El patrón multi-visualización para la generación de distintas presentaciones en un sistema de comercio electrónico

José R. Gulias, Victor M. Gulias, Alberto Valderruten, Carlos Abalde

MADS Group - LFCIA
Departamento de Computación, Universidad de La Coruña
Campus de Elviña, 15071 La Coruña, España
{gulias,valderruten,carlos}@lfcia.org

Resumen

El sistema de *comercio electrónico* SCED dispone de una gran variedad de opciones y facilidades, orientadas todas ellas a favorecer las compras de los usuarios. Al interactuar con SCED se deberían tener en cuenta las características y capacidades de cada dispositivo de cara a sacarle un rendimiento óptimo a cada uno de ellos. En este trabajo se comenta la solución empleada, presentada bajo la forma de patrón de diseño: El Patrón Multi-Visualización. Este se utiliza para estructurar todo el proceso de generación de las presentaciones para el usuario.

1 Introducción

Los patrones de diseño ([GHJV95], [Dou03]) son soluciones a problemas comunes, que han sido utilizadas con éxito en el pasado y que son suficientemente generales para ser reutilizadas ante otros problemas similares.

Por lo tanto, el uso de patrones permite que los desarrolladores de aplicaciones no tengan que plantearse nuevamente soluciones a problemas comunes cada vez que se encuentren ante la construcción de un nuevo sistema. La solución más adecuada ante la aparición de un problema conocido es aplicar un patrón que se sabe que es útil para la resolución de ese problema y adaptarlo a sus características particulares.

Todo ello implica una reducción en el esfuerzo y en el coste del desarrollo de un sistema, y una mayor claridad y reusabilidad del código generado, características todas ellas inherentes al uso de patrones en el desarrollo de *software*.

Y, dentro del terreno del desarrollo de aplicaciones, hay que destacar la importancia que han ido adquiriendo a lo largo de los últimos años un conjunto de sistemas agrupados bajo la categoría de aplicaciones de *comercio electrónico* ([Men98], [TS98]).

Actualmente se acostumbra a identificar el concepto de *comercio electrónico* con el comercio a través de Internet (*Internet Commerce*), aunque hay que subrayar que esta es sólo una de las modalidades posibles de *comercio electrónico*. En este artículo, nos centraremos en este término como intercambio de dinero a través de redes abiertas como Internet a cambio de bienes, información o servicios del algún tipo.

Por lo tanto, las aplicaciones de *comercio electrónico* serán aquellas que permitirán a los usuarios acceder desde sus propias casas a la información de todos aquellos productos que tiene a la venta una tienda y realizar sus compras sin necesidad de desplazamientos.

La importancia de estas aplicaciones y sus grandes expectativas de futuro, nos llevan a tener en consideración la relevancia que tiene facilitar a los posibles usuarios las compras en cuanto sea posible, y esto incluye el método de acceso que estén usando.

El desarrollo tecnológico ha desembocado en la aparición en el mercado de un gran número de dispositivos que se adaptan a las preferencias y limitaciones de los usuarios. Cada uno de estos dispositivos cuenta con unas características propias como, por ejemplo, el tamaño o el lenguaje que entiende, que hacen que una misma página no pueda ser utilizada en todos estos dispositivos.

Por lo tanto, parece necesario, dada la importancia que se espera que tengan este tipo de aplicaciones, construir sistemas en los cuales se ofrezca una visualización que se adapte a las particularidades de los distintos dispositivos y faciliten al máximo la interacción de los potenciales clientes.

2 Sistema de Comercio Electrónico SCED

SCED (Sistema de Comercio Electrónico Distribuido) ([Fer01]) es una aplicación cubre las funcionalidades básicas de venta y gestión de un conjunto de tiendas (*mall*). El sistema se centra en el escenario B2C (*Business to Consumer*) proporcionando las funciones requeridas por los distintos actores que interactúan con el sistema, tanto usuarios, potenciales clientes de la aplicación, como administradores (de tienda o de *mall*).

Además, este sistema cuenta con una serie de peculiaridades en lo que respecta a su arquitectura e implementación. Para la implementación de este prototipo se utilizó el lenguaje funcional distribuido Erlang ([ERL01], [AWWV95]), y se manejaron aplicaciones y patrones específicos de este lenguaje de programación.

