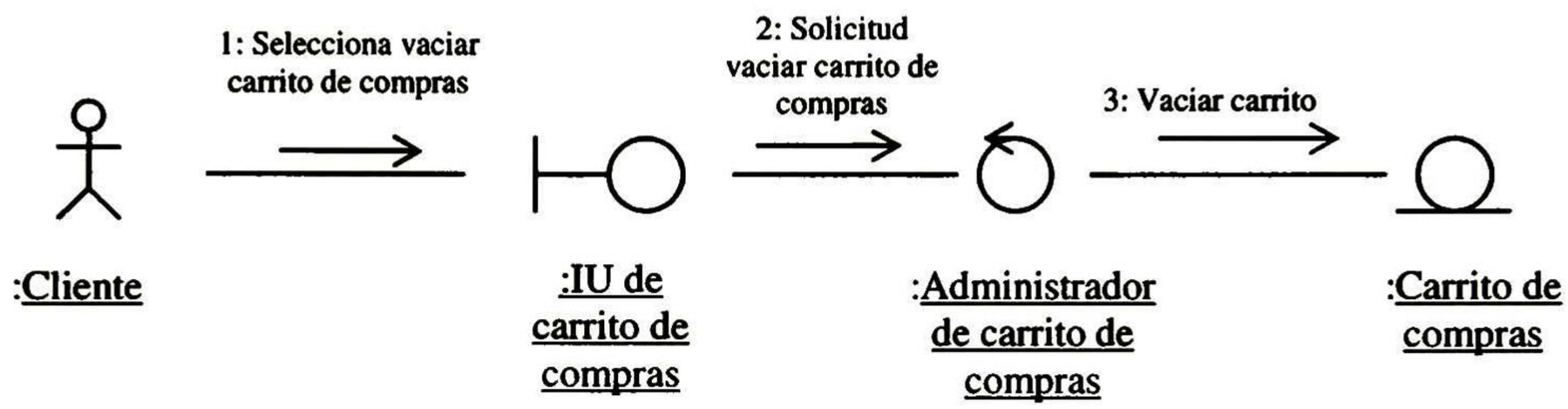
Eliminar producto



Modificar producto



Vaciar carrito de compras



Flujo de eventos

Agregar producto

El *cliente* selecciona un producto para agregar al carrito de compras a través de la *interfaz de usuario de catálogo* de productos (1). La *interfaz de catálogo* utiliza al *administrador de carrito de compras* para agregar el producto al *carrito de compras* (2). El producto es añadido al *carrito de compras* (3). El caso de uso **Mostrar carrito de compras** es ejecutado.

Eliminar producto

El *cliente* selecciona un producto para eliminar del carrito de compras a través de la *interfaz de usuario de carrito de compras* (1). La *interfaz de usuario de carrito de compras* utiliza al *administrador de carrito de compras* para eliminar el producto del *carrito de compras* (2). El producto es eliminado del *carrito de compras* (3). El caso de uso **Mostrar carrito de compras** es ejecutado.

Modificar producto

El *cliente* selecciona un producto del *carrito de compras* para modificar la cantidad a través de la *interfaz de usuario de carrito de compras* (1). La *interfaz de usuario de carrito de compras* utiliza al *administrador de carrito de compras* para modificar la cantidad del producto en el *carrito de compras* (2). El *carrito de compras* es actualizado (3). El caso de uso **Mostrar carrito de compras** es ejecutado.

Vaciar carrito de compras

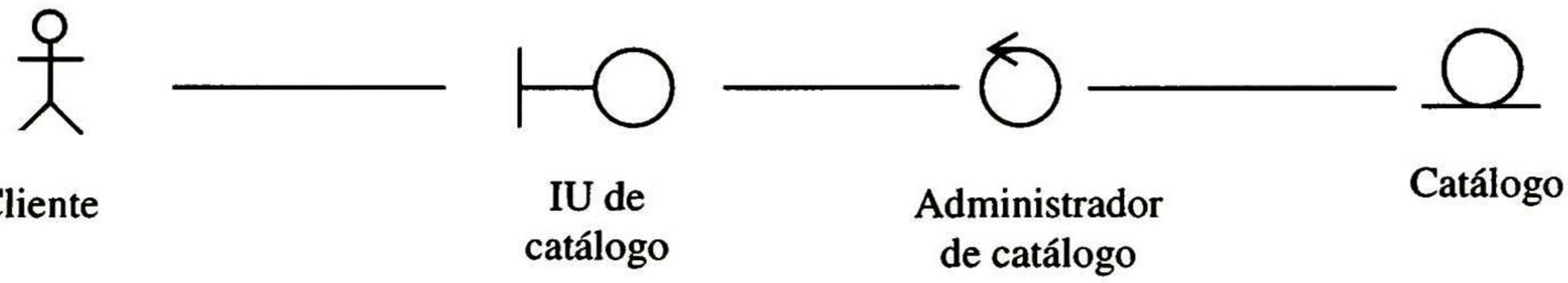
El *cliente* selecciona vaciar el carrito de compras a través de la *interfaz de usuario de carrito de compras* (1). La *interfaz de usuario de carrito de compras* utiliza al *administrador de carrito de compras* para eliminar todos los productos del *carrito de compras* (2). El *carrito de compras* es actualizado (3). El caso de uso **Mostrar carrito de compras** es ejecutado.

Requerimientos especiales

Ninguno.

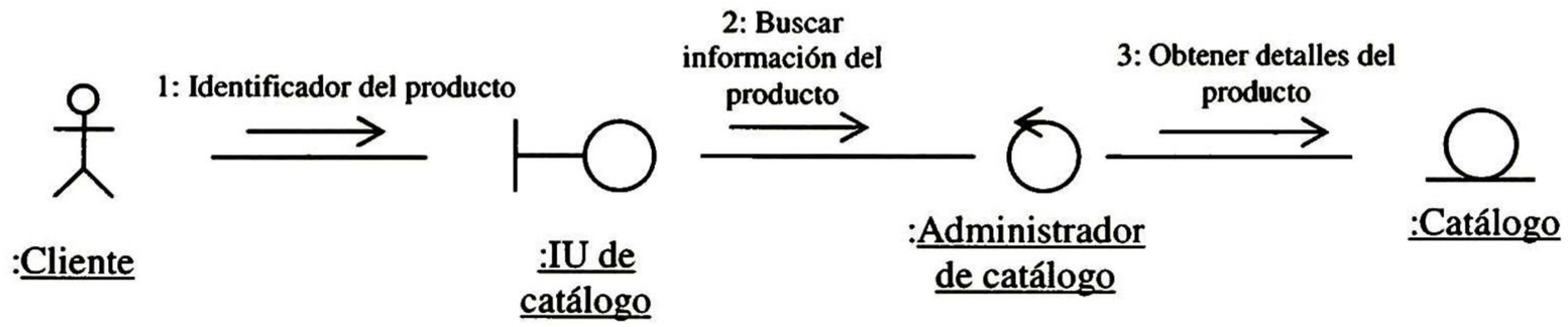
B.2 Consultar Catálogo de Productos

Diagrama de clases

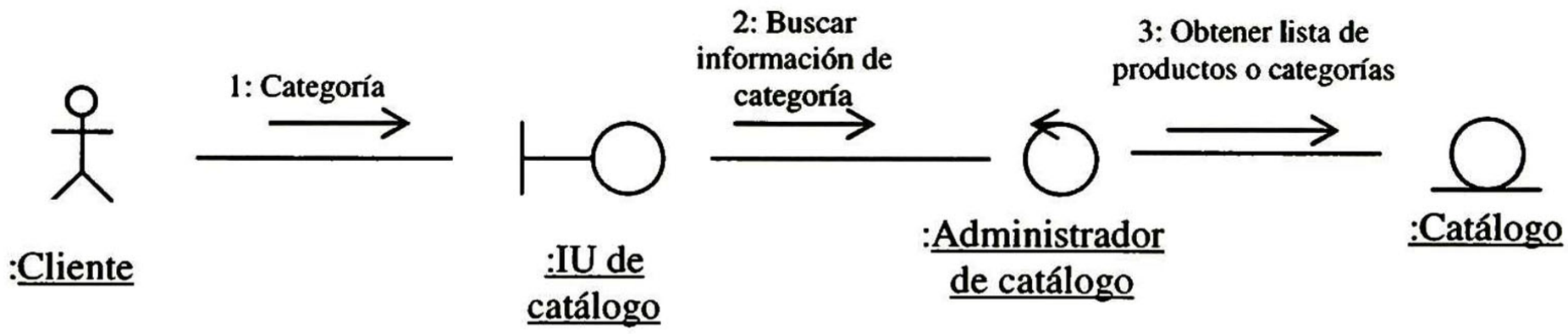


Diagramas de colaboración

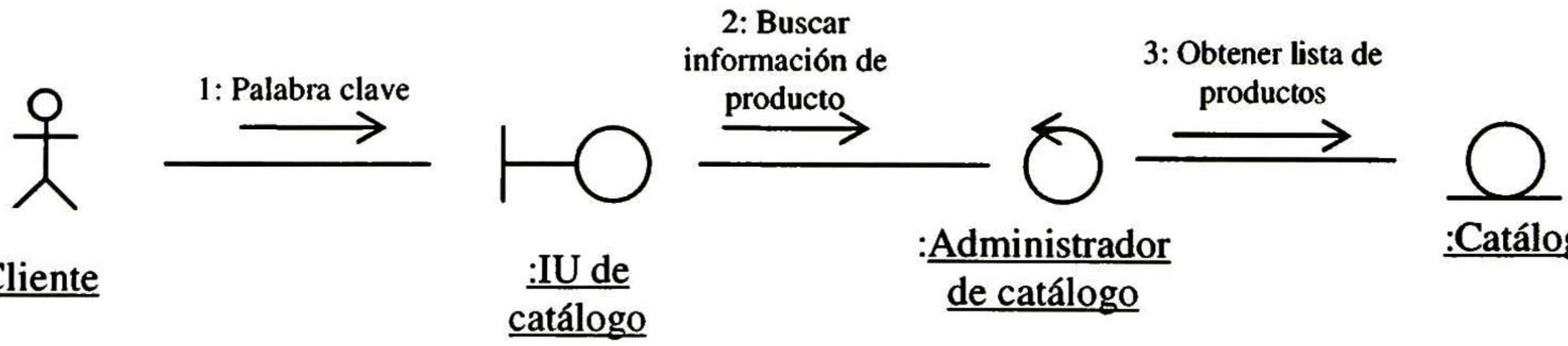
Consultar productos por identificador



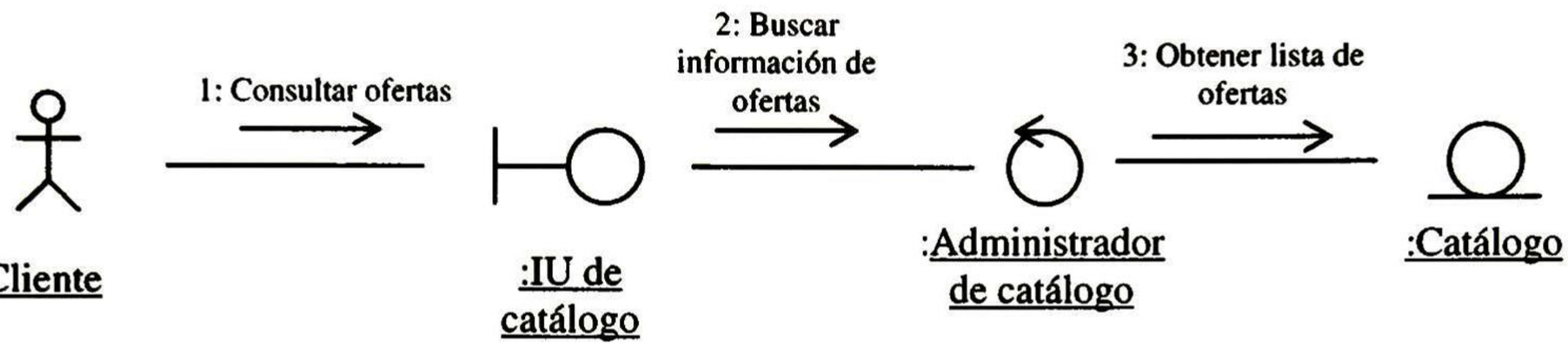
Consultar productos por categoría



Consultar productos por palabra clave



Consultar ofertas



Flujo de eventos

Consultar productos por identificador

El *cliente* solicita consultar algún producto introduciendo el identificador del producto a través de la *interfaz de usuario de catálogo* (1). La *interfaz de usuario de catálogo* utiliza al *administrador de catálogo* para realizar la búsqueda (2). El *administrador de catálogo* obtiene los detalles del producto especificado (3). La *interfaz de usuario de catálogo* muestra la información del producto al *cliente*.

Consultar productos por categoría

El *cliente* solicita consultar algún producto introduciendo la categoría del producto a través de la *interfaz de usuario de catálogo* (1). La *interfaz de usuario de catálogo* utiliza el *administrador de catálogo* para realizar la búsqueda (2). El *administrador de catálogo* obtiene la lista de productos o subcategorías si las hubiera (3). La *interfaz de usuario de catálogo* muestra la información obtenida al *cliente*.

Consultar productos por palabra clave

El *cliente* solicita consultar algún producto introduciendo una palabra clave a través de la *interfaz de usuario de catálogo* (1). La *interfaz de usuario de catálogo* utiliza al *administrador de catálogo* para realizar la búsqueda (2). El *administrador de catálogo* obtiene una lista de productos que contienen la palabra clave (3). La *interfaz de usuario de catálogo* muestra la lista de productos al *cliente*.

