Construyendo aplicaciones web con una metodología de diseño orientada a objetos

Darío Andrés Silva* Bárbara Mercerat*

Resumen

El presente artículo tiene como principal objetivo mostrar las ventajas del uso de una metodología de diseño orientada a objetos para desarrollar aplicaciones *web*.
Existen en la actualidad tecnologías que permiten un rápido desarrollo de aplicaciones poco reusables y difíciles de mantener. La metodología propuesta en este artículo, aplicada con las tecnologías brevemente descritas, permite obtener aplicaciones mediante un proceso de desarrollo en capas, aprovechando al máximo la potencia de la programación orientada a objetos.

Palabras claves: *Aplicaciones web, tecnologías de desarrollo, programación orientada a objetos, diseño en capas, contenido dinámico, patrones de diseño*

Abstract

The main goal of this paper is to show the advantages of using an object oriented design method to develop *web* applications.
Nowadays, there are technologies that allow a fast development of applications with a poor level of reuse and very difficult to maintain. The method proposed in this paper, applied with the briefly described technologies, allows the developers to obtain applications in a layered development process, taking advantages of the power of object oriented programming.

Keywords: *web applications, development technologies, object oriented programming, layered design, dynamic content, design patterns*

1 Introducción

El desarrollo de aplicaciones *web* involucra decisiones no triviales de diseño e implementación que inevitablemente influyen en todo el proceso de desarrollo, afectando la división de tareas. Los problemas involucrados, como el diseño del modelo del dominio y la construcción de la interfaz de usuario, tienen requerimientos disjuntos que deben ser tratados por separado.

El alcance de la aplicación y el tipo de usuarios a los que estará dirigida son consideraciones tan importantes como las tecnologías elegidas para realizar la implementación. Así como las tecnologías pueden limitar la funcionalidad de la aplicación, decisiones de diseño equivocadas también pueden reducir su capacidad de lextensión y reusabilidad. Es por ello que el uso de una metodología de diseño y de tecnologías que se adapten naturalmente a ésta, son de vital importancia para el desarrollo de aplicaciones complejas.

* LIFIA, Laboratorio de Investigación y Formación en Informática Avanzada. Facultad de Informática, Universidad Nacional de La Plata. Calle 50 esq. 115 (1900), La Plata, Provincia de Buenos Aires, República Argentina. {dsilva,bmercerat}@lifia.info.unlp.edu.ar

Existen en la actualidad tecnologías ampliamente usadas para el desarrollo de aplicaciones *web*, pero muchas de ellas obligan al desarrollador a mezclar aspectos conceptuales y de presentación. Esto sucede principalmente con aquellas tecnologías no basadas en objetos, y por esta razón no serán mencionadas en este artículo.

La elección de tecnologías complejas demora el proceso e incrementa los costos, pero en ocasiones permite adecuarse a metodologías de diseño más fácilmente. Tal es el caso de las tecnologías orientadas a objetos, las cuales tienden a demorar el desarrollo en etapas tempranas. El tiempo de desarrollo en la actualidad es crítico, tanto por razones de marketing como por límites en el presupuesto y los recursos [1], pero la adopción de estas tecnologías hace que el mantenimiento se transforme en una actividad más simple, la división en capas sea tarea natural del desarrollo y el tiempo invertido en el diseño facilite el trabajo necesario para el resto de las actividades.

El objetivo de este artículo es presentar una metodología de diseño de aplicaciones *web*, y mostrar por medio de un ejemplo su implementación con las tecnologías adecuadas para cada capa de diseño.

En la siguiente sección se presenta una introducción a las aplicaciones *web* en general, haciendo hincapié en la importancia del diseño en capas y en la programación orientada a objetos como herramienta de desarrollo.

En la sección 3 se describe cómo realizar el diseño de las aplicaciones *web*, qué capas involucra y en qué consisten cada una de ellas. Se introduce para ello a OOHDM (Método de Diseño Hipermedia Orientado a Objetos[1]) [2], una metodología de diseño de aplicaciones hipermedia, y en particular de aplicaciones *web*.

En la sección 4 se describen las características sobresalientes de las tecnologías sugeridas para la implementación de una aplicación concreta construida con la metodología.

En la sección 5 se presenta una idea de implementación basada en un ejemplo simple, utilizando las tecnologías sugeridas, en función de los requerimientos de cada capa de diseño.

Finalmente, se resumen conclusiones sobre el uso conjunto de la metodología y las tecnologías elegidas.

2 Aplicaciones *web* y la importancia del desarrollo en capas

Las aplicaciones hipermedia han evolucionado en los últimos años y se han concentrado mayormente en la *web*. Las antiguas aplicaciones distribuidas en cd's dieron lugar a aplicaciones dinámicas, de constante actualización e incluso personalizables, capaces de adaptarse a los tipos de usuarios y en casos avanzados, a cada usuario en particular. Estas características encuentran el medio ideal en la *web*, ya que de otra forma sería costoso su mantenimiento y evolución.

La complejidad del desarrollo [3] ocurre a diferentes niveles: dominios de aplicación sofisticados (financieros, médicos, geográficos, etc.); la necesidad de proveer acceso de navegación simple a grandes cantidades de datos multimediales, y por último la aparición de nuevos dispositivos para los cuales se deben construir interfaces *web* fáciles de usar. Esta complejidad en los desarrollos de software sólo puede ser alcanzada mediante la separación de los asuntos de modelización en forma clara y modular.

La metodología OOHDM [4], presentada en la próxima sección, ha sido utilizada para diseñar diferentes tipos de aplicaciones hipermedia como galerías interactivas, presentaciones multimedia y como veremos en este artículo, aplicaciones *web*. El éxito de esta metodología es la clara identificación de los tres diferentes niveles de diseño en forma independiente de la implementación.

[1] Object Oriented Hypermedia Design Method

La justificación de tanto trabajo puede encontrarse en cualquier aplicación que requiera navegación: en términos de programación orientada a objetos, si los elementos por los que se navega son los del diseño conceptual se estaría mezclando la funcionalidad hipermedia con el comportamiento propio del objeto. Por otro lado, si los nodos de la red de navegación tienen la capacidad de definir su apariencia, se estaría limitando la extensión de la aplicación para ofrecer nuevas presentaciones del mismo elemento y eventualmente se estaría dificultando la personalización de la interfaz.

Es necesario, entonces, mantener separadas las distintas decisiones de diseño según su naturaleza (conceptual, navegacional, de interfaz) y aplicar las tecnologías adecuadas a cada capa en el proceso de implementación.

Esta idea de desarrollo será discutida en detalle en el resto del artículo, donde con ayuda de un ejemplo concreto se podrá apreciar con claridad la real importancia del uso de una metodología de diseño.

3 Introducción a OOHDM

Las metodologías tradicionales de ingeniería de software, o las metodologías para sistemas de desarrollo de información, no contienen una buena abstracción capaz de facilitar la tarea de especificar aplicaciones hipermedia. El tamaño, la complejidad y el número de aplicaciones crecen en forma acelerada en la actualidad, por lo cual una metodología de diseño sistemática es necesaria para disminuir la complejidad y admitir evolución y reusabilidad.

Producir aplicaciones en las cuales el usuario pueda aprovechar el potencial del paradigma de la navegación de sitios *web*, mientras ejecuta transacciones sobre bases de información, es una tarea muy difícil de lograr. En primer lugar, la navegación posee algunos problemas. Una estructura de navegación robusta es una de las claves del éxito en las aplicaciones hipermedia. Si el usuario entiende dónde puede ir y cómo llegar al lugar deseado, es una buena señal de que la aplicación ha sido bien diseñada.

Construir la interfaz de una aplicación *web* es también una tarea compleja; no sólo se necesita especificar cuáles son los objetos de la interfaz que deberían ser implementados, sino también la manera en la cual estos objetos interactuarán con el resto de la aplicación.

En hipermedia existen requerimientos que deben ser satisfechos en un entorno de desarrollo unificado [2]. Por un lado, la navegación y el comportamiento funcional de la aplicación deberían ser integrados. Por otro lado, durante el proceso de diseño se debería poder desacoplar las decisiones de diseño relacionadas con la estructura navegacional de la aplicación, de aquellas relacionadas con el modelo del dominio.