Una de las características destacables que ofrece el prototipo implementado es la personalización del contenido de las páginas en función del perfil del usuario y del navegador que esté usando para acceder al sistema. Esto se consiguió mediante la puesta en marcha de dos subsistemas independientes: uno encargado de generar un documento XML ([XML01],[PM99]) a partir

de la información del usuario y del entorno, y otro responsable de seleccionar y aplicar una hoja de estilos XSL para obtener la visualización en el formato deseado en función del navegador usado.

SCED está diseñado para funcionar de manera distribuida sobre los nodos de un *cluster*, y cumple otra serie de características interesantes como la escalabilidad, tolerancia a fallos, concurrencia, ...

2.1 Actores del sistema

Se estableció una jeraquía de actores durante el diseño del sistema, pensando en los distintos roles que podrían interactuar con la aplicación. Cada uno de ellos, podrá realizar una serie de acciones determinadas sobre el sistema, a las que se añaden otras más específicas a medida que descendemos por la jerarquía.

1. Usuario: Este actor se corresponde con cualquier potencial visitante del *mall* de tiendas.

2. Usuario anónimo: Este actor representa a aquellos usuarios que no reconoce el sistema, bien porque sea su primera visita o bien porque aún no se haya registrado en el sistema y este sea incapaz de identificarlo correctamente.

3. Usuario registrado: Este actor representa a aquellos usuarios que están registrados en el sistema y, por lo tanto, se dispone de sus datos y su historia previa, pudiendo ofrecer unos contenidos personalizados en función de su perfil.

4. Administrador de tienda: Este actor se encarga de gestionar una tienda individual del *mall*.

5. Administrador de *mall*: Este actor es el principal responsable de la gestión del conjunto de tiendas que componen el *mall*.

2.2 Opciones

2.2.1 Opciones del actor Usuario

1. Visitar tienda: Un usuario que visita el *mall* tiene la posibilidad de navegar libremente por cualquiera de las tiendas que lo componen. Así, puede encontrarse en las páginas de una tienda y decidir cambiar a otra, pudiendo incluso realizar una navegación jerárquica a través de la estructura de tiendas.

2. Buscar productos: Un usuario pretende obtener una lista de productos que cumplan una serie de criterios establecidos en la búsqueda que se realiza. Las búsquedas que puede realizar un usuario pueden ser de

diferentes tipos: búsquedas a partir de una serie de palabras claves, búsquedas por una categoría determinada, ...

3. Consultar producto: El usuario quiere obtener información detallada acerca de un producto concreto. Se deben proporcionar al usuario todos los datos del producto, desde los datos propios del producto (nombre, descripción, precio, ...) hasta el conjunto de recursos asociados (imágenes, vídeos, ...), los productos relacionados, aquellos que han comprado otros usuarios junto con este, las críticas que se han realizado sobre el producto...

4. Buscar usuarios: Un usuario puede buscar a una persona determinada con la intención de regalarle algún producto de los que aparecen en su lista de deseos. Esta búsqueda la puede realizar introduciendo el nombre o apellidos del usuario buscado o la dirección de correo electrónico.

5. Consultar lista de deseos: El usuario puede visualizar aquellos productos de la lista de deseos de otro usuario que aún no le han sido regalados, y que por lo tanto le puede comprar.

6. Consultar noticia: El usuario puede visualizar información relativa a las últimas noticias que han aparecido en el servidor de noticias del *mall*.

2.2.2 Opciones del actor Usuario Anónimo

1. Darse de alta: El usuario introduce sus datos personales, quedando almacenados en el sistema de forma permanente.

2. Registrarse: El usuario introduce su identificador y su clave, y si la autentificación tiene éxito, será reconocido por el sistema.

2.2.3 Opciones del actor Usuario Registrado

1. Comprar productos: Un usuario elige una serie de productos que quiere comprar, se obtiene el importe total de los productos junto con los impuestos y gastos de envío, y se procesa el pedido para que se le pueda enviar al cliente cuando se considere oportuno.

2. Actualizar los datos personales: El usuario consulta sus datos personales, y modifica aquellos que sean erróneos, quedando los cambios almacenados en el sistema de forma permanente.

