Un sistema de comercio electrónico sobre un *cluster* de computadores

José R. Gulías
Víctor M. Gulías
Alberto Valderruten

LFCIA
Departamento de Computación
Facultad de Informática
Universidad de A Coruña
Campus de Elviña
15071 A Coruña, España
{gulias,valderruten}@dc.fi.udc.es

Resumen

El artículo aborda el desarrollo de un sistema distribuido que permite generar aplicaciones de comercio electrónico de manera sencilla sobre un *cluster* de computadores, proporcionando rendimiento y escalabilidad a un coste razonable. El sistema cubre las funcionalidades de catálogo y venta de productos, personalización de contenidos utilizando XML/XSL en función del perfil de usuario, su navegador y la tienda concreta, seguimiento de pedidos y gestión de un conjunto de tiendas (*mall*). Para su implementación se utiliza el lenguaje funcional distribuido ERLANG, manejando aplicaciones y patrones de diseño específicos de este lenguaje de programación. El sistema se instaló sobre el *cluster* del laboratorio LFCIA, y se hicieron pruebas para comprobar las propiedades de escalabilidad, tolerancia a fallos, concurrencia, distribución sobre el *cluster*, etc.

Palabras claves: Sistemas distribuidos, *cluster* de computadores, comercio electrónico, patrones de diseño, programación funcional distribuida, escalabilidad, tolerancia a fallos, XML/XSL.

Abstract

This paper shows the development of a distributed system allowing the generation of e-commerce applications over a clustered computer system, offering performance and scalability at a reasonable cost. The functionalities of the system include product catalogue and saling, contents personalization using XML/XSL from the user profile, his navigator interface and the particular commerce, purchase order state information and mall administration. For its implementation, the distributed functional language ERLANG is used, exploiting their applications and design patterns. The system was installed on the LFCIA laboratory cluster, and properties of scalability, fault-tolerance, concurrency and distribution were tested.

Keywords: Distributed Systems, Clustering, e-commerce, Design Patterns, Distributed Functional Programming, Scalability, Fault-Tolerant Systems, XML/XSL.

1. Introducción

Actualmente, tras el auge de las telecomunicaciones y todo aquello relacionado con la Web, el dominio que abarcan los computadores en la vida cotidiana se incrementa día a día. Así, cada vez es más común que cualquier tienda ofrezca a sus clientes nuevas posibilidades para que realicen sus compras de manera más rápida, sencilla y cómoda, mediante aplicaciones de comercio electrónico [13, 10].

Frente a la creciente demanda de aplicaciones de comercio electrónico resultante de esta situación, surge la necesidad de un sistema con el que puedan generar y gestionar aplicaciones de este tipo de una manera sencilla. Este trabajo presenta el desarrollo de un sistema que permite no sólo la gestión de una única tienda sino la de un conjunto de ellas (*mall*). El sistema proporciona las funcionalidades típicas de este tipo de aplicaciones, abordando tanto las opciones de cara a los posibles compradores como aquellas necesarias para los administradores de las distintas tiendas.

El sistema propuesto se ejecuta en un *cluster* de computadores, de manera que pueda ofrecer un buen rendimiento en las condiciones de carga elevadas que se esperan en un futuro, y a un coste razonable. La distribución sobre un *cluster* de computadores es una solución más racional que la de utilizar una sóla máquina más potente, ya que así se gana en escalabilidad, permitiendo la adaptación del sistema a las necesidades reales de uso y ofreciendo a la vez mayores garantías de tolerancia a fallos.

Otro aspecto importante a destacar es que el sistema puede extenderse, permitiendo la generación de nuevas funcionalidades de manera sencilla mediante la creación de nuevos módulos sin necesidad de cambiar el código existente. El sistema se ha diseñado de forma que quede abierta una interfaz a la que se adapten los módulos que se puedan añadir en un futuro. De esta forma, se podrán incluir posteriormente por ejemplo nuevos métodos de pago o nuevas formas de cobro por transporte.