OOHDM propone el desarrollo de aplicaciones hipermedia a través de un proceso compuesto por cuatro etapas: diseño conceptual, diseño navegacional, diseño de interfaces abstractas e implementación.

3.1 Diseño Conceptual

Durante esta actividad se construye un esquema conceptual representado por los objetos del dominio, las relaciones y colaboraciones existentes establecidas entre ellos. En las aplicaciones hipermedia convencionales, cuyos componentes de hipermedia no son modificados durante la ejecución, se podría usar un modelo de datos semántico estructural (como el modelo de entidades y relaciones). De este modo, en los casos en que la información base pueda cambiar dinámicamente o se intenten ejecutar cálculos complejos, se necesitará enriquecer el comportamiento del modelo de objetos.

[2] framework

En OOHDM, el esquema conceptual está construido por clases, relaciones y subsistemas. Las clases son descritas como en los modelos orientados a objetos tradicionales. Sin embargo, los atributos pueden ser de múltiples tipos para representar perspectivas diferentes de las mismas entidades del mundo real.

Se usa notación similar a UML (Lenguaje de Modelado Unificado[3]) y tarjetas de clases y relaciones similares a las tarjetas CRC (Clase Responsabilidad Colaboración[4]). El esquema de las clases consiste en un conjunto de clases conectadas por relaciones. Los objetos son instancias de las clases. Las clases son usadas durante el diseño navegacional para derivar nodos, y las relaciones que son usadas para construir enlaces.

3.2 Diseño Navegacional

La primera generación de aplicaciones *web* fue pensada para realizar navegación a través del espacio de información, utilizando un simple modelo de datos de hipermedia. En OOHDM, la navegación es considerada un paso crítico en el diseño aplicaciones. Un modelo navegacional es construido como una *vista* sobre un diseño conceptual, admitiendo la construcción de modelos diferentes de acuerdo con los diferentes perfiles de usuarios. Cada modelo navegacional provee una vista subjetiva del diseño conceptual.

El diseño de navegación es expresado en dos esquemas: el esquema de clases navegacionales y el esquema de contextos navegacionales. En OOHDM existe un conjunto de tipos predefinidos de clases navegacionales: nodos, enlaces y estructuras de acceso. La semántica de los nodos y los enlaces son las tradicionales de las aplicaciones hipermedia, y las estructuras de acceso, tales como índices o recorridos guiados, representan los posibles caminos de acceso a los nodos.

La principal estructura primitiva del espacio navegacional es la noción de contexto navegacional. Un contexto navegacional es un conjunto de nodos, enlaces, clases de contextos, y otros contextos navegacionales (contextos anidados). Pueden ser definidos por comprensión o extensión, o por enumeración de sus miembros.

Los contextos navegacionales juegan un rol similar a las colecciones y fueron inspirados sobre el concepto de contextos anidados. Organizan el espacio navegacional en conjuntos convenientes que pueden ser recorridos en un orden particular y que deberían ser definidos como caminos para ayudar al usuario a lograr la tarea deseada.

Los nodos son enriquecidos con un conjunto de clases especiales que permiten de un nodo observar y presentar atributos (incluidos las anclas), así como métodos (comportamiento) cuando se navega en un particular contexto.

3.3 Diseño de Interfaz Abstracta

Una vez que las estructuras navegacionales son definidas, se deben especificar los aspectos de interfaz. Esto significa definir la forma en la cual los objetos navegacionales pueden aparecer, cómo los objetos de interfaz activarán la navegación y el resto de la funcionalidad de la aplicación, qué transformaciones de la interfaz son pertinentes y cuándo es necesario realizarlas.

---

[3] *Unified Modeling Language*
[4] *Class Responsibility Collaboration*

Una clara separación entre diseño navegacional y diseño de interfaz abstracta permite construir diferentes interfaces para el mismo modelo navegacional, dejando un alto grado de independencia de la tecnología de interfaz de usuario.

El aspecto de la interfaz de usuario de aplicaciones interactivas (en particular las aplicaciones *web*) es un punto crítico en el desarrollo que las modernas metodologías tienden a descuidar. En OOHDM se utiliza el diseño de interfaz abstracta para describir la interfaz del usuario de la aplicación de hipermedia.

El modelo de interfaz ADVs (Vista de Datos Abstracta[5]) [5] especifica la organización y comportamiento de la interfaz, pero la apariencia física real o de los atributos, y la disposición de las propiedades de las ADVs en la pantalla real son hechas en la fase de implementación.

3.4 Implementación

En esta fase, el diseñador debe implementar el diseño. Hasta ahora, todos los modelos fueron construidos en forma independiente de la plataforma de implementación; en esta fase es tenido en cuenta el entorno particular en el cual se va a correr la aplicación.

Al llegar a esta fase, el primer paso que debe realizar el diseñador es definir los ítems de información que son parte del dominio del problema. Debe identificar también, cómo son organizados los ítems de acuerdo con el perfil del usuario y su tarea; decidir qué interfaz debería ver y cómo debería comportarse. A fin de implementar todo en un entorno *web*, el diseñador debe decidir además qué información debe ser almacenada.

4 Comparación de OOHDM con otras metodologías

La comparación de métodos de desarrollo de sistemas de software es una tarea difícil [6]. El foco de cada metodología puede ser diferente, algunas tratan de concentrarse en varios aspectos del proceso de desarrollo, otras tratan de detallar en profundidad algún aspecto en particular. En la Tabla 1 se presenta una comparación de distintas metodologías extraída de [6], teniendo en cuenta los pasos que componen el proceso, la técnica de modelado, la representación gráfica, la notación elegida para los modelos y la herramienta *CASE* de soporte proporcionada para el desarrollo.

Las metodologías comparadas son: HDM (Método de Diseño Hipermedia[6]) [7], RMM (Metodología de Administración de Relaciones[7]) [8], EORM (Metodología de Relaciones de Objetos Mejorada[8]) [9], OOHDM, SOHDM (Metodología de Diseño Hipermedia orientada a objetos y basada en escenarios[9]) [10], WSDM (Método de Diseño de Sitios *Web*[10]) [11], y WAE-Proceso *Conallen* (Extensión de Aplicación *Web* para UML[11]) [12].

Tabla 1. Comparación de OOHDM con otras metodologías

	Proceso	Técnica de modelado	Representación gráfica	Notación	Herramienta de soporte

[5] *Abstract Data View*
[6] *Hypermedia Design Method*
[7] *Relationship Management Methodology*
[8] *Enhanced Object Relationship Methodology*
[9] *Scenario-based Object-oriented Hypermedia Design Methodology*
[10] *Web Site Design Method*
[11] *Web Application Extension for UML – Process Conallen*

HDM	1.Desarrollo a largo plazo 2.Desarrollo a corto plazo	E-R[12]	1.-2.Diagrama E-R	1.E-R	
RMM	1.Diseño E-R 2.Diseño *Slice*[13] 3.Diseño de navegación 4.Diseño de protocolo de conversión 5.Diseño de UI[14] 6.Diseño de comportamiento en tiempo de ejecución 7.Prueba y construcción	E-R	1.Diagrama E-R 2.Diagrama *Slice* 3.Diagrama RMDM[15]	1.E-R 2.3.Propio	*RMCase*
EORM	1.Clases del entorno de desarrollo 2.Composición del entorno de desarrollo 3.Entorno de desarrollo de UI	OO[16]	1.Diagrama de clases 2.Diseño GUI[17]	1.OMT[18]	*ONTOS Studio*
OOHDM	1.Diseño conceptual 2.Diseño navegacional 3.Diseño abstracto de la UI 4.Implementación	OO	1.Diagrama de clases 2.Diagrama navegacional, clase + contexto 3.Diagrama de configuración de ADV + Diagrama ADV	1.OMT/ UML[19] 2.Propio 3.ADVs	*OOHDM-Web*
SOHDM	1.Análisis del dominio 2.Modelo en OO 3.Diseño de la vista 4.Diseño navegacional 5.Diseño implementación 6.Construcción	Escenarios Vistas-OO	1.Diagramas de escenarios de actividad 2.Diagrama de estructura de clase 3.Vista OO 4.Esquema de enlace navegacional 5.Esquema de páginas	1.-5.Propio	
WSDM	1.Modelado del usuario 2.Diseño conceptual 2.1.Modelo objetos 2.2.Diseño navegacional 3.Diseño implementación 4.Implementación	E-R/ OO	1.Diagrama de E-R o clase 2.Capas de navegación	1.E-R/ OMT 2.Propio	
WAE-Proceso *Conallen*	1.Manejo de proyecto 2.Captura de requerimientos 3.Análisis 4.Diseño 5.Implementación 6.Prueba 7.Desarrollo 8.Configuración y manejo de cambios	OO	2.-5.Diagramas UML	UML	*Rational Rose*

En la Tabla 2 se presenta un segundo estudio comparativo de la misma fuente [6], que relaciona los conceptos de diseño de los tres niveles típicos de diseño *web*: conceptual, estructural y visible. La mayoría de estos métodos realizan una clara separación entre el análisis del dominio, la especificación de la estructura navegacional y el diseño de la interfaz de usuario.