3. Cambiar de *password*: El usuario cambia el *password* con el que accede al sistema por otro nuevo.

4. Salir del sistema: El usuario termina de realizar sus operaciones y abandona el sistema.

5. Ver la lista de deseos: El usuario visualiza el contenido de su lista de deseos para ver el estado de los productos que quiere que le regalen.

6. Añadir un producto a la lista de deseos: El usuario selecciona un producto que le interesa y lo pone en su lista de deseos con la intención de que alguien se lo regale.

7. Ver los pedidos realizados: El usuario obtiene una lista con todos los pedidos que ha realizado en el *mall* junto con los datos de ese pedido y su estado actual (servido, no servido o cancelado).

8. Obtener información detallada de un pedido: El usuario visualiza la información relativa a un pedido concreto de forma detallada, es decir, cada una de las líneas que componen el pedido.

9. Cancelar un pedido: El usuario, si se arrepiente de haber realizado un determinado pedido y este aún no se ha servido, puede cancelarlo para que no se llegue a hacer efectivo.

10. Realizar una crítica de un producto: El usuario realiza un crítica de un producto que ha comprado o consultado y le da una valoración. Posteriormente, otro usuario podrá ver esta información, de forma que le ayude a decidir si compra el producto o no.

2.2.4 Opciones del actor Administrador de Tienda

1. Realizar el mantenimiento de los producto de la tienda: El administrador de tienda se ocupa de crear nuevos productos y de actualizar aquellos con datos erróneos u obsoletos.

2. Realizar el mantenimiento de las categorías de la tienda: El administrador de tienda se encarga de crear y actualizar las categorías en las que se clasifican los productos existentes en la tienda.

2.2.5 Opciones del actor Administrador de *Mall*

1. Visualizar pedidos pendientes: El administrador de *mall* obtiene una lista de todos aquellos pedidos que se han realizado y que todavía no han sido servidos.

2. Marcar un pedido como servido: El administrador de *mall* analiza los pedidos pendientes y si hay suficiente material en *stock* para cumplimentar un pedido, lo selecciona para envío y lo marca como servido.

3. Realizar el mantenimiento de los países del sistema: El administrador de *mall* se ocupa de crear nuevos países disponibles en el sistema y de actualizar aquellos con datos erróneos u obsoletos.

4. Realizar el mantenimiento de las formas de envío del sistema: El administrador de *mall* se ocupa de crear nuevas formas de envío dentro de cada país en el sistema y de actualizar aquellas con datos erróneos u obsoletos.

5. Realizar el mantenimiento de las formas de pago del sistema: El administrador de *mall* se ocupa de crear nuevas formas de pago en el sistema y de actualizar aquellas con datos erróneos u obsoletos.

6. Realizar el mantenimiento de los proveedores: El administrador de *mall* se ocupa de crear nuevos proveedores de material en el sistema y de actualizar aquellos con datos erróneos u obsoletos.

7. Realizar el mantenimiento de las noticia del servidor: El administrador de *mall* es el responsable de insertar y actualizar las noticias del servidor que serán accesibles para los usuarios a través de las páginas de la aplicación.

8. Consultar *stock*: El administrador del *mall* puede consultar en cada una de las tiendas aquellos productos que tienen su número de existencias por debajo de un determinado umbral.

9. Gestionar el *stock*: El administrador del *mall* se encarga de actualizar las unidades existentes de los productos de las tiendas cuando recibe nuevo material de los proveedores.

10. Consultar un proveedor de material: El administrador del *mall* puede acceder a los datos del proveedor de cualquier producto de sus tiendas con el fin de poder ponerse en contacto con él y pedir un reabastecimiento de material.

3 El Problema

El sistema de *comercio electrónico* desarrollado dispone de una gran variedad de opciones y facilidades, orientadas todas ellas a favorecer las compras de los usuarios. Incluso, si se desease, se podrían implementar nuevas opciones para aumentar las capacidades del sistema.

Pero estaríamos limitando mucho el sistema si nos centrásemos únicamente en las funcionalidades que la aplicación ofrece. Dado que se trata de un sistema abierto al gran público mediante Internet, habría que tener en cuenta los distintos tipos de usuarios que van a conectarse al sistema.