Consultar ofertas

El *cliente* solicita consultar las ofertas existentes a través de la *interfaz de usuario de catálogo* (1). La *interfaz de usuario de catálogo* utiliza al *administrador de catálogo* para realizar la búsqueda de información de ofertas (2). El *administrador de catálogo* obtiene una lista de ofertas (3). La *interfaz de usuario de catálogo* muestra la lista de ofertas al *cliente*.

Requerimientos especiales

Ninguno.

B.3 Comprar

Diagrama de clases

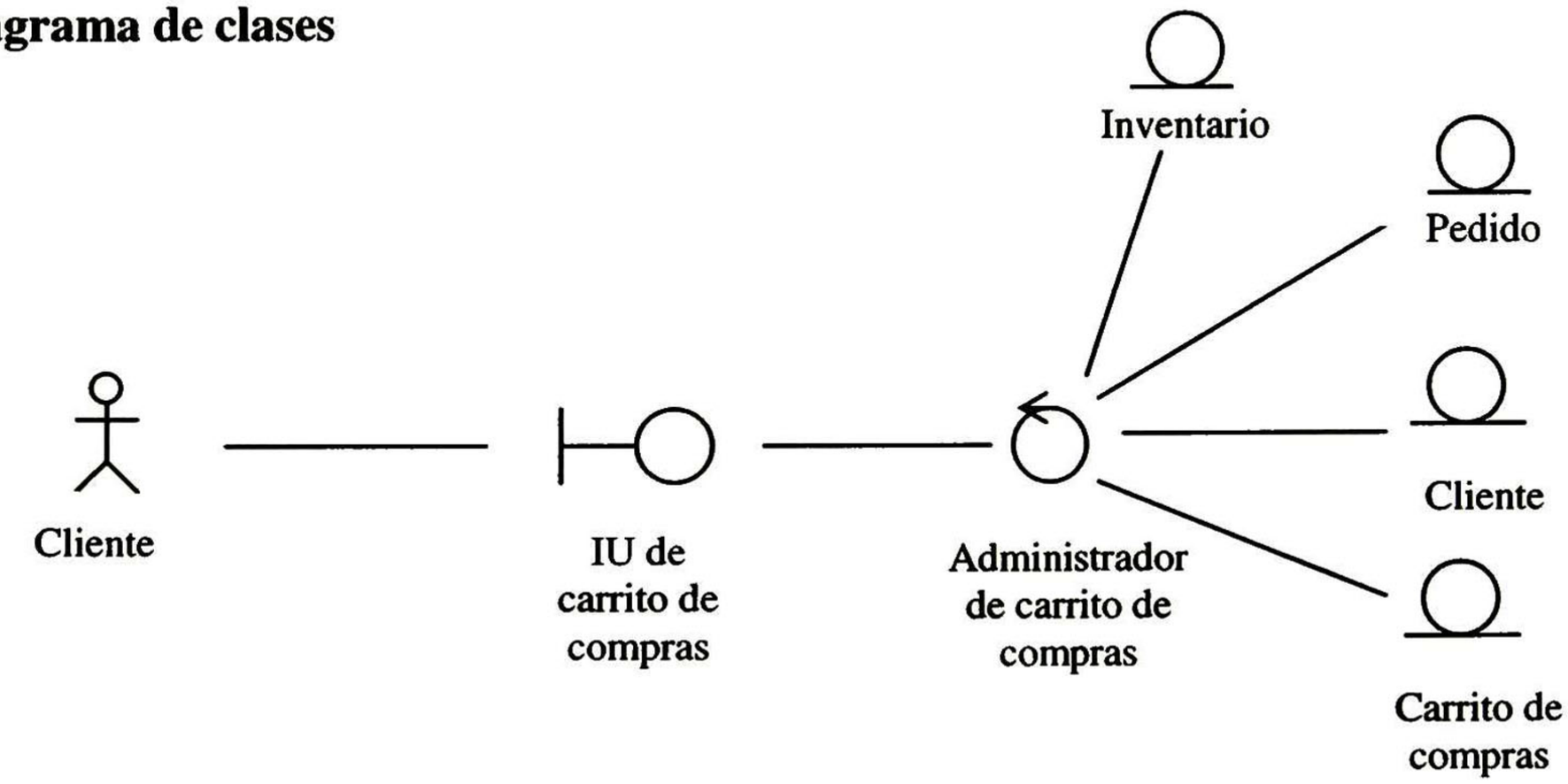
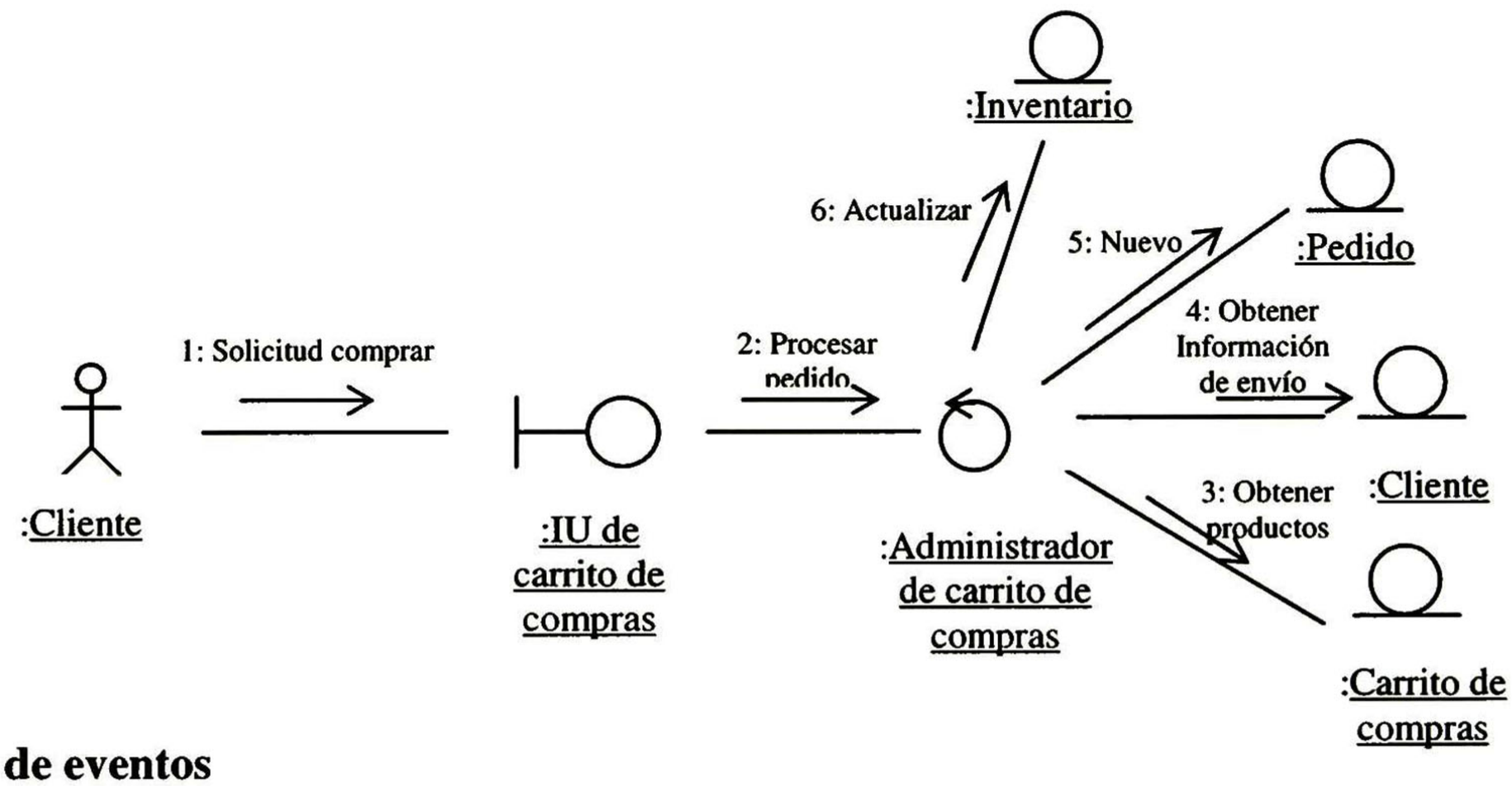


Diagrama de colaboración



Flujo de eventos

El *cliente* solicita realizar la compra y confirma que la información mostrada a través de la *interfaz de usuario de carrito de compras* es correcta (1). El *administrador de carrito de compras* recibe un mensaje para crear el *pedido* (2). El *administrador de carrito de compras* obtiene la información de los productos en el carrito de compras (3). Después el *administrador de carrito de compras* obtiene la información de facturación y envío del *cliente* (4). Para obtener la información de pago se utiliza el caso de uso **Efectuar pago**. El *administrador de carrito de compras* crea el *pedido* con el estado "Pendiente" (5). Al crearse el pedido se actualiza el *inventario* (6).

Requerimientos especiales

El cliente debe haber ingresado al sistema mediante el caso de uso **Ingresar** antes de utilizar este caso de uso.

B.4 Efectuar Pago

Diagrama de clases

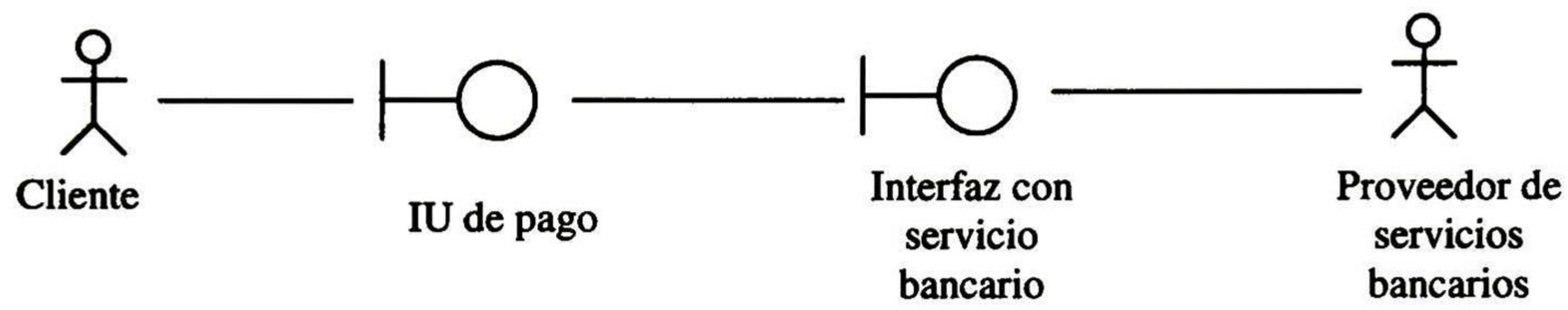
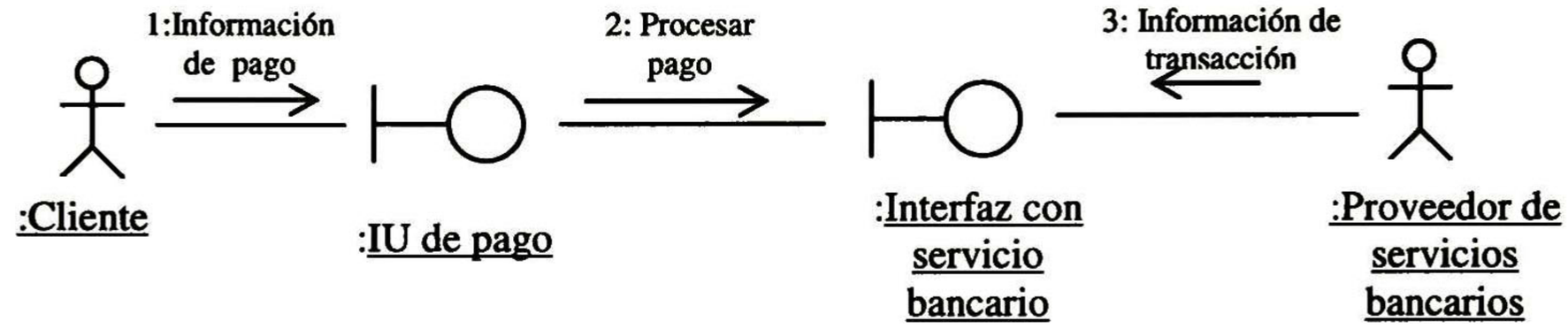


Diagrama de colaboración



Flujo de eventos

El *cliente* solicita realizar el pago en línea a través de la *interfaz de usuario de pago* ingresando los datos de su tarjeta de crédito (1). La *interfaz de usuario de pago* envía la información de pago para su procesamiento a través de la *interfaz con el servicio bancario* (2). El proveedor de servicios bancarios realiza la validación de la información y efectúa la transacción (3).

Requerimientos especiales

El cliente debe haber ejecutado el caso de uso **Comprar**.

Apéndice C Descripción de la Arquitectura de Software

Este apéndice presenta una descripción de la arquitectura de software para el Sistema de Comercio Electrónico. La descripción de la arquitectura incluye algunos extractos de los modelos de casos de uso, análisis y diseño como se describe en [Jacobson, 1999]. Este apéndice está basado en el Documento de Arquitectura de Software para el Sistema de Comercio Electrónico donde puede encontrarse la descripción completa.