Tabla 2. Comparación de conceptos de diseño de las metodologías de desarrollo *web*

	HDM	**RMM**	**EORM**	**OOHDM**	**SOHDM**	**WSDM**	**WAE**

[12] *Entity – Relationship,* Entidad - Relación
[13] Unidades de presentación que aparecen como páginas de una aplicación hipermedia [6]
[14] *User Interface*, Interfaz de Usuario
[15] *Relationship Management Data Model,* Modelo de Datos de Administración de Relaciones
[16] *Object Oriented*, Orientando a Objetos
[17] *Graphical User Interface*, Interfaz de Usuario Gráfica
[18] *Object Modeling Technique*, Técnica de Modelado de Objetos
[19] *Unified Modeling Language*, Lenguaje de Modelado Unificado

Nivel Conceptual	Entidad Colección Perspectiva Relaciones	entidad relación	clases relación-OO -generalizada -definida por el usuario	clases perspectiva relación-OO	escenarios: -evento -actividad fujo de actividad	ojeto perspectiva relación	*case* relación-OO
Nivel Estructural	Enlace: -estructural -aplicación -perspectiva componente nodo Estructuras de acceso: -enlace colección -enlace índice -visita guiada	enlace: -unidireccional -bidireccional *Slices* primitivas de acceso: -agrupar (menú) -índice -visita guiada -visita guiada indexada	enlace: -simple -navegacional -nodo a nodo -tramo a nodo -estructural -conjunto -lista	enlace clase navegacional contexto navegacional estructuras de acceso: -índice -visita guiada	enlace navegacional vista-OO: -base -asociación -colaboración ASN[20]: -agrupar -índice -visita guiada	enlace componente -navegación -información -externo camino navegacional	enlace enlace dirigido redirigir construir enviar página *web* -página del cliente -página del servidor
Nivel Visible	Ranura Marco	*Slices*		ADV en contexto	componente UI: -elección -texto de entrada de búsqueda -botón -imagen -barra de desplazamiento - ancla HTML[21] -otros		conjunto de marcos formulario objetivo elemento de selección elemento de entrada elemento de área de texto

5 Revisión de tecnologías para el desarrollo de aplicaciones web

[20] *Acces Structure Node,* Nodo de Estructura de Acceso
[21] *HyperText Markup Language*, Lenguaje de Marcado Hiper-Texto

Cada capa de diseño OOHDM constituye un sector de la aplicación con objetivos específicos; las tecnologías aplicadas en cada uno de ellos deben ser capaces de satisfacer sus requerimientos y de acoplarse fácilmente a las tecnologías asociadas a las capas restantes.

En esta sección se presentarán las tecnologías básicas para llevar a cabo la implementación de aplicaciones construidas con OOHDM. Se optó por tecnologías orientadas a objetos, no sólo para facilitar el traslado (explicado en detalle en la Sección 5), sino para aprovechar al máximo la expresividad lograda en los distintos diseños.

5.1 Lenguaje Java

Java es un lenguaje de programación orientado a objetos desarrollado por la compañía Sun Microsystems [13]. Está construido a partir de lenguajes orientados a objetos anteriores, como C++, pero no pretende ser compatible con ellos sino ir mucho más lejos, añadiendo nuevas características como recolección de basura, programación multihilos y manejo de memoria a cargo del lenguaje.

Java fue diseñado para que la ejecución de código a través de la red fuera segura, para lo cual fue necesario deshacerse de herramientas de C tales como los punteros. También se han eliminado aspectos que demostraron ser mejores en la teoría que en la práctica, tales como sobrecarga de operadores, que por cierto todavía está en discusión, y herencia múltiple.

La portabilidad fue otra de las claves para el desarrollo de *Java*, para lograr que las aplicaciones se escriban una sola vez sin la necesidad de modificarlas para que corran en diferentes plataformas. Esta independencia se alcanza tanto a nivel de código fuente (similar a C++) como a nivel de código binario. La solución adoptada fue compilar el código fuente para generar un código intermedio (*bytecodes*) igual para cualquier plataforma. La JVM (Máquina Virtual de *Java*[22]), donde reside el intérprete *Java*, sólo tiene que interpretarlos.

5.2 Java DataBase Connectivity

JDBC (Conectividad de Base de Datos) es una interfaz que provee comunicación con bases de datos. Consiste de un conjunto de clases e interfaces escritas en *Java* [14], que proveen una API (Interfaz de Programación de Aplicación[23]) estándar para desarrolladores de herramientas de base de datos, permitiendo independizar la aplicación de la base de datos que utiliza.

La API JDBC es la interfaz natural a las abstracciones y conceptos básicos de SQL (Lenguaje de Consultas Simple[24]): permite crear conexiones, ejecutar sentencias SQL y manipular los resultados obtenidos. Conceptualmente es similar a ODBC (Conectividad de Base de Datos Abierta[25]), pero ésta no es apropiada para usar directamente desde *Java* porque usa una interfaz en C y una traducción literal de C a *Java* no es deseable.

JDBC soporta dos modelos de acceso a base de datos: modelo de dos capas[26] y modelo de tres capas[27][14]. En el primer caso, la aplicación *Java* se comunica directamente con la base de datos mediante un controlador JDBC específico para cada DBMS (Sistema de Administración de Base de Datos[28]) que se desee manipular.

---

[22] *Java Virtual Machine*
[23] *Application Programming Interface*
[24] *Simple Query Language*
[25] *Open Database Connectivity*
[26] *two - tier*
[27] *three - tier*
[28] *DataBase Management System*

En el segundo caso, los comandos son enviados a un capa intermedia[29] de servicios, encargado de reenviar las sentencias SQL a la base de datos.

Existe un controlador, llamado puente JDBC-ODBC, que implementa las operaciones de JDBC traduciéndolas en operaciones ODBC, con lo cual se provee acceso a cualquier base de datos cuyo controlador ODBC se encuentre disponible.

5.3 Servlets

Un *servlet* es una clase *Java* [15], embebida dentro del *web server,* y utilizada para extender la capacidad del servidor. La API de *servlets* provee clases e interfaces para responder a cualquier tipo de requerimientos; en particular, para las aplicaciones que corren en servidores *web*, la API define clases de *servlet* específicas para requerimientos HTTP.

No necesitan ser ejecutados como nuevos procesos dado que corren directamente en el *web server*. Viven entre sesiones y se puede decir que son persistentes: no es necesario crear un *servlet* por cada requerimiento realizado desde el cliente, sino que corren dentro de éste múltiples hilos.

Los *servlets* [16] son programas *Java* que proveen la funcionalidad de generar dinámicamente contenidos *Web*. Pueden ser ejecutados a través de una línea de comando. A diferencia de los *applets*, no poseen restricciones en cuanto a seguridad. Tienen las propiedades de cualquier aplicación *Java* y pueden acceder a los archivos del servidor para escribir y leer, cargar clases, cambiar propiedades del sistema, etc. Del mismo modo que las aplicaciones de programas *Java*, los *servlets* están restringidos por los permisos del sistema.