Este aspecto es especialmente relevante en el hecho de que, dados los continuos avances tecnológicos que se están produciendo en los últimos años, la gama de dispositivos electrónicos disponibles en el mercado es muy amplia y cada uno de ellos dispone de unas características propias que se adaptan a las condiciones de los diversos usuarios.

Al interactuar con el sistema de *comercio electrónico* se deberían tener en cuenta las características y capacidades de cada dispositivo de cara a sacarle un rendimiento óptimo a cada uno de ellos. Por ejemplo, un usuario que accede al sistema utilizando un PC con un navegador tradicional, puede visualizar una cantidad de información por pantalla muchísimo mayor que una persona que se quiere conectar mediante un dispositivo WAP. Además, también puede ser muy diferente la forma de navegación recomendable para cada uno de los dispositivos.

Pero dejando a un lado las diferencias físicas obvias entre algunos de los dispositivos que podrían utilizar los usuarios, hay que pensar también que el lenguaje que puede entender cada uno de estos dispositivos no tiene por qué ser el mismo, lo cual obliga a que la respuesta ante una petición sea distinta para cada uno de ellos.

De esta manera, el resultado que debería devolver el sistema ante una petición realizada por un dispositivo WAP (formato WML) es muy diferente de si la petición se recibe de un navegador tradicional (formato HTML). Incluso se debería poder construir una página HTML específica para cada navegador (Internet Explorer o Netscape), que a pesar de entender en teoría el mismo formato HTML, ofrecen ciertas diferencias que podemos desear controlar.

Este distinto comportamiento ante peticiones de distintos dispositivos, nos lleva a pensar que posiblemente habría que tener diversas aplicaciones corriendo, cada una de ellas atendiendo a peticiones de un dispositivo concreto. Sin embargo, dada la gran variedad de dispositivos existentes, esta solución no parece viable.

Además, la esencia del sistema de *comercio electrónico* es la misma para todos los casos, ya que el motor de la aplicación, y el contenido y funcionalidades que se pretenden ofrecer a los usuarios es común a todos ellos, por lo que parece un gasto excesivo e innecesario de recursos para resolver el problema que se nos plantea.

Por lo tanto, habría que intentar buscar una solución que nos permitiese utilizar un núcleo común del sistema, pero generando visualizaciones distintas que se adapten a las características particulares del dispositivo con el que se está realizando la petición.

4 Cómo se Resuelve

La solución al problema planteado consiste en definir cuatro subsistemas independientes que se encarguen de las siguientes funciones:

- El tratamiento de la petición y la obtención de la información y criterios necesarios para procesarla correctamente.

- La generación de un documento XML que contenga toda la información necesaria para proporcionar el resultado pero sin aportar ningún tipo de formato específico que esté orientado a una visualización concreta.
- Determinación del tipo de transformación a aplicar en función de las características de la petición recibida y del contexto.
- La transformación de este contenido general en la visualización deseada (transformación XSL) en función de los criterios determinados con anterioridad.

La arquitectura final del prototipo implementado puede verse en la figura 1. En este modelo se ofrece una descomposición del sistema global en una serie de subsistemas independientes que se intercambian mensajes.