C.1 Breve Descripción

El documento de descripción de la arquitectura se deriva de versiones de los diferentes modelos de la fase de elaboración del sistema. La descripción de la arquitectura es un extracto, un conjunto de perspectivas de los modelos de la línea de base de la arquitectura. Las perspectivas incluyen los elementos significativos para la arquitectura.

El propósito de este documento es presentar un mapa general del sistema, no una especificación detallada del sistema. Además de contener la información necesaria para que los desarrolladores desempeñen su trabajo.

C.2 Referencias

Este documento contiene extractos de los modelos de requerimientos, análisis y diseño del sistema para comercio electrónico. Para más detalles, referirse a estos documentos.

C.3 Representación de la Arquitectura

Para la representación de la arquitectura de este documento se utiliza el Lenguaje Unificado de Modelado (UML) y el Proceso Unificado para desarrollo de software. UML tiene elementos apropiados para representar una arquitectura y el Proceso Unificado nos proporciona guías para lo que constituye una buena arquitectura y los productos del trabajo o entregables necesarios.

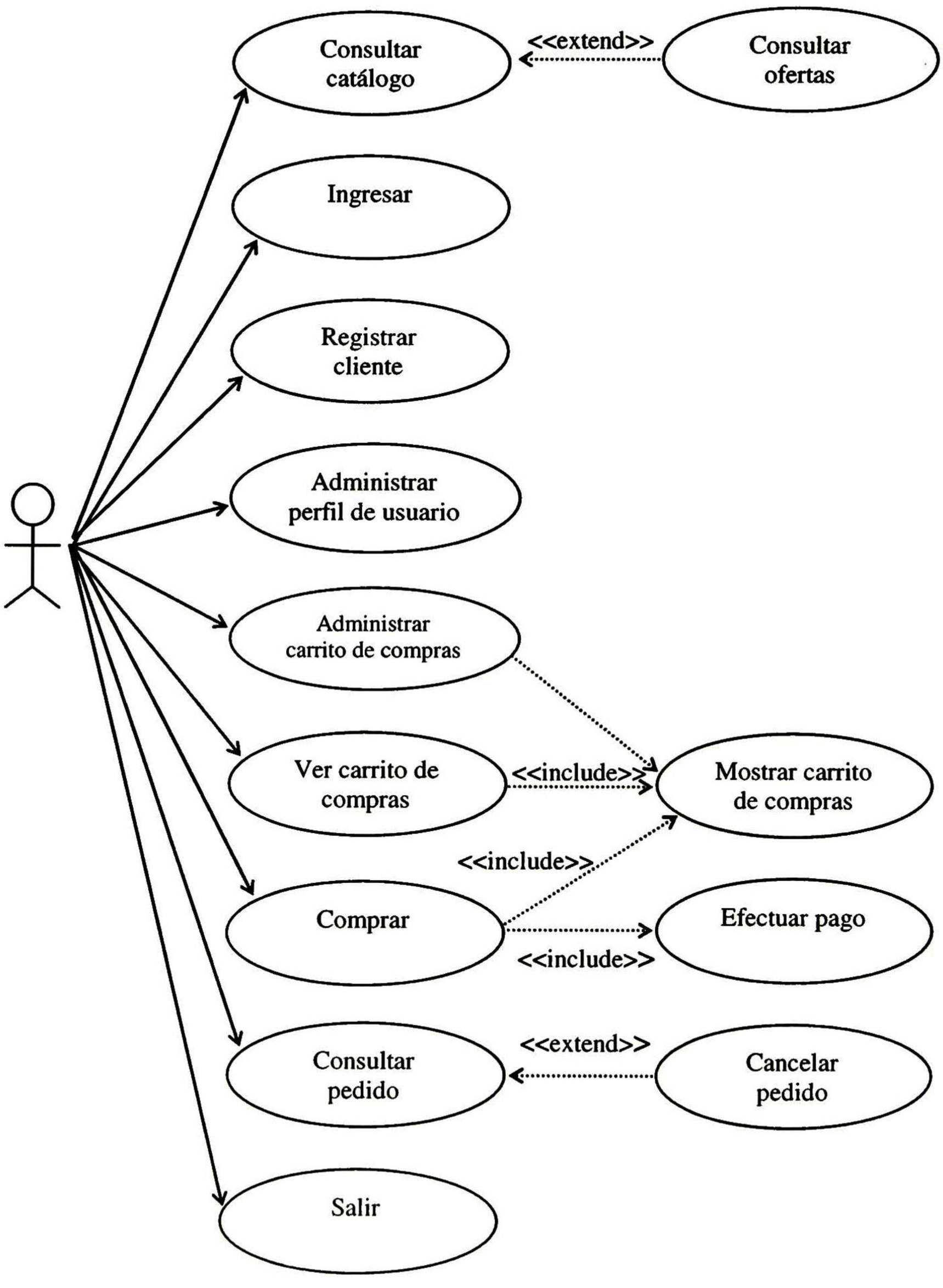
C.4 Metas y Restricciones de la Arquitectura

La arquitectura que se describe en este documento debe permitir el desarrollo rápido de aplicaciones para Web, en particular, de comercio electrónico B2C (Business to Consumer). Estos sistemas deben

ser independientes de la plataforma de hardware, flexibles y fáciles de mantener. Se utiliza un estilo de arquitectura cliente / servidor y la tecnología de Internet para la comunicación.

C.5 Perspectiva de Casos de Uso

La perspectiva de Casos de Uso es una entrada importante para la selección del conjunto de escenarios y casos de uso que serán implementados en una iteración. Describe el conjunto de escenarios y casos de uso que representan funcionalidad importante del sistema. También describe el conjunto de escenarios y casos de uso que tienen una cobertura importante de la arquitectura o que ilustran puntos delicados de la arquitectura. La siguiente figura muestra los casos de uso significativos para la arquitectura de software del sistema.

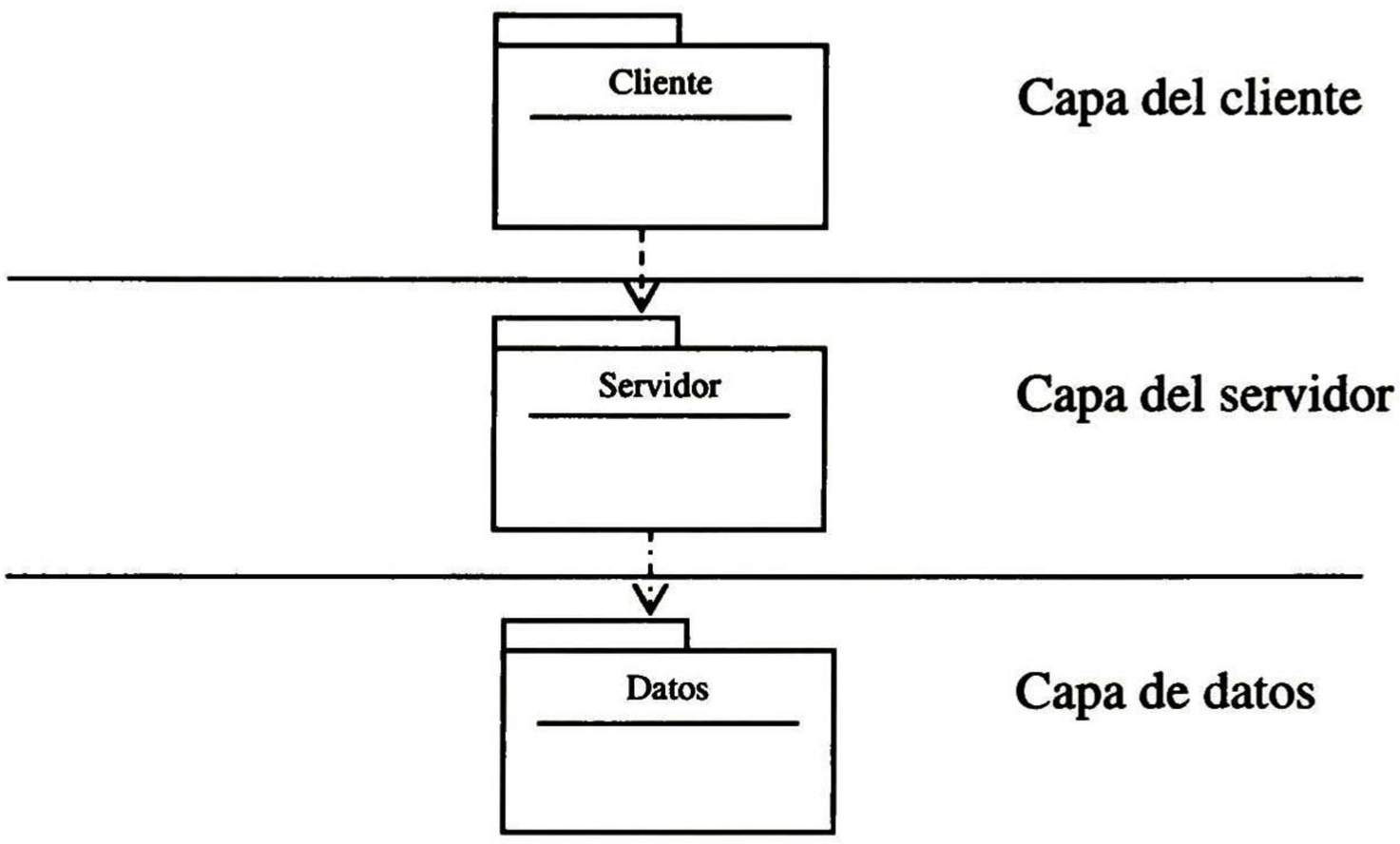


La descripción detallada de los casos de uso puede consultarse en el Documento de Especificación de Requerimientos del Sistema para Comercio Electrónico.

C.6 Perspectiva Lógica

En las siguientes secciones se examina la arquitectura de la aplicación desde una perspectiva de alto nivel. La división de la aplicación se realiza de dos formas. La primera partición divide la aplicación en múltiples capas, separando los clientes de la aplicación, la funcionalidad del servidor de la aplicación y la funcionalidad de la persistencia de datos y sistemas externos. La segunda forma de partición es la tradicional descomposición modular de las funciones de la aplicación en susbsistemas.

C.6.1 Descripción de las Capas del Sistema



Capa del Cliente

La capa del Cliente es responsable de presentar los datos al usuario, tener interacción con el usuario y comunicarse con otras capas de la aplicación. Frecuentemente la capa del Cliente es la única que el usuario de la aplicación puede ver. La capa del Cliente de la aplicación consiste principalmente en un navegador desplegando páginas Web generadas por Java Server Pages en el servidor. Esta capa se comunica con otras capas a través de interfaces bien definidas y proporciona las cualidades de flexibilidad y extensibilidad. Debido a que la capa del Cliente principalmente ofrece funciones de presentación, se utilizó un estilo de arquitectura de cliente delgado.

Capa del Servidor

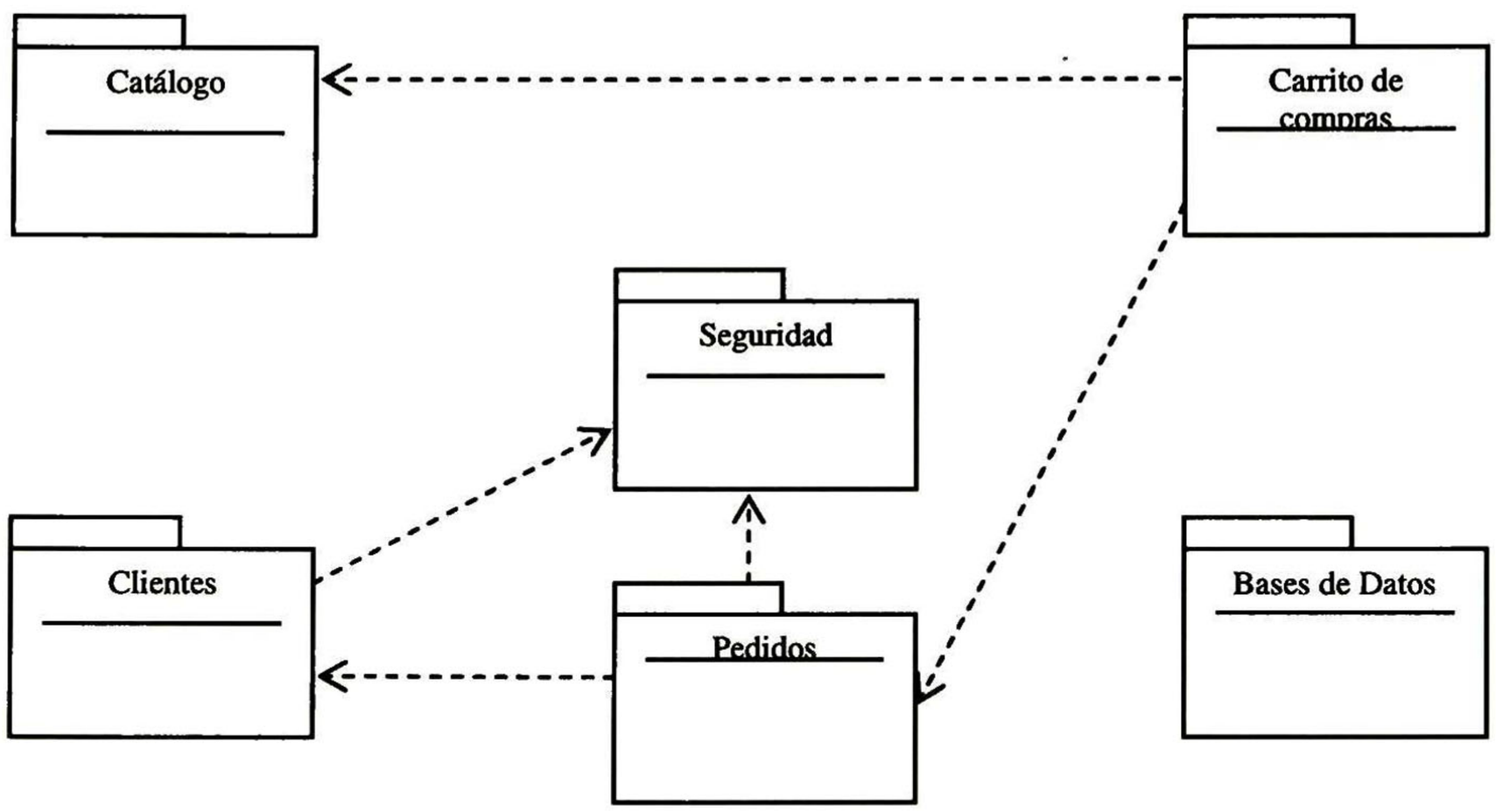
La capa del Servidor puede dividirse en dos: servidor Web y servidor de aplicación. El servidor Web es responsable de realizar todo el procesamiento relacionado con el servicio de páginas HTML y JSP para su presentación en el navegador. El servidor de aplicación es responsable de cualquier procesamiento o lógica de negocio utilizando Enterprise JavaBeans (EJB). Enterprise JavaBeans son componentes de software de negocio los cuales extienden la funcionalidad específica de la aplicación. La interfaz entre estos componentes y sus contenedores está definida en la especificación de EJB. Típicamente los servidores implementan contenedores para EJB, estos contenedores proporcionan servicios como: control de transacciones, administración de seguridad y manejo de persistencia. Los desarrolladores de la