Son cargados la primera vez que son usados, y permanecen en memoria para satisfacer futuros requerimientos. Tiene un método *init*, donde el programador puede inicializar el estado del *servlet*, y un método *destroy* para administrar los recursos que son mantenidos por el *servlet*.

5.4 Java Server Pages

JSP (Páginas de Servidor *Java*[30]) provee a los desarrolladores de *web* de un entorno de desarrollo para crear contenidos dinámicos en el servidor usando plantillas[31] HTML y XML (Lenguaje de Marcado Extensible[32]), en código *Java*, encapsulando la lógica que genera el contenido de las páginas [17].

Cuando se ejecuta una página JSP es traducida a una clase de *Java*, la cual es compilada para obtener un *servlet*. Esta fase de traducción y compilación ocurre solamente cuando el archivo JSP es llamado la primera vez, o después de que ocurran cambios.

JSP y XML tienen un interesante relación, descrito en [18]. Así como pueden generarse páginas HTML dinámicas a partir de una fuente en JSP, pueden generarse dinámicamente en forma análoga documentos XML. Más adelante, en la sección de implementación de este artículo, podrá observarse un ejemplo concreto de generación de contenido XML a partir de páginas JSP.

5.5 eXtensible Markup Language

---

[29] *middle tier*
[30] *Java Server Pages*
[31] *templates*
[32] *Extensible Markup Language*

La familia XML es un conjunto de especificaciones que conforman el estándar que define las características de un mecanismo independiente de plataformas desarrollado para compartir datos. Se puede considerar a XML como un formato de transferencia de datos multi-plataforma. Es un subconjunto de SGML (Lenguaje de Marcado Generalizado Standard[33]) y uno de sus objetivos es permitir que SGML genérico pueda ser servido, recibido y procesado en la *web* de la misma manera que actualmente es posible con HTML.

XML ha sido diseñado de tal manera que sea fácil de implementar. No ha nacido sólo para su aplicación en Internet, sino que se propone como lenguaje de bajo nivel (a nivel de aplicación, no de programación) para intercambio de información estructurada entre diferentes plataformas.

XML hace uso de etiquetas (únicamente para delimitar datos) y atributos, y deja la interpretación de los datos a la aplicación que los utiliza. Por esta razón se van formando lenguajes a partir del XML, y desde este punto de vista XML es un metalenguaje.

El conjunto de reglas o convenciones que impone la especificación XML permite diseñar formatos de texto para los datos estructurados, haciendo que se almacenen de manera no ambigua, independiente de la plataforma y que en el momento de la recuperación se pueda verificar si la estructura es la correcta.

Para comprobar que los documentos estén bien formados se utiliza un DTD (Definición de Tipo de Documento[34]). Se trata de una definición de los elementos que pueden incluirse en el documento XML, la relación entre ellos, sus atributos, posibles valores, etc. Es una definición de la gramática del documento, es decir, cuando se procesa cualquier información formateada mediante XML, el primer paso es comprobar si está bien formada, y luego, si incluye o referencia a un DTD, comprobar que sigue sus reglas gramaticales.

5.6 eXtensible Stylesheet Language

XSL (Lenguaje de Hojas de Estilo Extensible) [20] es una especificación desarrollada dentro del W3C (*World Wide Web Consortium*) para aplicar formato a los documentos XML de forma estandarizada.

Aunque se ha establecido un modo para que puedan usarse hojas de estilo CSS (Hojas de Estilo en Cascada[35]) dentro de documentos XML, es lógico pensar que para aprovechar las características del nuevo lenguaje hace falta tener un estándar paralelo y similar asociado a él.

Según el W3C, XSL es "un lenguaje para transformar documentos XML", así como un vocabulario XML para especificar semántica de formateo de documentos.

Además del aspecto que ya incluía CSS referente a la presentación y estilo de los elementos del documento, añade una pequeña sintaxis de lenguaje de comandos para poder procesar los documentos XML de forma más cómoda. La XSL permite añadir lógica de procesamiento a la hoja de estilo.

La idea es asociar al documento XML con una hoja de estilo y a partir de esto visualizar el documento XML en cualquier plataforma: *PalmPC*, *PC*, *Internet Explorer*, *Netscape*, etc. y con el aspecto (colores, fuentes, etc) que se quiera utilizar.

En esencia, XSL son dos lenguajes: uno de transformación y otro de formateo. El lenguaje de transformación permite transformar un documento XML en otro con diferente formato, como HTML o texto plano, o bien en otro documento XML. El lenguaje de formateo no es más que un vocabulario XML para especificar objetos de formateo (FO[36]).

---

[33] *Standard Generalized Markup Language*
[34] *Document Type Definition*
[35] *Cascade Style Sheets*
[36] Formatting Objects

Al igual que con HTML, se pueden especificar las hojas de estilo, CSS o XSL, dentro del propio documento XML o haciendo referencia a ellas en forma externa. Esto es muy útil para mover datos de una representación XML a otra representación, basada en correo electrónico, intercambio de datos electrónicos, intercambio de metadatos, y alguna aplicación que necesite convertir datos entre diferentes representaciones de XML a otro tipo de representación.

La gran ventaja de utilizar XML y XSL es que los datos y la presentación de estos quedan en dos archivos diferentes.

Existen tres opciones para transformar un documento XML a otro formato, como se describen en [21], tal como HTML, utilizando hojas de estilo XSL:

- en el cliente: los documentos XML y las hojas de estilo son enviados al cliente (*Web* Browser), el cual se encarga de transformar los documentos basándose en la especificación de las hojas de estilo, y luego de la transformación se presentan al usuario en el explorador.
- en el servidor: el servidor es el encargado de aplicar el XSL al documento XML para transformarlo en algún otro formato (generalmente HTML) y envía el documento transformado al cliente.
- ni en el servidor, ni en el cliente: una tercera opción consiste en transformar el documento original XML en algún otro formato (usualmente HTML) antes de que el documento sea ubicado en el servidor.

6 Trasladando OOHDM a una implementación

En esta sección se describirán las etapas de desarrollo de una aplicación *web* simple, siguiendo la metodología OOHDM. En cada capa de diseño, la implementación correspondiente se basa en diferentes tecnologías, elegidas con el propósito de minimizar la dificultad del desarrollo y aprovechar al máximo las virtudes de la metodología.

6.1 Capa Conceptual

En OOHDM, el desarrollo se inicia diseñando la capa conceptual, siendo el principal objetivo de esta etapa capturar los conceptos involucrados en el dominio de la aplicación y describirlos en detalle, haciendo uso de diagramas que permitan expresar con claridad el comportamiento, la estructura y las relaciones entre dichos conceptos. La Programación Orientada a Objetos facilita el traslado del diseño conceptual a la implementación, proveyendo al programador con herramientas que permiten reducir la distancia entre el problema del mundo real y la programación de la solución en la computadora.

El modelo de objetos del ejemplo a discutir consta de las entidades básicas de un dominio específico: un comercio de venta de productos B2C (Negocio a Consumidor[37]). En este dominio, entidades como *producto*, *categoría de productos*, *carro de compras*, *usuario*, *venta*, se interrelacionan para responder a la navegación del usuario por la aplicación y a sus actividades transaccionales. Todas las entidades mencionadas se construyen a partir de información persistente, propiedad mantenida por la empresa en forma directa a través de un DBA (Administrador de Bases de Datos[38]) o simplemente aprovechando una funcionalidad incorporada de la aplicación que permita manipular la base de datos. Esta última alternativa es un ejemplo, entre otros fundamentos, de la importancia del diseño conceptual en el desarrollo de aplicaciones: el mismo esquema de objetos y relaciones que en principio pudieron ser diseñadas únicamente para mostrar información navegacional, puede ser abastecida con una interfaz que lleve a cabo la interacción con la base de datos, concentrando así esta actividad en un único sector de la aplicación, sin afectar al resto del modelo.

[37] *Bussiness to Consumer*
[38] *DataBase Administrator*

En cada diseño conceptual existe comportamiento que escapa a la simple navegación de información. Se trata del comportamiento inherente de cada clase, y aunque la aplicación particular no requiera que se implemente, es importante destacar que si la aplicación crece el diseño conceptual debe estar preparado para ser extendido, tal como cualquier diseño orientado a objetos. Más adelante podrá observarse cómo un modelo robusto y eficiente puede facilitar el trabajo de las capas restantes de la aplicación.