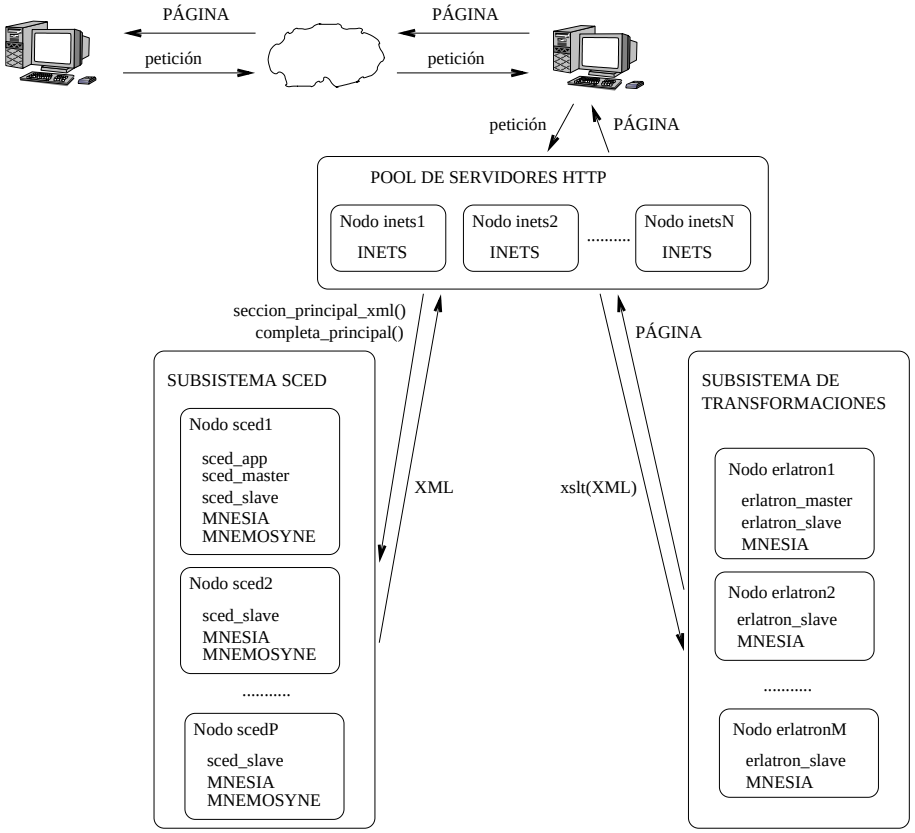

Figura 1: Arquitectura final del prototipo

En el sistema implementado existen tres subsistemas perfectamente diferenciables:

1. Subsistema de servidores Web.

   En un conjunto de nodos van a estar corriendo servidores HTTP (INETS). Este subsistema permitirá el acceso de los clientes al sistema de comercio electrónico.

   Las peticiones que llegan a cualquiera de los servidores HTTP, son interpretadas y redirigidas a los otros subsistemas. Como resultado se obtendrá la página solicitada en el formato deseado, que será enviada como respuesta. La existencia de varios INETS permite dotar al sistema de la redundancia necesaria para que, ante el fallo de cualquiera de los servidores, pueda seguir siendo accesible a través de cualquiera de los otros.

   Para conseguir que de cara al exterior parezca que el punto de entrada es único, se puede configurar una máquina para que funcione como *DNS Round Robin*, de manera que se encargue de redirigir las peticiones que le lleguen a los distintos servidores HTTP.

2. Subsistema `sced` (Sistema de Comercio Electrónico Distribuido).

   Sobre una serie de nodos va a estar corriendo la aplicación distribuida `sced_app`. Este subsistema es el encargado de construir el documento XML en función de los parámetros de la petición recibida y del contexto actual. Así, según la página en la que se encuentre un usuario y el perfil del mismo, este subsistema construirá documentos XML distintos.

   En cada uno de los nodos en los que corre esta aplicación, va a existir un `sced_slave`, que será el encargado de construir el documento XML. En uno sólo de esos nodos, va a estar el `sced_master` que será globalmente accesible, delegando las peticiones que reciba en los `sced_slave` del subsistema para que estos realicen el trabajo.

3. Subsistema de transformaciones (`erlatron`).

   Para la implantación de este subsistema se utilizó una aplicación disponible (`erlatron`), que va a estar corriendo sobre otra serie de nodos. Este subsistema se encarga de realizar la transformación del documento XML que construye el sistema `sced` aplicando una hoja de estilos XSL.

   El fichero XSL que debe aplicar lo elige en función de una serie de reglas, que dependen principalmente del navegador que esté usando el usuario y de la tienda en la que se encuentre.

   Al igual que en el sistema `sced`, existe un `erlatron_master` escuchando peticiones en uno de los nodos, y un `erlatron_slave` corriendo en cada uno de ellos, y con la responsabilidad de realizar la transformación.