aplicación diseñan componentes EJB enfocándose en la funcionalidad del negocio o lógica de la aplicación.

Capa de Datos

La capa de Datos conforma la infraestructura de información. En esta capa están incluidos sistemas de administración de bases de datos relacionales, sistemas de planeación de recursos (ERP), y otros sistemas externos. El acceso a esta capa se realiza a través de transacciones que mantienen la consistencia de los datos. Esta capa también debe garantizar la seguridad de la información y ofrecer cualidades de facilidad de extensión.

C.6.2 Arquitectura General – Separación en Subsistemas



Bases de Datos. Contiene los elementos de diseño necesarios para el mecanismo de persistencia del sistema.

Carrito de compras. Contiene los elementos para el soporte de los casos de uso relacionados con el carrito de compras: Administrar carrito de compras, Mostrar carrito de compras, Ver carrito de compras.

Catálogo. Contiene los elementos de diseño para el soporte de los casos de uso relacionados con el catálogo de productos: Consultar catálogo.

Clientes. Contiene los elementos de diseño para el soporte de los casos de uso relacionados con clientes: Registrar cliente, Administrar perfil de usuario.

Pedidos. Contiene los elementos de diseño para el soporte de los casos de uso relacionados con pedidos: Consultar pedido, Cancelar pedido, Comprar, Efectuar pago.

Seguridad. Contiene los elementos de diseño para la implementación del mecanismo de seguridad.

C.7 Perspectiva de Emplazamiento

Descripción de los nodos físicos para la configuración de la plataforma física. Esta sección es organizada por medio de la configuración de red física y la asignación de los servicios del sistema a cada nodo. La siguiente figura muestra el diagrama de emplazamiento para el sistema.

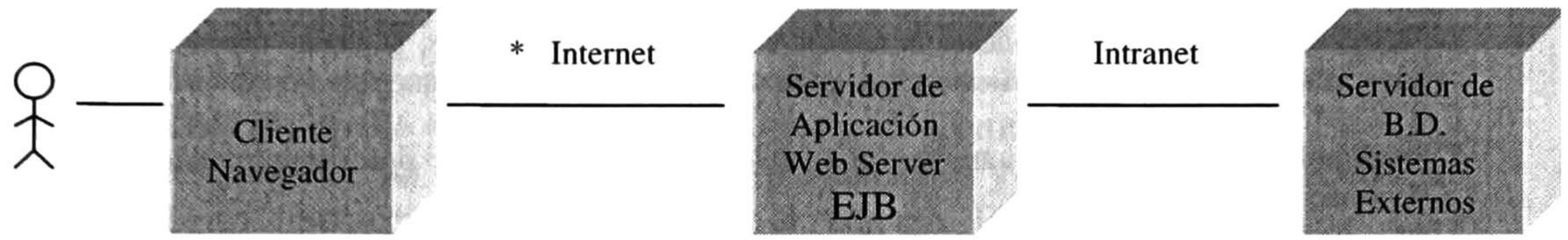


Diagrama de emplazamiento

Cliente. Computadora personal con capacidad para ejecutar un navegador que despliega las páginas Web generadas por Java Server Pages en el servidor. El cliente utiliza el protocolo de comunicación TCP/IP para conectarse con el servidor de aplicación.

Servidor de Aplicación. Computadora con capacidad para ejecutar los servicios Web y los procesos de negocio mediante un contenedor de Enterprise JavaBeans. El servidor de aplicación se comunica vía LAN con el servidor de bases de datos.

Servidor de Bases de Datos. Computadora que ejecuta los procesos para los mecanismos de persistencia o sistemas externos que se comunican con el servidor de aplicación.

C. 8 Tamaño y Desempeño

La mayor parte del tamaño del sistema se localiza en el servidor de aplicación debido al estilo de arquitectura de cliente delgado utilizado. El servidor de aplicación debe contar con el poder de procesamiento necesario para proporcionar tiempos de respuesta adecuados a un número variable de clientes conectados de manera simultánea.

C. 9 Cualidades

Las siguientes cualidades de diseño se tomaron en cuenta para la elaboración de la arquitectura del sistema.

Reutilización de código. La reutilización de código disminuye el costo de nuevos productos, permite mejoras de la calidad y el establecimiento de buenas prácticas de programación en la organización.

Descomposición funcional. Cada clase en el sistema tiene un rol bien definido en el sistema. Lo cual facilita el mantenimiento, la extensión y entendimiento del sistema.

Facilidad de extensión. La funcionalidad de la aplicación puede extenderse de acuerdo a los cambios tecnológicos y de organización que se presenten.

Diseño modular. La descomposición del diseño en módulos que interactúan a través de interfaces bien definidas, permite a los desarrolladores trabajar independientemente, facilita el mantenimiento y las pruebas, y la adquisición de componentes desarrollados externamente.

Seguridad. Dado que la aplicación realiza transacciones financieras, y como instrumento para la privacidad y seguridad de los clientes, la seguridad de la información es una parte importante del sistema.

Facilidad de uso. Las aplicaciones son más fáciles de usar, si el usuario puede intuir donde encontrar la información que desea.

Minimizar el tráfico de datos. La aplicación debe evitar transmitir datos no necesarios o redundantes.

Persistencia. La persistencia de los datos debe ser consistente siempre.

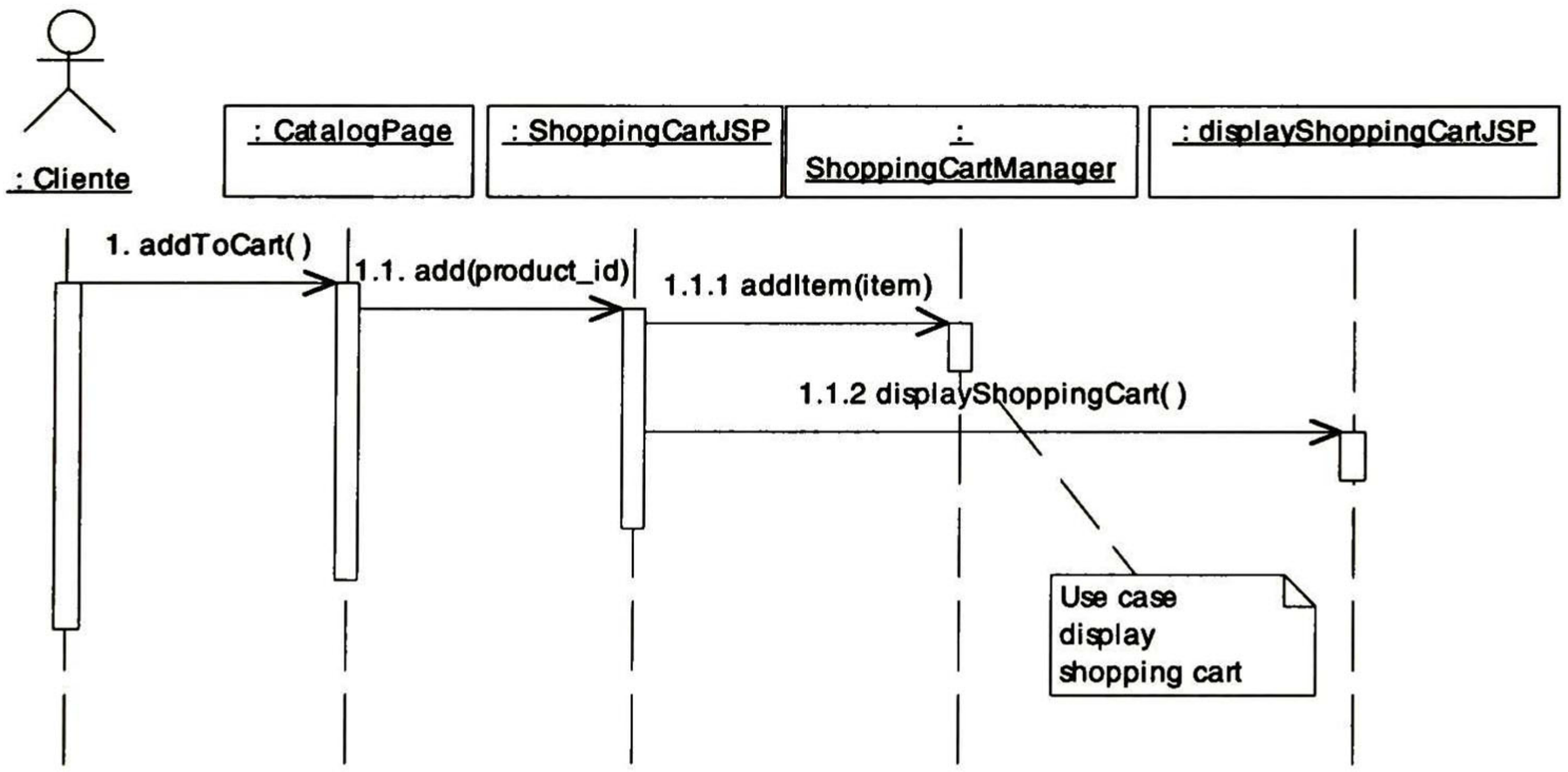
Apéndice D Modelo de Diseño

En este apéndice se presentan las realizaciones de algunos de los principales casos de uso del modelo de diseño del cliente para el Sistema de Comercio Electrónico. En cada caso de uso se incluye un diagrama de clases participantes y el diagrama de secuencia que describe la interacción del usuario con el sistema. En estos diagramas se han utilizado los estereotipos para las clases de diseño propuestos en [Conallen, 1999] y que fueron presentados en la sección 5.3 de esta tesis. El modelo completo de diseño puede consultarse en el Documento de Diseño del Cliente para el Sistema de Comercio Electrónico.