Retomando el diseño conceptual del ejemplo, el análisis anterior de las entidades del dominio permite afirmar que dichas clases comparten (al menos) el comportamiento correspondiente a la interfaz con la capa de persistencia: todas las entidades fuertes son capaces de construirse a partir de identificadores, coincidentes con las claves primarias de las tablas correspondientes de la base de datos [23]. Las clases del diseño conceptual que representen a estas entidades podrán obtener sus atributos al iniciarse y actualizar los cambios cuando sea necesario (realizando eventualmente algún tipo de almacenamiento temporal para mejorar la eficiencia). La consistencia de la información queda asegurada, entonces, por el comportamiento de los objetos del modelo.

El lenguaje elegido para desarrollar la implementación de esta capa de OOHDM es *Java*, presentado en la sección de tecnologías de este artículo. Durante esta etapa se utilizará también JDBC como paquete de vital importancia para el manejo de base de datos.

En la Figura 1 pueden observarse las primeras capas de implementación descritas anteriormente.

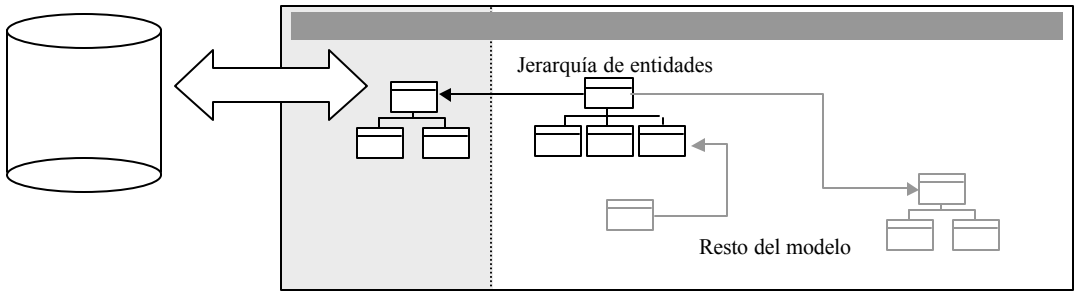

Figura 1: Paquete de interfaz con la base de datos, dentro del Diseño Conceptual

La clase abstracta que define el comportamiento básico de las entidades del modelo y concentra la lógica de interacción con la base de datos será denominada *EntidadAbstracta*. Para cumplir con los objetivos propuestos, esta clase debe ser capaz de crear una conexión con la base de datos, ejecutar consultas y retornar los resultados para ser procesados. Por una cuestión de eficiencia, la creación de conexiones a la base de datos podría ser delegada a un *singleton* [22], es decir, una clase capaz de controlar su instanciación para retornar siempre la misma instancia en reiteradas llamadas a su constructor. Una clase con estas características podría ser instanciada por *EntidadAbstracta* para luego solicitarle una conexión.
Como se explicó con anterioridad, las subclases de *EntidadAbstracta* son entidades capaces de construirse a partir de una clave, realizando una consulta a la base de datos que involucre una o más tablas y que retorne una tupla unívocamente identificada por dicha clave. Los constructores de las subclases concretas de *EntidadAbstracta* pueden realizar estas consultas, utilizando en forma de *template method* [22] el comportamiento heredado. En la Figura 2 puede observarse con mayor claridad la secuencia de pasos involucrados en la instanciación de una entidad concreta; en este caso se instanciará la clase *Producto* para ejemplificar el proceso.

Sólo la información más importante y de menor volumen es cargada desde la base de datos en el momento de la instanciación. La información restante puede ser cargada bajo demanda, a partir de un eventual requerimiento de la aplicación. Para ilustrar esta idea con claridad puede considerarse cargar los siguientes atributos en la instanciación de un *Producto*: descripción, categoría, cantidad disponible y precio. En algún momento de la ejecución, la aplicación puede requerir los productos relacionados de un determinado producto

(por ejemplo, el usuario podría solicitar una lista de productos relacionados con el producto televisor, tales como video grabadora, filmadora, mesa para televisor, etc.); dado el volumen de esta información y lo esporádico de su requerimiento, se sugiere entonces consultar a la base de datos para obtener esta información sólo cuando es requerida. Notar que la conexión a la base usada en cada consulta es siempre la solicitada en el momento de la instanciación, evitando con esto creaciones y destrucciones reiteradas de conexiones a la base de datos.

Un modelo conceptual de estas características, obliga a considerar a todas las entidades del dominio como subclases de EntidadAbstracta (al menos todas aquellas entidades que se mantienen en la base de datos). Esta decisión de diseño puede refinarse aún más para evitar al máximo la incorporación de comportamiento de persistencia en las clases del dominio. Para alcanzar este objetivo puede utilizarse interfaces[39] (mediante las cuales se pueden establecer contratos entre clases, permitiendo así ligar a las clases del dominio con las encargadas de interactuar con la base de datos) y/o un modelo paralelo de *wrappers* [22] que encapsulen a las clases del dominio (estos *wrappers* pueden encargarse directamente de la persistencia o interactuar con otras clases para factorizar código, dejando en cada *wrapper* el comportamiento específico requerido por la entidad encapsulada).

Es importante destacar que el diseño conceptual puede estar compuesto de otras clases que por su naturaleza no puedan considerarse subclases de *EntidadAbstracta*. Estos casos son simplemente colaboradores de entidades concretas, clases requeridas para realizar alguna actividad específica y de corta vida útil, o algunos casos de entidades débiles desde el punto de vista de diseño entidad-relación. Considérese como ejemplo la información relacionada con la navegación del usuario en una determinada sesión, irrelevante para otras sesiones e incluso para el resto de la aplicación. En este caso, se requiere una clase para contener la información mencionada y el comportamiento para manipularla, pero no requiere considerar la persistencia dentro de su funcionalidad.

El resto de las consideraciones a tener en cuenta para implementar el diseño conceptual depende del dominio específico. Hasta el momento se dispone de un modelo cuyas clases más importantes pueden instanciarse con facilidad para crear objetos de la capa navegacional y posteriormente decorarse para ser presentados en alguna interfaz. La generalidad del diseño conceptual es una característica determinante para llevar a cabo este tipo de construcciones: es necesario considerar a las entidades sin acotar su funcionalidad ni estructura por cuestiones de usuarios o contextos navegacionales.

[39] Una interfaz [16] es un conjunto de firmas, es decir, un conjunto de declaraciones de operaciones. Cada operación se declara con un nombre, los parámetros y el valor de retorno.