El procedimiento para descargar una página es el siguiente:

1. Uno de los servidores HTTP recibe una petición.
2. Como resultado de la petición, se ejecuta un `erlet`.
3. Este `erlet` se encarga de coger de la petición toda aquella información relativa a la sesión y capturar la opción que se debe ejecutar.
4. Después se le pide al módulo correspondiente a la opción, que obtenga de la petición aquellos parámetros que necesita para llevar a cabo su funcionalidad, mediante la llamada a una función del módulo (`obtener_parametros`), de acuerdo con la interfaz que debe cumplir cualquier opción implementada.
5. Estos parámetros, junto con la información relativa a la sesión, se le pasan a otra función del módulo correspondiente a la opción que se a a utilizar (`seccion_principal_xml`).
6. Esta función, tiene la responsabilidad de elaborar el XML con el contenido necesario para construir la página que posteriormente visualizará el usuario. Además del XML, esta función devuelve el tipo de XSL que debe aplicarse para transformar el XML e información sobre la sesión (que será la misma salvo que la sesión haya terminado).
7. Este XML propio de la página se completa con el resto de contenido que se presentará al usuario, como pueden ser los menús o los demás complementos comunes a todas las páginas de la aplicación Web. Es decir, se completa el contenido del documento XML con información del contexto.
8. Con el documento XML y el tipo de XSL a aplicar, se llama al subsistema de transformaciones (`erlatron`) con una serie de parámetros del entorno (como puede ser el tipo de navegador que se esté utilizando), el cual se encargará de elegir el XSL más adecuado en el contexto actual y realizar la transformación.
9. No queda más que devolver el resultado de la transformación (que estará en el formato adecuado: HTML, WML, ...) a INETS, el cual se encargará de devolvérselo al navegador del cliente, ofreciéndole la visualización de la página solicitada.

5 Generalización en Forma de Patrón

La solución explicada para resolver el problema de la generación de múltiples visualizaciones se corresponde con el patrón Multi-Visualización, cuya estructura general se puede ver en la figura 2.

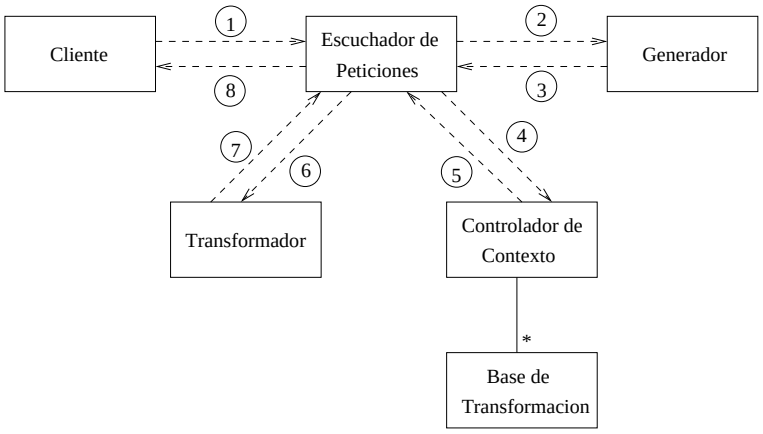

Figura 2: Patrón Multi-Visualización

En este diagrama, podemos distinguir los siguientes componentes:

- `Escuchador de Peticiones`: Se encarga de recoger las peticiones realizadas por el `Cliente`, ocupándose de procesarlas y obtener toda la información posible de la misma. Normalmente, con la petición vendrá información sobre la sesión que permitirá obtener datos necesarios para su procesamiento.

  Además, esta entidad es la encargada de coordinar el funcionamiento de los otros componentes, controlando los posibles errores de interacción entre los mismos (por ejemplo, la no disponibilidad de nodos encargados de construir el contenido o de aquellos que realizan la transformación).

- `Generador`: A partir de la información capturada por el `Escuchador de Peticiones`, este proceso se encarga de construir la respuesta en un formato estándar que contenga únicamente contenido, y nada de información relativa al aspecto que se le pueda dar a estos datos.

- `Controlador de Contexto`: A partir de la información capturada por el `Escuchador de Peticiones`, este proceso se encarga de determinar cuál es la `Base de Transformación` específica que debe aplicarse sobre el contenido construido con anterioridad.

- `Base de Transformación`: Entidad que permite, a partir del contenido generado en formato estándar, obtener una visualización en un formato específico.

- `Transformador`: A partir del contenido obtenido, este proceso aplica la `Base de Transformación` seleccionada por el `Controlador de Contexto` para obtener la salida en el formato deseado.

- `Cliente`: Entidad que interactúa con el sistema a través de peticiones esperando obtener el resultado en un formato acorde con sus características.