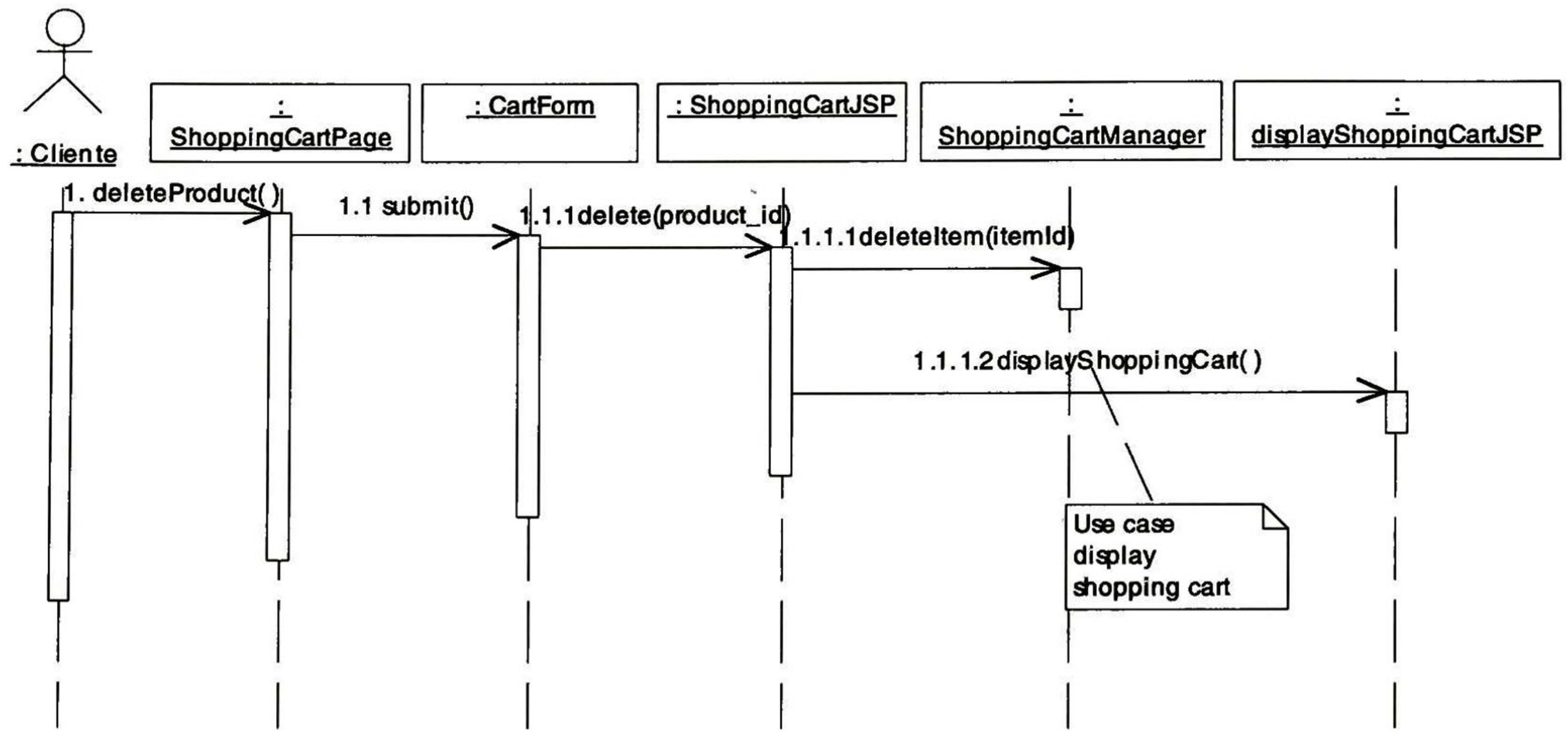
D.1 Administrar Carrito de Compras

Diagramas de secuencia

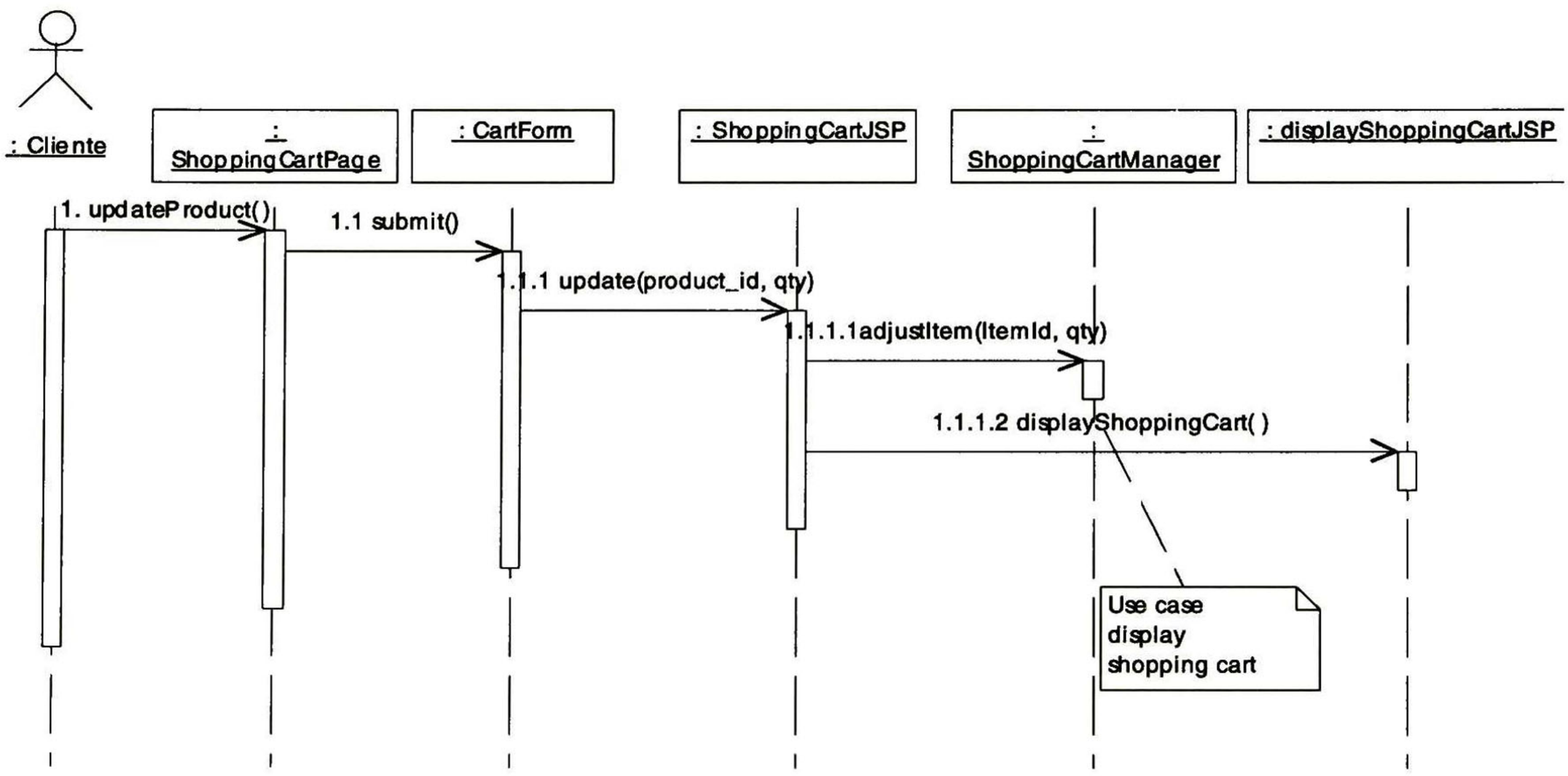
Agregar producto



Eliminar producto



Modificar producto



Vaciar carrito de compras

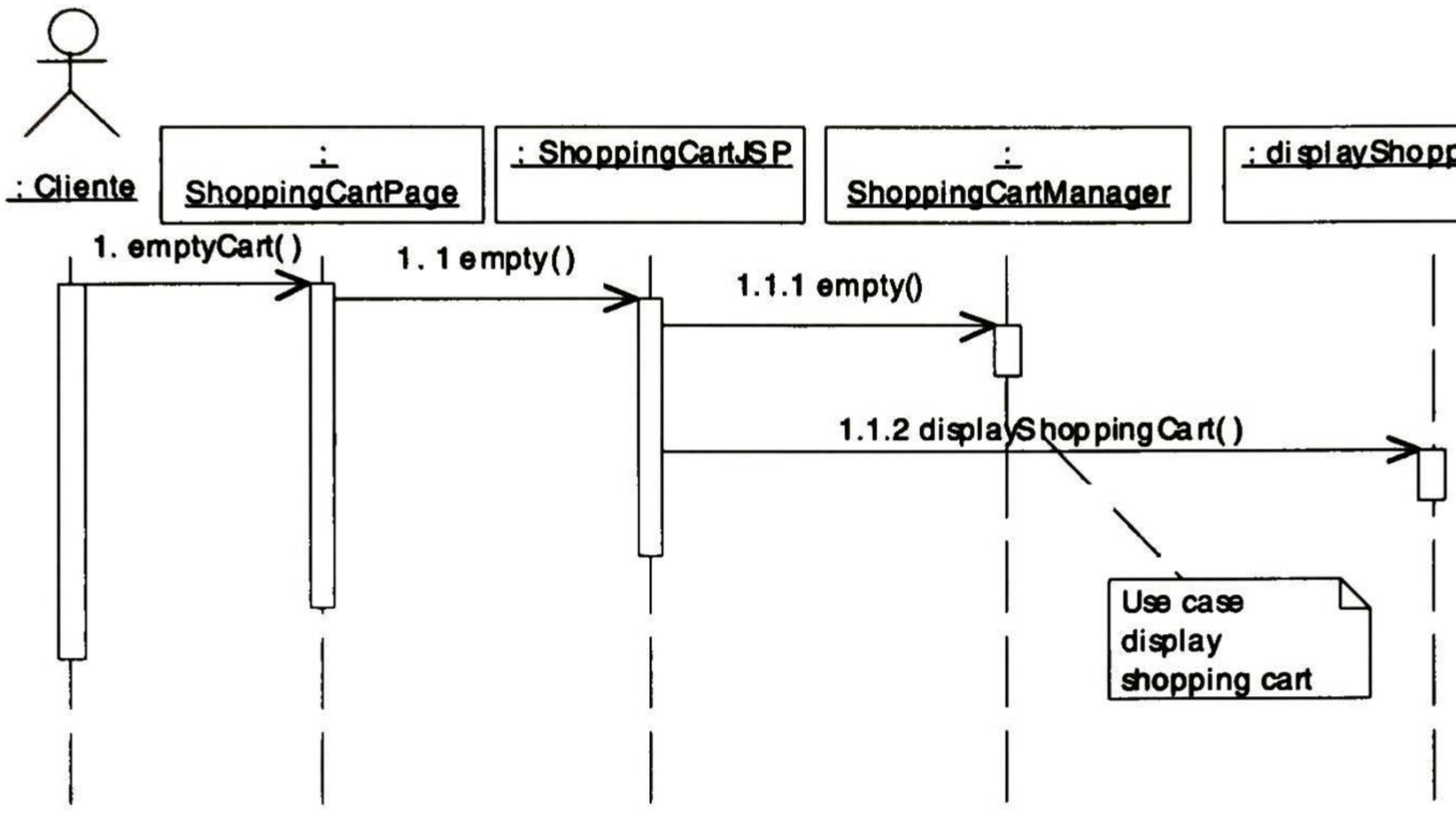
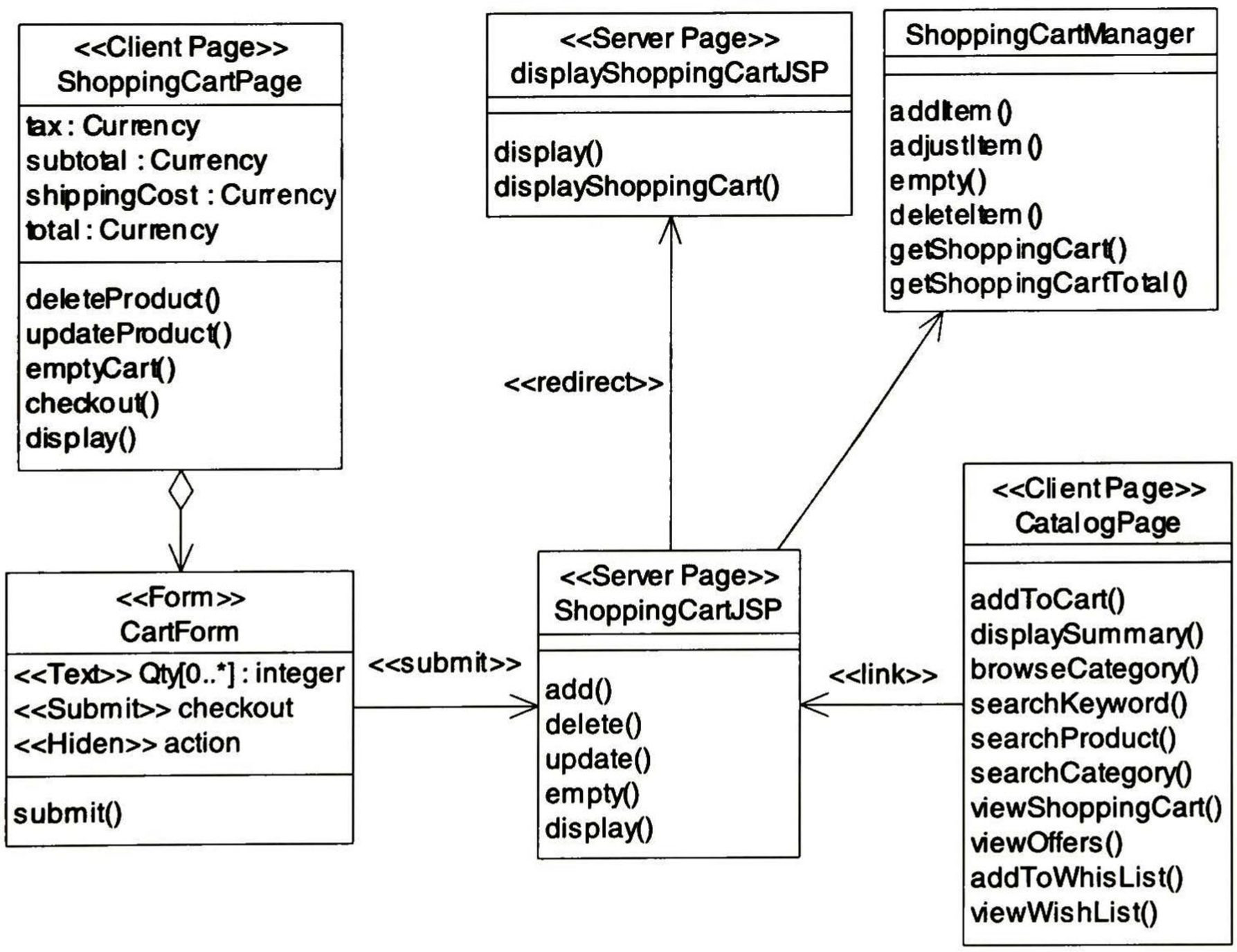