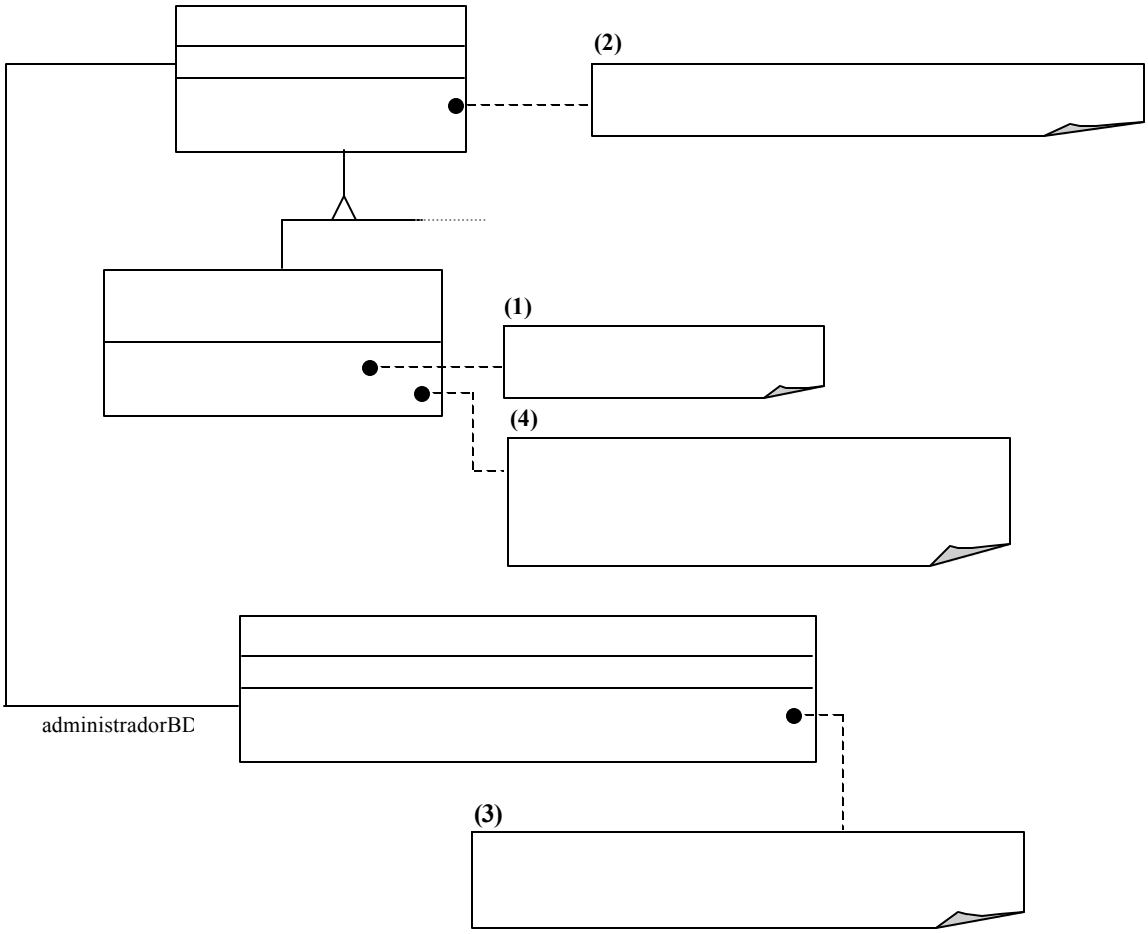

Figura 2: Instanciación de una subclase concreta de EntidadAbstracta

A continuación podrá observarse cómo un nodo de la capa navegacional es construido realizando los requerimientos convenientes a la entidad o entidades del diseño conceptual que observa, generando así una vista de dicha porción del diseño conceptual.

6.2 Capa Navegacional

La capa navegacional se compone de objetos construidos a partir de objetos conceptuales, y constituyen en general los elementos canónicos de las aplicaciones hipermedia tradicionales: *nodos*, *enlaces*, *anclas* y *estructuras de acceso*. Sin embargo, estas clases pueden extender el comportamiento característico para funcionar como *adaptadores* [22] de los objetos conceptuales y delegar así operaciones específicas del dominio.

Entonces, los objetos navegacionales pueden actuar como *observadores* [22], para construir vistas de objetos conceptuales, y como *adaptadores*, para extender la actividad navegacional de un nodo y poder aprovechar el comportamiento conceptual del objeto adaptado. Estas dos perspectivas pueden implementarse aprovechando las virtudes inherentes de diferentes tecnologías: JSP para observar y *Servlets* para adaptar.

Las páginas JSP serán responsables de construir los nodos de la capa navegacional. Esto se logra instanciando los objetos del diseño conceptual necesarios para mostrar la información del nodo y utilizando los datos solicitados a dichas instancias para generar árboles de elementos XML. Con este procedimiento se compone

un nodo concentrando información relacionada en un documento XML generado dinámicamente con cada requerimiento, lo que permite además personalizar el contenido (datos sin presentación) a partir de infinitas configuraciones, tales como un perfil de usuario, la sobrecarga de requerimientos de la aplicación, la historia registrada de la navegación, políticas de protección de contenidos, seguridad, etc.

La Figura 3 ilustra la construcción del nodo a partir de un requerimiento HTTP proveniente de un cliente remoto.

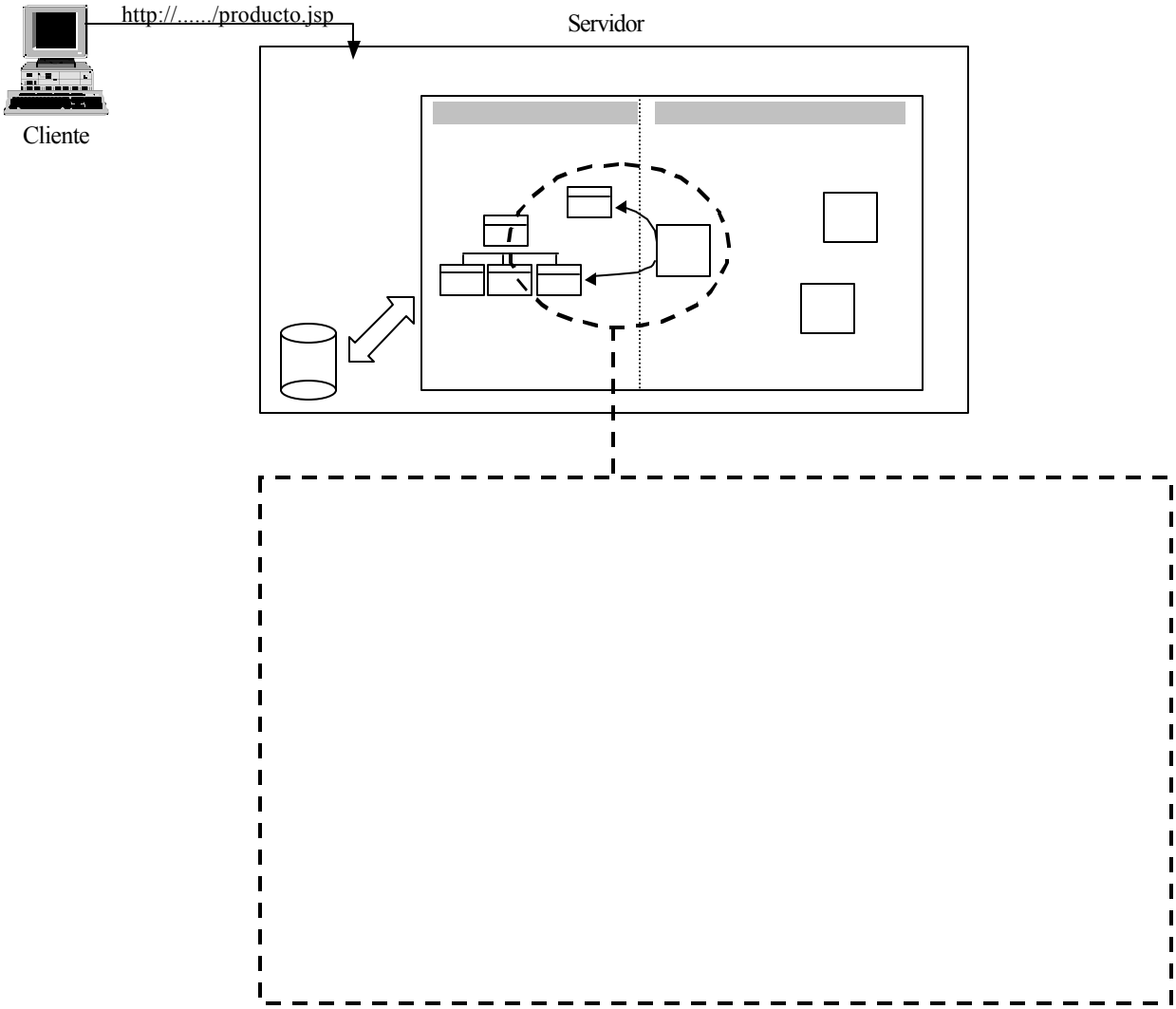

Figura 3: Construcción de un nodo de la capa navegacional

Como puede observarse, la cadena se inicia con un requerimiento del usuario que navega por la aplicación. Al acceder a un enlace, una página JSP es invocada para construir una página XML (cuya presentación será explicada en breve). En principio, el archivo XML generado puede ser útil para transferir información entre servidores cooperativos, o simplemente entre un cliente y un servidor con un esquema tradicional. En cualquier caso, la portabilidad brindada por las características simples de XML hacen posible una transferencia transparente incluso entre plataformas completamente heterogéneas. Más adelante se podrá descubrir la verdadera razón por la cual se elige una salida con formato XML para los nodos de la capa navegacional.