El funcionamiento de estos elementos es el siguiente:

- El `Cliente` interactúa únicamente con el `Escuchador de Peticiones`.
- El `Escuchador de Peticiones` se ocupará de procesar la petición e interactuar con el resto de los componentes para obtener el resultado en el formato adecuado, el cual será devuelto al `Cliente`.
- En primer lugar interactúa con el `Generador` pasádole la información recibida con la petición, pera que este construya el *contenido* de lo que será la futura visualización. Este *contenido* es equivalente para cualquier futura visualización y, por lo tanto, deberá estar construido en un formato estándar fácilmente transformable en otros formatos y que no contenga información sobre el aspecto que tendrá la salida (sólo tendrá datos sobre el contenido).
- A continuación, el `Escuchador de Peticiones` interactúa con el `Controlador de Contexto`, el cual es capaz de determinar a partir de la petición y la información del contexto cuál es la `Base de Transformación` que se debe aplicar. Normalmente, esto dependerá de la entidad que esté realizando la petición (`Cliente`), ya que la transformación tiene como finalidad dar un formato al *contenido* para que la visualización se adapte a sus características particulares.
- Una vez que tenemos el *contenido* que se quiere ofrecer y la *transformación* concreta que tenemos que aplicar, tan sólo tendremos que pasarle esta información al `Transformador` para que se encargue de aplicarla.
- El resultado obtenido tras realizar la transformación contará con el contenido que se desea ofrecer al `Cliente` pero estructurado en un formato apto para ser entendido correctamente por este. Por lo tanto, este resultado será el que se devuelva al `Cliente` y será específico para el tipo de dispositivo que haya hecho la petición.

Mediante la utilización de este patrón, se obtienen una serie de ventajas e inconvenientes como pueden ser:

- Ofrecer visualizaciones distintas a unos mismos contenidos, sin necesidad de duplicar elementos del sistema.
- División del proceso de generación del resultado de la petición en una serie de componentes independientes que pueden optimizarse de manera individual.

- Abstracción del tratamiento de peticiones en una serie de elementos con un único punto de entrada de cara al `Cliente`.
- Facilidad para la distribución en conjuntos de nodos de cada uno de los componentes explicados.
- Aumenta la complejidad del sistema.
- Ante la aparición de nuevos dispositivos, se pueden crear visualizaciones que se adpten a sus características sin necesidad de modificar el motor del sistema. Únicamente habrá que corregir el `Controlador de Contexto` para que sea capaz de detectar el nuevo dispositivo, y añadir una nueva `Base de Transformación`.

6 Adaptación del Patrón

Para una mayor comprensión de los elementos que costituyen el patrón, a continuación se van a identificar cada uno de los componentes genéricos del patrón, con aquellas entidades propias del caso particular estudiado.

En el sistema de *comercio electrónico*, utilizamos el patrón Multi-Visualización para estructurar todo el proceso de generación de las pantallas que visualiza el usuario. La estructura general del patrón adaptado a nuestro caso particular se puede ver en la figura 3.

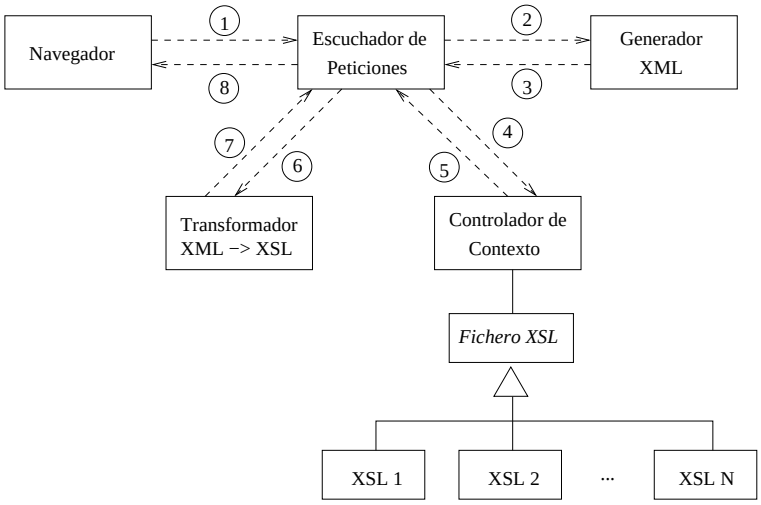

Figura 3: Generación de Páginas (Multi-Visualización)

En este caso, el `Navegador` (`Cliente`) realizará una petición sobre el servidor web (`Escuchador de Peticiones`) que mediante la invocación del

erlet coordinará todos los demás componentes de la arquitectura para generar el resultado.