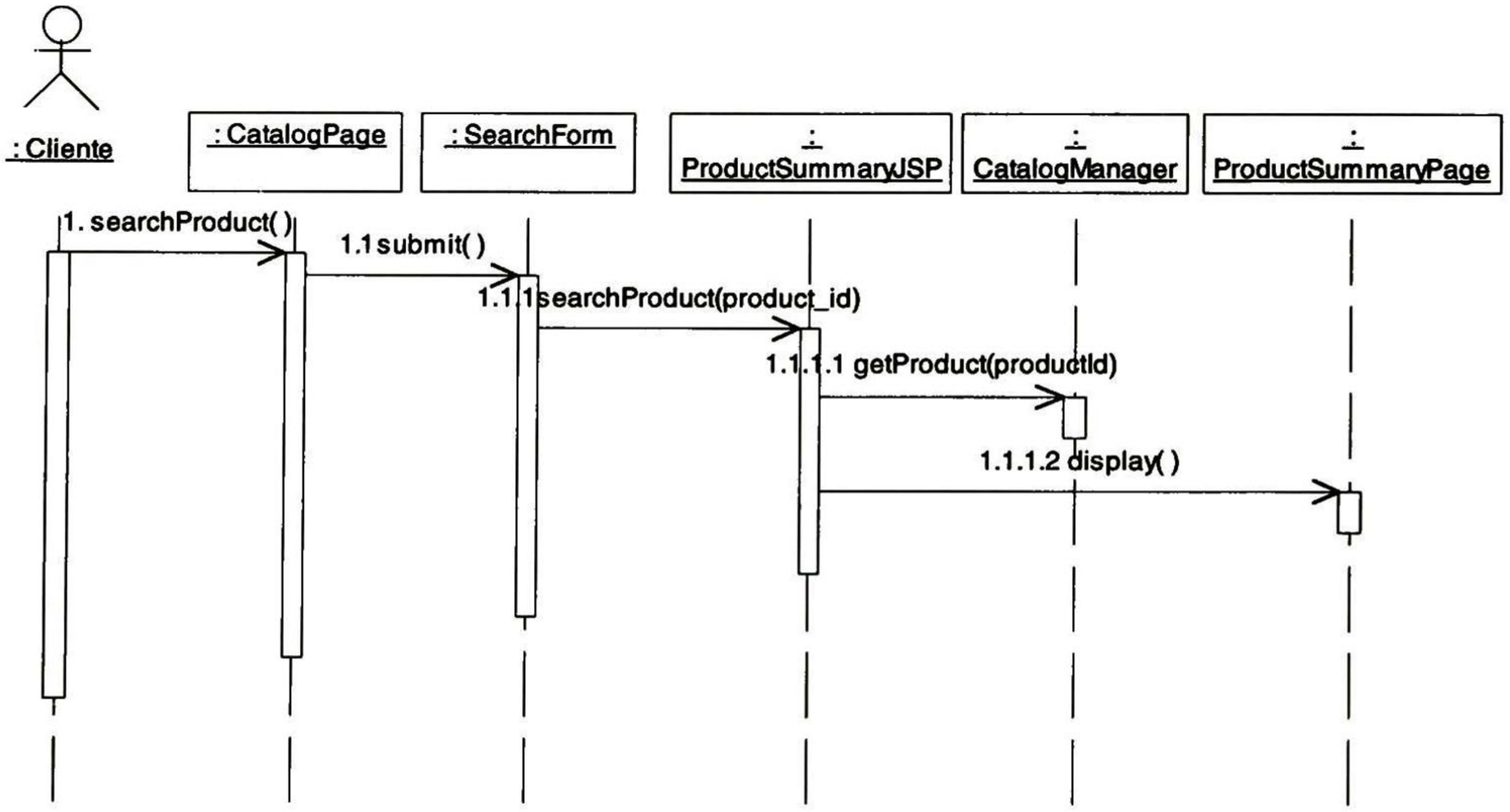
Diagrama de clases



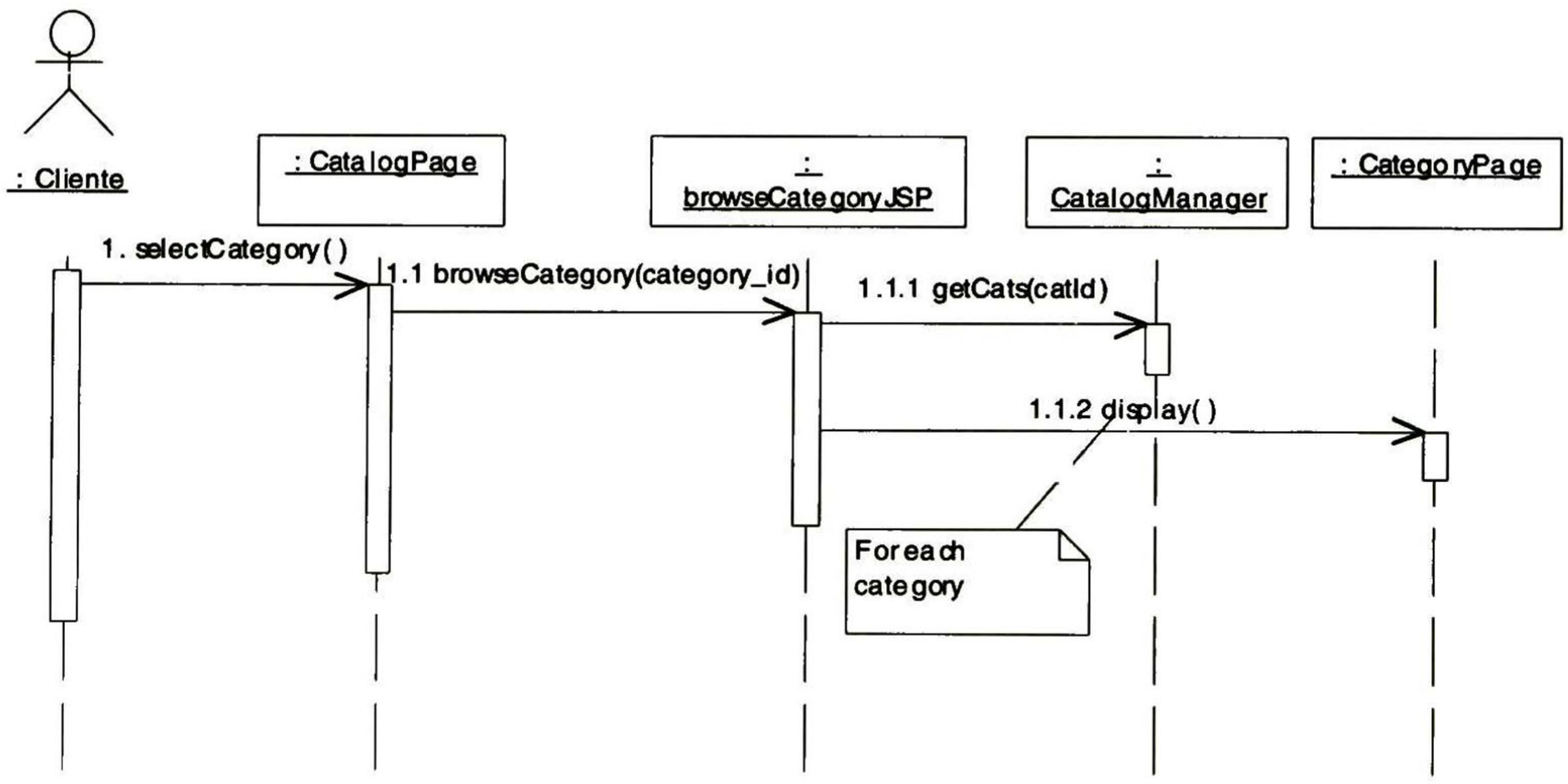
D.2 Consultar Catálogo de Productos

Diagramas de secuencia

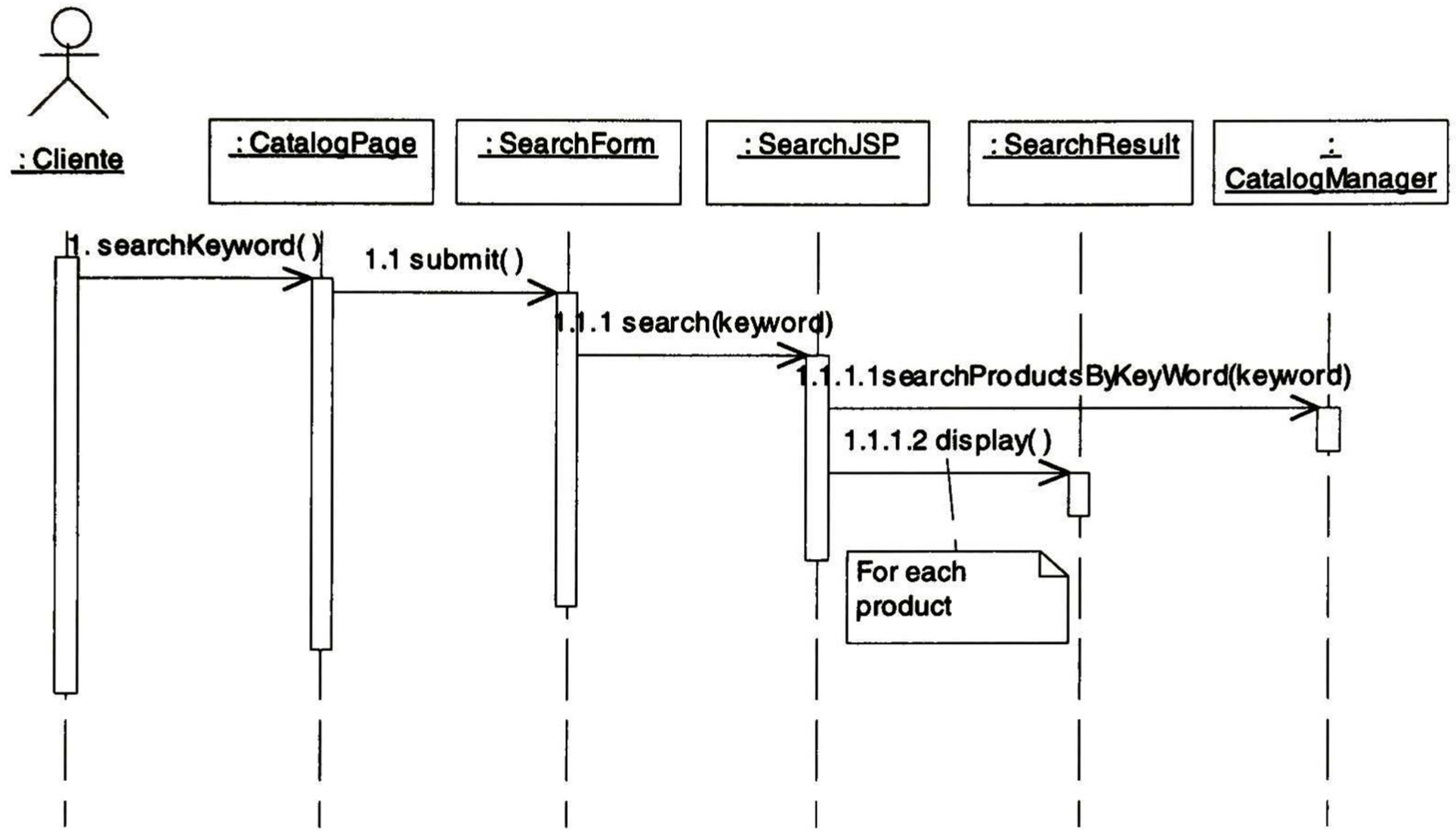
Consultar productos por identificador



Consultar productos por categoría