Antes de abordar la implementación de la capa de presentación es necesario describir la segunda perspectiva de los nodos, vistos como *adaptadores* de objetos conceptuales. Un ejemplo donde se observa claramente este tipo de nodos es el *carro de compras*. Para mostrar el contenido del carro de compras del usuario es necesario instanciar el objeto conceptual *CarroDeCompras* (notar que el identificador del usuario que navega la aplicación y solicita acceder a su carro de compras es el único dato que se necesita para invocar al constructor de dicha entidad). De este modo, el nodo construido a partir de esta información no sólo concentra los datos de las compras sino que además ofrece un menú de operaciones para manipular el carro de compras, como borrar un elemento, cambiar la cantidad solicitada de un producto, vaciar el carro, finalizar la compra. Todas estas operaciones no son responsabilidad del nodo, sino que son realizadas por el objeto conceptual. Entonces, delegar responsabilidad es lo único que debe hacer el nodo navegacional en este caso: la actividad delegada continúa dentro de un *servlet* y finalmente es el *servlet* quien se encarga de redireccionar la navegación a una página JSP para eventualmente mostrar un resultado.

Los *servlets* son una herramienta muy útil para realizar actividades del lado del servidor y retornar información al cliente para informarle sobre el trabajo realizado o para solicitarle parámetros y retomar alguna operación. Continuando con el ejemplo del carro de compras, en caso que la operación elegida por el usuario sea vaciar el contenido (entre otras operaciones que puede seleccionar), una transacción debe ejecutarse a cargo del objeto *CarroDeCompras* correspondiente, a fin de modificar la base de datos satisfaciendo el deseo del usuario. Dado que este deseo es manifestado a través de un requerimiento (por ejemplo, realizando un *get* a través de una URL, o un *post* a través de un formulario HTML) la acción debe continuar en algún sector de la aplicación, sin necesidad de mostrar información durante el proceso, al menos en este caso puntual.

Entonces, puede construirse un paquete de *servlets* capaces de realizar operaciones específicas e incluso agruparse en jerarquías para aprovechar la herencia de comportamiento. A manera de ejemplo, considérese la jerarquía de *servlets* de la Figura 4, diseñados para servir acciones específicas referidas al *CarroDeCompras*.

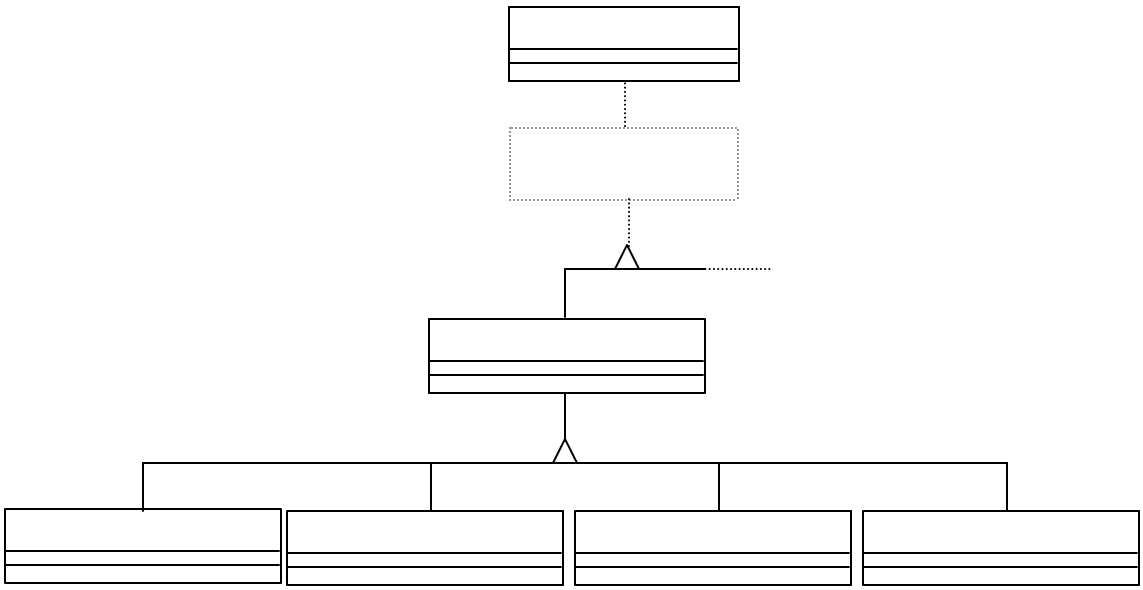

Figura 4: Jerarquía de *servlets* para administrar el carro de compras

El *servlet abstracto de la aplicación* es una clase que podría ser útil si se tiene comportamiento común a todos los *servlets* construidos para una aplicación específica. Luego, el *servlet* abstracto *CarroDeComprasServlet* define el comportamiento y la estructura de los *servlets* que encapsulan la lógica de una operación determinada sobre el carro de compras. No es necesario construir una subclase por operación concreta; varias operaciones pueden encapsularse en una única clase y parametrizarse adecuadamente si son semejantes.

En este escenario, cada subclase concreta de *servlet* tiene la responsabilidad de ejecutar una operación y retornar al cliente mostrando una página fija o calculada dinámicamente en función del resultado de la operación. Un ejemplo que puede considerarse para cualquier actividad es alternativa entre un resultado exitoso o uno erróneo, de la operación ejecutada. En el primer caso, la navegación podría continuar por una página JSP capaz de ilustrar la forma de proseguir la actividad, o de mostrar los resultados de la misma. En el segundo caso, la navegación se vería interrumpida por una página JSP que informara sobre el error producido y eventualmente solicitara la corrección de parámetros, u ofreciera reintentar la operación.

Por razones de espacio, la implementación de contextos no será tratada y se continúa directamente con la capa de presentación.

6.3 Capa de Interfaz Abstracta

Tanto un nodo actuando como observador como un nodo actuando como adaptador, finalmente continúa por mostrar cierta información y para ello necesita definir la forma de presentación mediante la cual dicha información será visualizada en la interfaz. Para ello se incorporan las dos últimas tecnologías a las que se hará mención en este artículo: XSL y un mecanismo de análisis sintáctico para obtener una página HTML en función de un par de documentos XML/XSL.

Las páginas XSL, ubicadas también del lado del servidor, definirán la apariencia de los nodos que se generaron en formato XML. Cada página XSL define la forma en que los elementos del XML asociados serán mostrados, haciendo uso de código HTML y eventualmente CSS [24], para dar el formato deseado a las páginas finales. Enlaces y estructuras de acceso también son presentados conforme con los estilos adoptados para este tipo de información navegacional.

El vínculo entre un XML y su apariencia puede establecerse en forma explícita, con una etiqueta especial situada en la primera línea del archivo XML, o bien podría ser calculado en forma dinámica para satisfacer demandas de personalización. En cualquier caso, retornar al cliente con un archivo XML que haga referencia a un archivo XSL ubicado en el servidor tiene al menos dos inconvenientes: genera un tráfico de ida y vuelta innecesario, ya que podría evitarse enviando directamente el código HTML al cliente, y puede requerir características especiales en el explorador del cliente que pueden privarlo de visualizar la página correctamente.

La transformación del lado del servidor es provista en la actualidad por paquetes como *Xalan-Java* [25] y *Saxon* [26]. Un esquema de la operación de transformación puede observarse en la Figura 5.