El módulo de la opción hará las funciones de `Generador`, construyendo el documento XML con la información necesaria para la generación de la página.

Después, el propio *erlet*, obtendrá la información de contexto (`Controlador de Contexto`) a partir de la petición.

El `Controlador de Contexto` conoce a la familia de ficheros XSL (`Base de Transformación`) y los tiene clasificados según una serie de criterios determinados. Con la información obtenida de la petición que le pasa el `Escuchador de Peticiones`, es capaz de determinar el `Fichero XSL` adecuado.

La información de contexto y el documento XML se le envía al subsistema de transformaciones `erlatron` (`Transformador`) que procesará la petición y devolverá la página en el formato adecuado (HTML, WML, ...) para que pueda ser visualizado con el dispositivo usado para la conexión por parte del `Cliente`.

7 Conclusiones

Como puede verse, la utilización del patrón Multi-Visualización ha solucionado el problema de ofrecer distintas visualizaciones según el dispositivo que un usuario esté utilizando para conectarse al sistema de *comercio electrónico*.

Pero esta solución, aunque ha sido obtenida de un caso particular, no tiene una vinculación directa y única con las aplicaciones de *comercio electrónico*, sino todo lo contrario. Durante el proceso de análisis y explicación del patrón, se ha partido del caso particular del sistema estudiado para, posteriormente, realizar una generalizando hasta llegar a un nivel de abstracción, en el cual los distintos componentes y características del patrón extraído son totalmente independientes del caso particular de las aplicaciones de comercio electrónico.

Por lo tanto, se puede concluir que el patrón obtenido tiene unas características generales y, dado que ya ha sido probado que es útil para resolver un problema concreto, podrá ser utilizado para solucionar casos semejantes que surjan en el desarrollo de nuevos sistemas.

De esta manera, a partir de este momento, se dispone de una nueva herramienta que pueden usar los desarrolladores de *software* cuando se encuentren con el problema de implementar sistemas que ofrezcan diferentes visualizaciones para unos mismos contenidos.

Ante un problema similar, como ocurre con el resto de patrones, los desarrolladores de aplicaciones no tienen que plantearse nuevamente una solución, sino que como sabe que el patrón Multi-Visualización puede resol-

ver el problema, únicamente tendrá que adaptar sus componentes al caso particular del que se trate.

Por lo tanto, el uso del patrón Multi-Visualización, y en general de cualquier otro patrón, supone una reducción en el esfuerzo y en el coste del desarrollo de un sistema, y en una mayor claridad y reusabilidad del código generado, características muy importantes en cualquier ámbito del desarrollo de *software*.

Referencias

[AWWV95] J. L. Armstrong, M. C. Williams, C. Wikström, and S. R. Virding. *Concurrent Programming in Erlang*. Prentice Hall, 2nd edition edition, 1995.

[Dou03] Bruce Powel Douglass. *Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems*. Addison-Wesley, 2nd edition edition, 2003.

[ERL01] Erlang otp documentation. www.erlang.org, 2001.

[Fer01] José Ramón Gulías Fernández. Desarrollo de un Sistema de Comercio Electrónico Ejecutable en un *Cluster* de Ordenadores. Proyecto Fin de Carrera. Dpto. de Computación. Facultad de Informática de A Coruña, Julio 2001.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns*. Addison Wesley, Reading, MA, 1995.

[Men98] Martin Menzow. *Comercio electrónico: Construcción de ciberalmacenes*. McGraw-Hill, first edition, 1998.

[PM99] Natanya Pitts-Moultis. *XML in record time*. Sybex, 2021 Challenger Driver, Suite 100, Alameda, CA 94501, USA, 1999.

[TS98] G. Winfield Treese and Lawrence C. Stewart. *Designing Systems for Internet Commerce*. Addison-Wesley, Reading, MA, USA, 1998.

[XML01] Xml bible. www.ibiblio.org/xml/books/bible/, 2001.