Consultar productos por palabra clave



Consultar ofertas

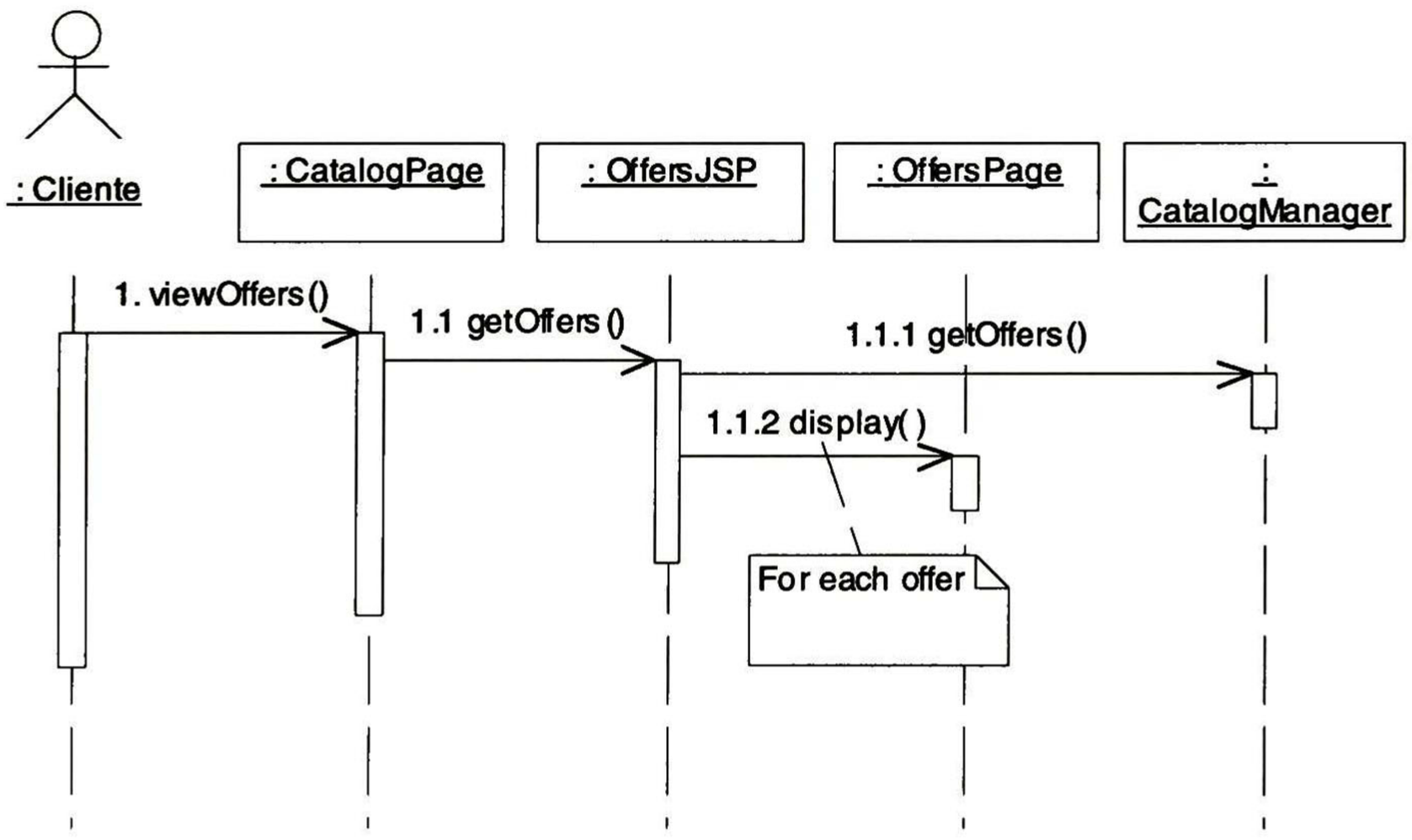
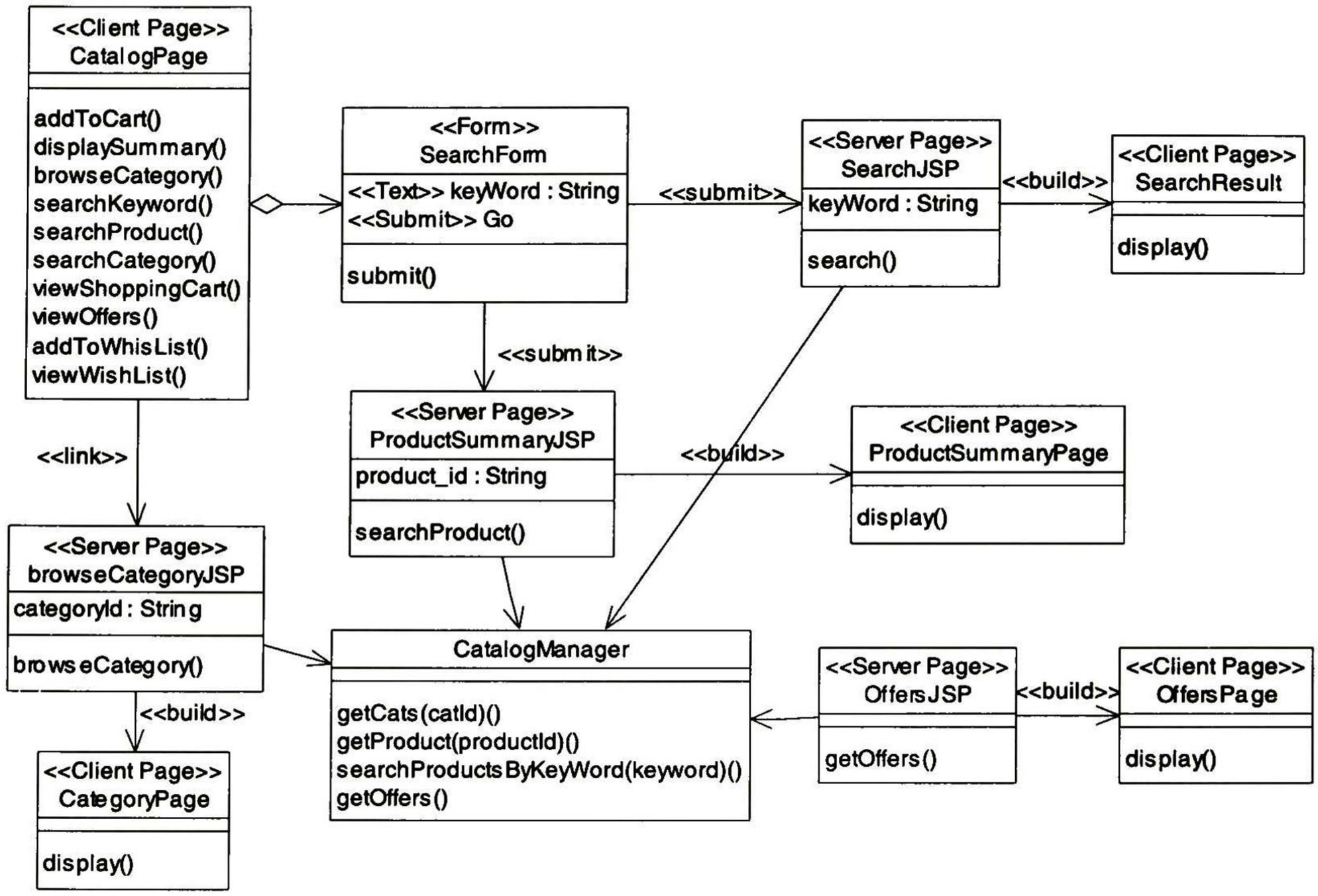


Diagrama de clases



D.3 Comprar

Diagrama de secuencia

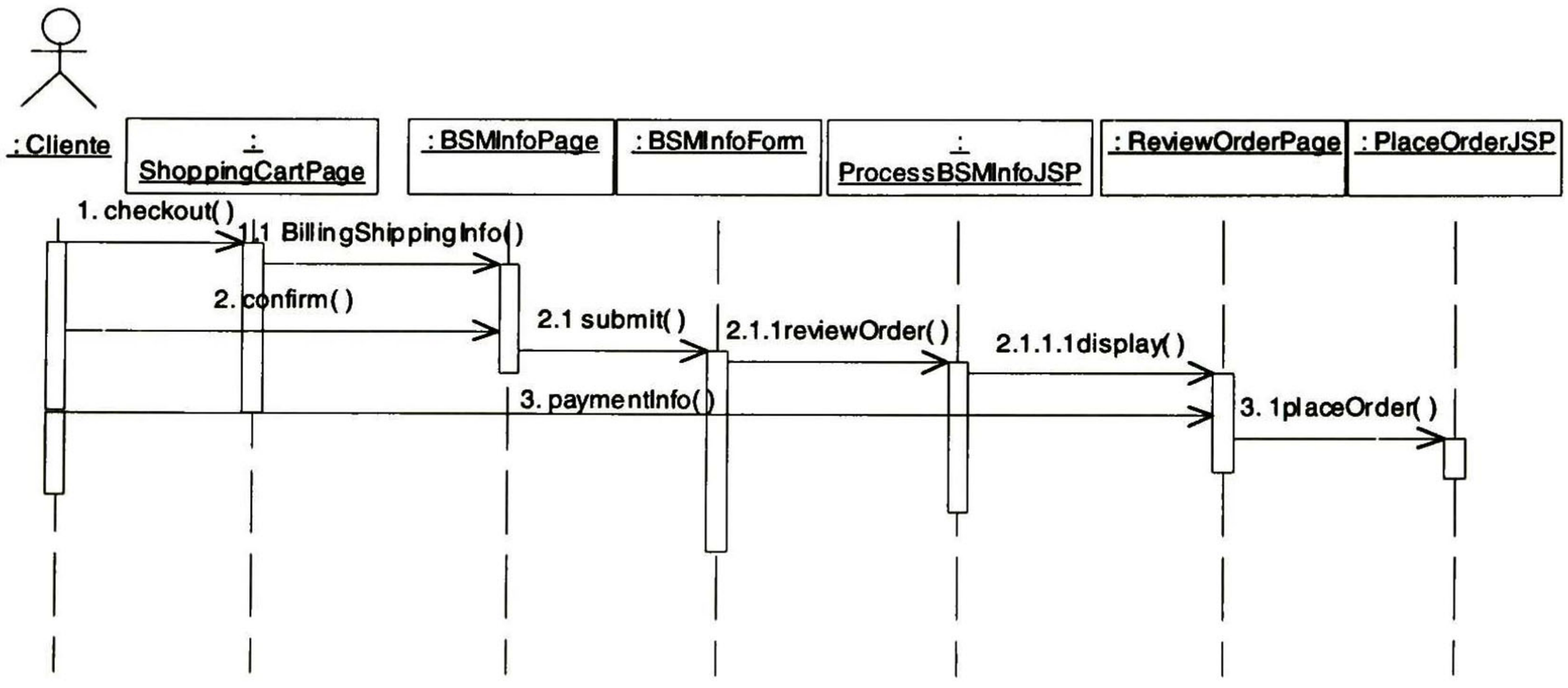
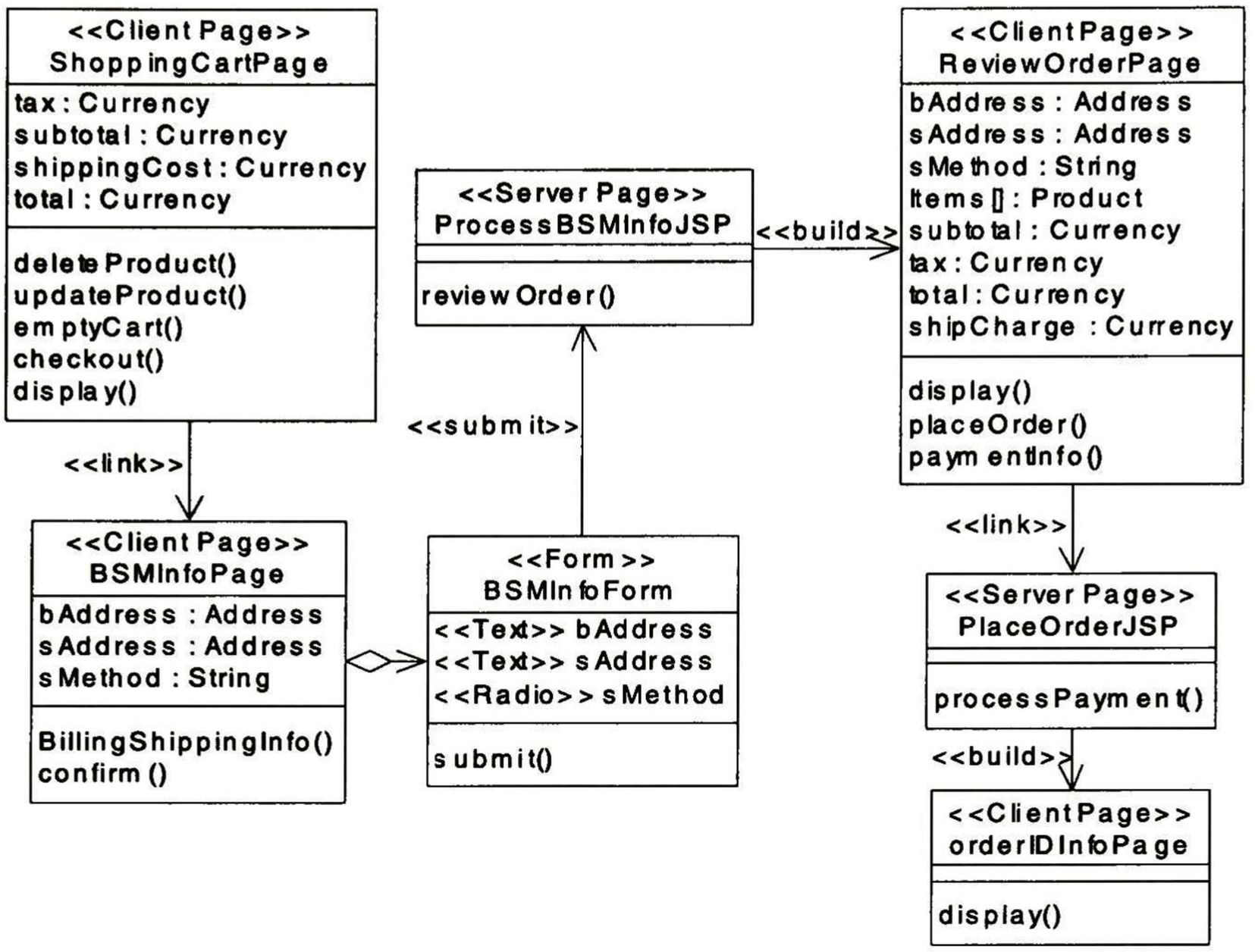