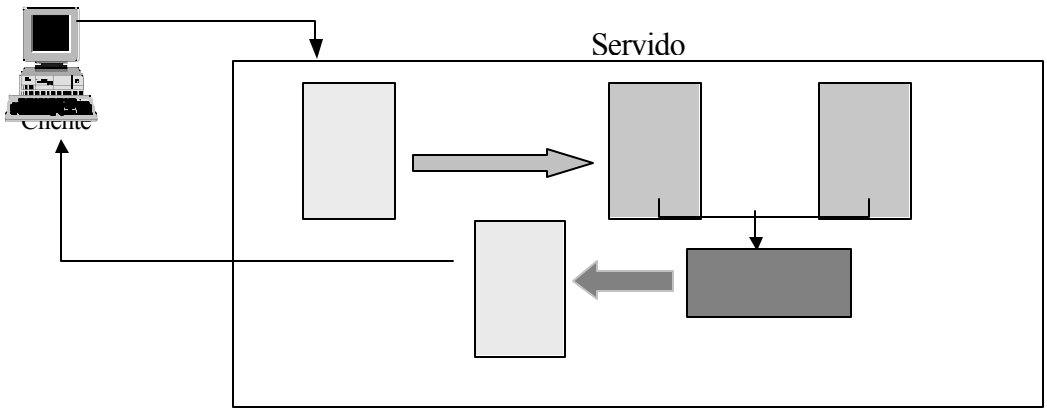

Figura 5: Generación de un documento HTML a partir de una fuente XML + XSL

El lenguaje XSLT (Transformaciones XSL[40]) [27] se utiliza para componer hojas de estilo XSL. Estos documentos contienen instrucciones que mediante el uso de *Xalan-Java*, por ejemplo, sirven para llevar a cabo las transformaciones y producir un documento de salida, una secuencia de caracteres o de *bytes*, un DOM (Modelo de Objetos de Documento[41]), etc. En el caso particular mencionado anteriormente, se desea obtener un documento de salida con formato HTML a partir de un par de documentos XML / XSL.

Aquí concluye la planificación del desarrollo en capas. Finalmente, las tres capas de diseño han sido implementadas por separado. Se ha intentado mantener la independencia de los elementos del dominio con las construcciones navegacionales y de presentación, haciendo uso de las tecnologías adecuadas para trabajar con OOHDM, y aprovechar al máximo las virtudes de una aplicación de objetos.

7 Conclusiones

En este artículo se presenta un enfoque para implementar aplicaciones *Web* usando OOHDM como técnica de diseño. Dicha metodología propone dedicar un tiempo importante en las fases previas a la implementación. Esta inversión de tiempo está ampliamente justificada no sólo porque simplifica el proceso de desarrollo, facilitando el trabajo del equipo encargado de cada capa de la aplicación, sino también durante su mantenimiento y eventual extensión. Son quizás estas últimas tareas las más difíciles de lograr con tecnologías tradicionales, y aún imposibles en muchos casos donde no existe diseño detallado y la implementación concentra conceptos heterogéneos muy difíciles de modificar.

OOHDM propone un conjunto de tareas que en principio pueden involucrar mayores costos de diseño, pero que a mediano y largo plazo reducen notablemente los tiempos de desarrollo al tener como objetivo principal la reusabilidad de diseño, y así simplificar la evolución y el mantenimiento. El presente artículo complementa la presentación teórica de la metodología con una idea concreta de implementación, usando tecnologías potentes y de alto crecimiento.

Referencias

[1] W. De Muynck. Bridging the Gap between XML and Hypermedia: a Layered Transformational Approach, Tesis. Approach, Vrije Universiteit Brussel, Belgium, 2000.

[2] D. Schwave and G. Rossi. An Object Oriented Approach to Web-Based Application Desing. En: Theory and Practice of Object Systems (TAPOS), October 1998.

[40] *XSL Transformations*
[41] *Document Object Model*

and Practice of Object Systems (TAPOS), October 1998.

[3] D. Schwave *et al*. Engineering Web Applications for Reuse. IEEE Multimedia, Vol 8 Nro 1, pp 20-31.

[4] G. Rossi; D. Schwave and Fernando Lyardet. Web application models are more than conceptual models. En: Proceedings of the First Internation Workshop on Conceptual Modeling and the WWW, Paris, France, November 1999.

[5] D. Cowan and C. Lucena. Abstract Data Views: An Interface Specification Concept to Enhance Design for Reuse. IEEE Transactions on Software Engineering. Vol. 21, No. 3, March 1995.

[6] N. Koch. Comparing Development Methods for Web Applications. Ludwig-Maximilians-University Munich, Institute of Computer Science Oettingenstr. 67, 80538 München, Germany. 2000.

[7] F. Garzotto; L. Mainetti and P. Paolini. Hypermedia design analysis. Communications of the ACM, 8(38), 74-86. 1995.

[8] T. Isakowitz; E. Stohr and P. Balasubramanian. A methodology for the design of structured hypermedia applications. Communications of the ACM, 8(38), 34-44. 1995.

[9] D. Lange. An object-oriented design approach for developing hypermedia information systems. Journal of Organizational Computing and Electronic Commerce, 6(3),269-293. 1996.

[10] H. Lee; C. Lee and C. Yoo. A scenario-based object-oriented methodology for developing hypermedia information systems. En: Proceedings of 31st Annual Conference on Systems Science, Eds. Sprague R. 1998.

[11] O. De Troyer and C. Leune. WSDM: A user-centered design method for Web sites. En: Proceedings of the 7th International World Wide Web Conference. 1997.

[12] J. Conallen. Building Web application with UML. Addison Wesley. 1999.

[13] Java Sun, The Java Tutorial: A practical guide for programmers, http://Java.sun.com/docs/books/tutorial, Marzo de 2001.

[14] Java Sun, JDBC API tutorial and reference - Second Edition, http://Java.sun.com/products/jdbc/, Marzo de 2001.

[15] Java Sun, Java™ Servlet Technology: The Power Behind the Server, http://Java.sun.com/products/servlet, Marzo de 2001.

[16] Eva.Arderiu, Javier.Conde. Objectivity/DB and JAVA, http://wwwinfo.cern.ch/asd/cernlib/rd45/objy_Java_info.htm#Servlets, Marzo de 2001.

[17] K. Avedal *et al*. Professional JSP, Wrox Press 2000.

[18] Java Sun, Java Server Pages™: Dinamically Generated Web Content, http://www.Javasoft.com/products/jsp/, Marzo de 2001.

[19] World Wide Web Consortium (W3C), Extensible Markup Language (XML). The base specifications are XML 1.0, W3C Recommendation Feb '98, and Namespaces, Jan '99. http://www.w3.org/XML,

Marzo de 2001.

[20] World Wide Web Consortium (W3C), Extensible Stylesheet Language (XSL), Version 1.0.W3C Candidate Recommendation 21 November 2000. http://www.w3.org/TR/xsl, Marzo de 2001.

[21] R. Anderson et at.. Professional XML, Wrox Press 2000.

[22] E. Gamma *et al*. Design Patterns: Elements of reusable object-oriented software, Addison-Wesley, 1995.

[23] H. F. Korth y Abraham Silberschatz. Fundamentos de Base de Datos. Segunda Edición en español, McGraw-Hill, 1993.

[24] CCS Test Suite, W3C Core Styles. http://www.w3.org/Style/CSS/, Marzo de 2001.

[25] The Apache Software Foundation, Xalan – Java Version 1.2.2. http://xml.apache.org/xalan/, Marzo de 2001.

[26] Michael H. Kay, About Saxon, http://users.iclway.co.uk/mhkay/saxon/instant.html, Marzo de 2001.

[27] World Wide Web Consortium (W3C), Extensible Stylesheet Language Transformations (XSLT), Version 1.0. W3C Recommendation 16 November 1999. http://www.w3.org/TR/xslt, Marzo de 2001.

Nombre de archivo: Articulo Dario Silva-Barbara Mercera.doc
Directorio: A:
Plantilla: C:\WINDOWS\Application Data\Microsoft\Plantillas\Normal.dot
Título: Construyendo aplicaciones web con una metodología de diseño orientada a objetos
Asunto:
Autor: Silva - Mercerat
Palabras clave: Aplicaciones web, tecnologías de desarrollo, programación orientada a objetos, diseño en capas, contenido dinámico, patrones de diseño.
Comentarios:
Fecha de creación: 29/01/02 12:12 P.M.
Cambio número: 2
Guardado el: 29/01/02 12:12 P.M.
Guardado por: UNAB
Tiempo de edición: 0 minutos
Impreso el: 01/02/02 09:13 A.M.
Última impresión completa
Número de páginas: 20
Número de palabras: 7,640 (aprox.)
Número de caracteres: 43,553 (aprox.)