Diagrama de clases



D.4 Efectuar pago

Diagrama de secuencia

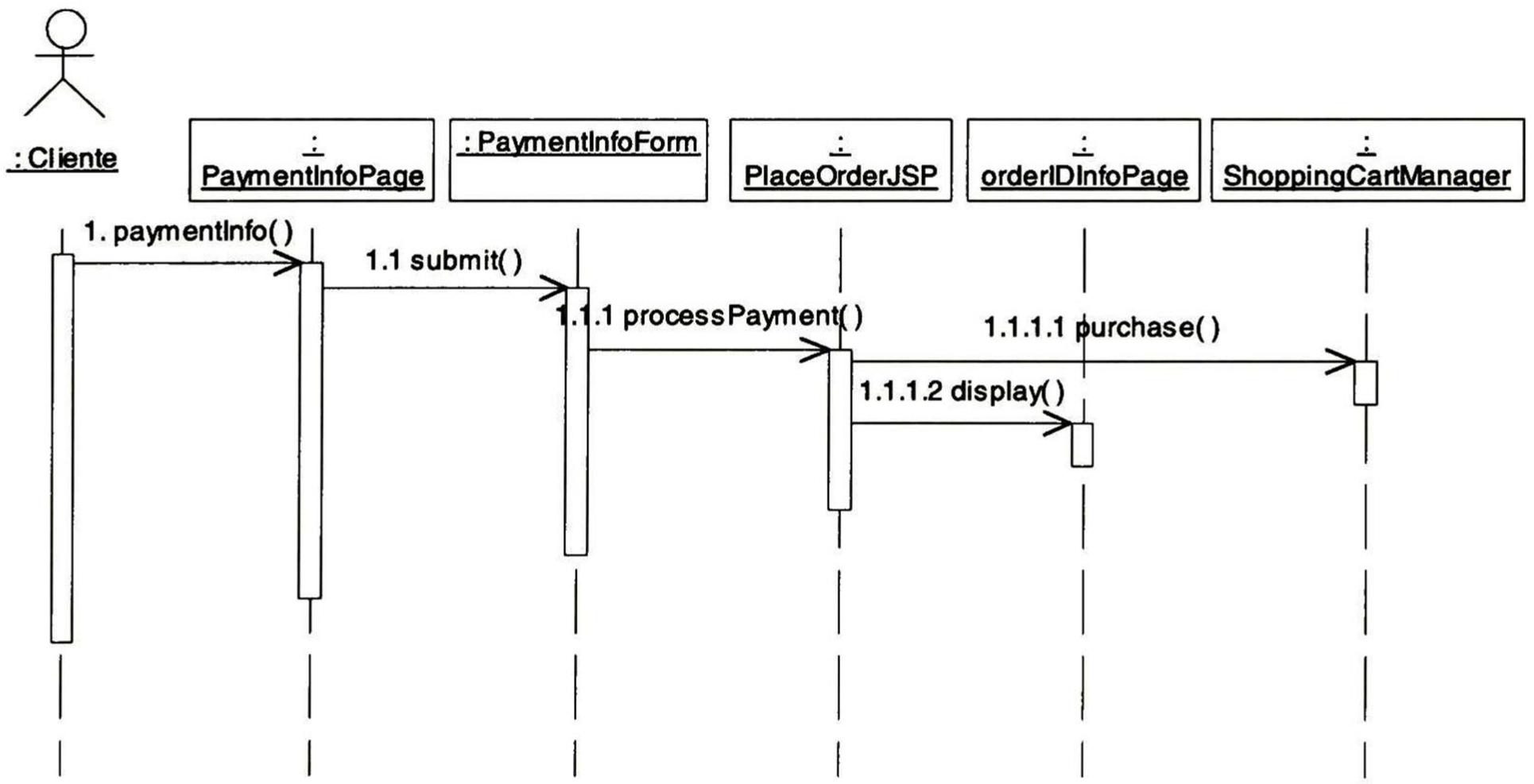
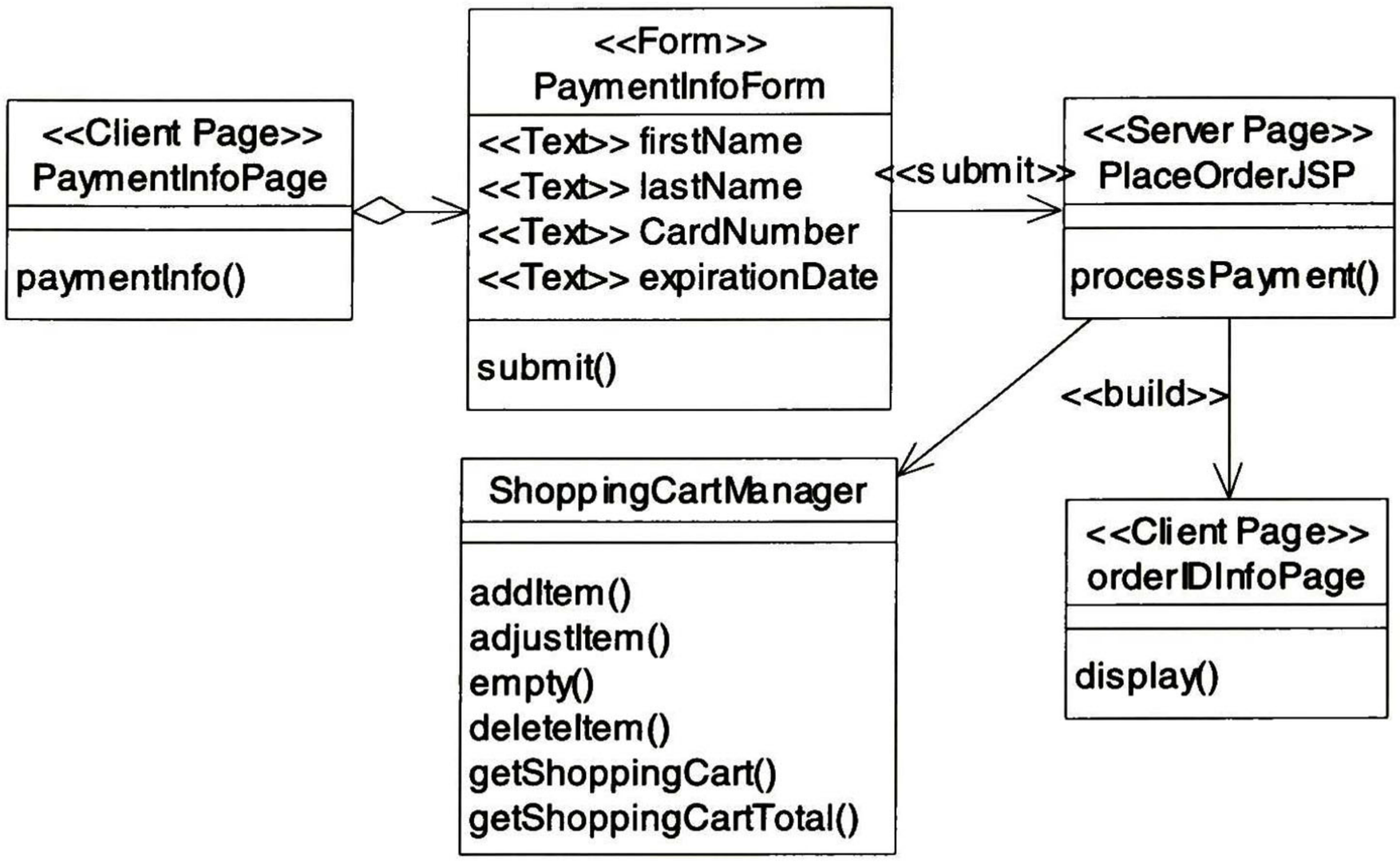


Diagrama de clases



Apéndice E Acrónimos

ASP	Active Server Pages
B2B	Business to Business
B2C	Business to Consumer
CGI	Common Gateway Interface
CSS	Cascading Style Sheet
DBMS	Data Base Management System
DCOM	Distributed Component Object Model
DOM	Document Object Model
DTD	Document Type Definition
EJB	Enterprise JavaBeans
ERP	Enterprise Resource Planning
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
IP	Internet Protocol
IIOP	Internet Inter-ORB Protocol
JSP	JavaServer Pages
JDBC	Java Data Base Connectivity
J2EE	Java 2 Enterprise Edition
LAN	Local Area Network
MVC	Model View Controller
OBI	Open Buying on the Internet
OMG	Object Management Group
PDF	Portable Document Format

RMI	Remote Method Invocation
SAX	Simple API for XML
SGML	Standard Generalized Markup Language
SSL	Secure Socket Layer
TCP	Transmission Control Protocol
UML	Unified Modeling Language
W3C	World Wide Web Consortium
WAP	Wireless Application Protocol
WML	Wireless Markup Language
WWW	World Wide Web
XLL	Extensible Linking Language
XML	Extensible Markup Language
XPL	Extensible Pointer Language
XSL	Extensible Stylesheet Language

Bibliografía

[Abowd, 1996] Abowd, G., Baas, L., Clements, P., Kazman, R., Northrop, L., and Zaremski, A., "Recommended Best Industrial Practice for Software Architecture Evaluation." Software Engineering Institute, Carnegie Mellon University, Technical Report CMU/SEI-96-TR-025,1996.

[Amor, 2000] Amor, D., *The E-business (r)evolution*, Prentice Hall, 2000.

[André, 2003] André, R., *Modelo de Datos para Comercio Electrónico*, Tesis en desarrollo.

[Baas et al, 1998] Bass, L., P. Clements and R. Kazman, *Software Architecture in practice*, Addison Wesley.

[Booch, 1999] Booch, G., J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*, Reading, MA: Addison Wesley, 1999.

[Chang et al, 1999] Chang, D. and D. Harkey, *Client/Server data access with Java and XML*, John Wiley, Inc., 1998.

[Comer, 1997] Comer, D.E., *Computer networks and internets*, Prentice Hall, 1997.

[Cortés, 2003] Cortés, A., *Seguridad para Comercio Electrónico*, Tesis en desarrollo.

[Conallen, 1999] Conallen, J., *Building Web applications with UML*, Addison Wesley, 1999.

[Gamma, 1995] Gamma, E., Helm, R., Johnson, R., and Vlissides, J., *Design Patterns, Elements of reusable Object-Oriented Software*. Reading, MA: Addison Wesley, 1995.

[Garlan, 1995] Garlan, D. Allen, R., and Ockerbloom, J., "Architectural Mismatch: Why Reuse Is So Hard." *IEEE Software*,12(6):17-26,1995.

[Herzum et al, 2000] Herzum, P., and O. Sims, *Business component factory: a comprehensive overview of component based development for the enterprise*, John Wiley & Sons, 2000.

[IEEE, 1995] Special Issue on Software Architecture, IEEE Transactions on Software Engineering, 21(4), April 1995.

[Jacobson, 1999] Jacobson, I., Booch, G., Rumbaugh, J., *The Unified Software Development Process*, Addison Wesley, 1999.

[JSP, 2000] Sun Microsystems, Inc., JavaServer *Pages Technology White Paper,* http://java.sun.com/products/jsp/whitepaper.html, 2000.

[Kirtland, 1999] Kirtland, M., *Designing component based applications*, Microsoft Press, 1999.

[Korper, 2000] Korper, S., Ellis, J., *The E-commerce Book*, Academic Press, 2000.

[Krasner, 1988] Krasner, G., and Pope, S., "A Cookbook for Using Model-View-Controller User Interface Paradigm in Smalltalk-8." *Journal of Object Oriented Programming*, 1(3):26-49, 1988.

[McLaughlin, 2000] McLaughlin, B., *Java and XML*, O'Reilly, 2000.

[Orfali et al, 1999] Orfali, R, D. Harkey and J. Edwards, *The essential client/server survival guide*, John Wiley and Sons, Inc., 1999.

[Parnas, 1972] Parnas, D., "On the Criteria for Decomposing Systems into Modules." CACM, 15(12):1053-1058, 1972.

[Parnas, 1974] Parnas, D., "On a 'Buzzword': Hierarchical Structure." *Proceedings of the IFIP Congress*, 74:336-390, 1974.

[Parnas, 1976] Parnas, D., "On the Design and Development of Program Families." *IEEE Transactions on Software Engineering*, SE-2(1):1-9, 1976.

[Parnas, 1979] Parnas, D., "Designing Software for Ease of Extension and Contraction." *IEEE Transactions on Software Engineering*, SE-5 (2): 128-137, 1979.

[Pfaff, 1985] Pfaff, G.E., (Ed.), *User Interface Systems* (Eurographics Seminars). New York: Springer-Verlag, 1985.

[Reyes, 2003] Reyes, J., *Tecnologías para Comercio Electrónico*, Tesis en desarrollo.

[Shaw, 1997] Shaw, M., and Clements, P., "A Field Guide to Boxology: Preliminary Classification of Architectural Styles for Software Systems." Proceedings of COMPSAC, Washington, D.C., August 1997.

ANIVERSARIO
Cinvestav

Centro de Investigación y de Estudios Avanzados del IPN

Unidad Guadalajara

El Jurado designado por la Unidad Guadalajara del Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, aprobó la tesis: ARQUITECTURA DE SOFTWARE PARA UN SISTEMA DE COMERCIO ELECTRÓNICO B2C del(a) C. Jaime Arturo CEDILLO NAVARRO el día 14 de Marzo de 2003

DR. ARTURO DEL SAGRADO CORAZÓN SÁNCHEZ CARMONA
INVESTIGADOR CINVESTAV 3B
CINVESTAV GDL GUADALAJARA

DR. MANUEL EDGARDO GUZMÁN RENTERIA
INVESTIGADOR CINVESTAV 3A
CINVESTAV GDL GUADALAJARA

DR. DENI LIBRADO TORRES ROMÁN
PROFESOR INVESTIGADOR 3A
CINVESTAV GDL GUADALAJARA