Una aplicación de comercio electrónico debe ofrecer sus contenidos a los usuarios de manera personalizada, para facilitar e incrementar las compras realizadas. En este sistema la personalización se aborda utilizando XML [12, 14, 11, 7] y hojas de estilo XSL [8], separando el contenido de las páginas de la visualización personalizada que se obtiene al aplicar las hojas de estilo.

El lenguaje utilizado para la implementación del prototipo es ERLANG, desarrollado por Ericsson, que presenta unas características idóneas para la creación de sistemas de procesamiento distribuido y tolerantes a fallos [2]. Además, cuenta con un servidor `http` (INETS [3]) y una base de datos distribuida (MNESIA [3]) que se utiliza para el almacenamiento persistente de los datos. También se utilizan las bibliotecas relativas al procesado de XML y XSL, y `erlatron`, puente necesario para utilizar dichas bibliotecas desde el código ERLANG.

2. Descripción del sistema

El sistema de comercio electrónico planteado cubre las funcionalidades básicas de venta y gestión de un conjunto de tiendas (*mall*). De cara al usuario, el sistema ofrece, entre otras, las siguientes funcionalidades:

- Registro de usuarios para permitir la identificación de los clientes del sistema y la personalización de los contenidos en función del perfil de usuario.
- Realización de búsquedas sobre productos que se contemplen en los catálogos de las tiendas.
- Obtención de información detallada sobre productos en los que el usuario esté interesado.
- Manejo de un *carrito de la compra*, al cual el comprador puede ir añadiendo productos según interaccione con el sistema.
- Realización efectiva de la compra de los productos seleccionados, ofreciendo al usuario información detallada y personalizada de la transacción que se está realizando.
- Manejo de una lista de deseos, en la cual el usuario pueda añadir los productos que desea que le sean regalados por otros usuarios.
- Consulta de pedidos realizados, pudiendo cancelar aquellos que todavía no hayan sido servidos.

En la administración de las tiendas, se plantean tres niveles diferentes, cada uno con sus propios responsables: administración a nivel de tienda, administración a nivel de grupos de tiendas y administración a nivel de *mall*. De cara a los administradores, el sistema ofrece, entre otras, las funcionalidades siguientes:

- Creación de nuevas tiendas.
- Mantenimiento de las tiendas: actualización de productos, categorías de productos, proveedores, etc.
- Tramitación y almacenamiento automático de los pedidos realizados por los compradores.

- Facilidades para la gestión de *stocks*.

En cuanto a aspectos propios de la implementación distribuida realizada, cabe destacar las características siguientes:

- El sistema permite el acceso concurrente de múltiples usuarios, equilibrando la carga entre los distintos computadores que componen el *cluster*.
- El sistema es escalable, de manera que se puede aumentar las prestaciones del sistema simplemente conectando nuevas máquinas al *cluster* sin necesidad de modificar la implementación.
- El sistema es tolerante a fallos, de manera que aunque falle alguna máquina la aplicación es capaz de recuperar el estado sin pérdidas de información y de manera transparente al usuario.
- Las páginas de que conste la aplicación están personalizadas en función del perfil del usuario y del navegador que esté utilizando.

Por último, el sistema deja abiertas las interfaces necesarias para que las aplicaciones de comercio electrónico puedan extenderse mediante la construcción de nuevos módulos sin necesidad de modificar el código existente.

3. Análisis de requisitos

La metodología seguida para el análisis del dominio se basa fundamentalmente en una notación UML con una serie de secciones prefijadas que permiten un análisis ordenado [9], así como en el uso de técnicas para realizar un análisis más preciso del dominio [4].

Una vez enunciada la descripción del problema e identificados los clientes, los objetivos y las diferentes funciones y atributos del sistema [6], se ha optado por tener una única jerarquía de actores (figura 1) que tiene como raíz a un usuario genérico. Se ha asumido que los administradores pueden ejercer el rol de usuarios normales y que, por lo tanto, pueden tener acceso a las funcionalidades típicas de los clientes potenciales además de aquellas propias de la administración. En caso de querer separar explícitamente a los administradores de los clientes sería necesario tener dos jerarquías distintas.

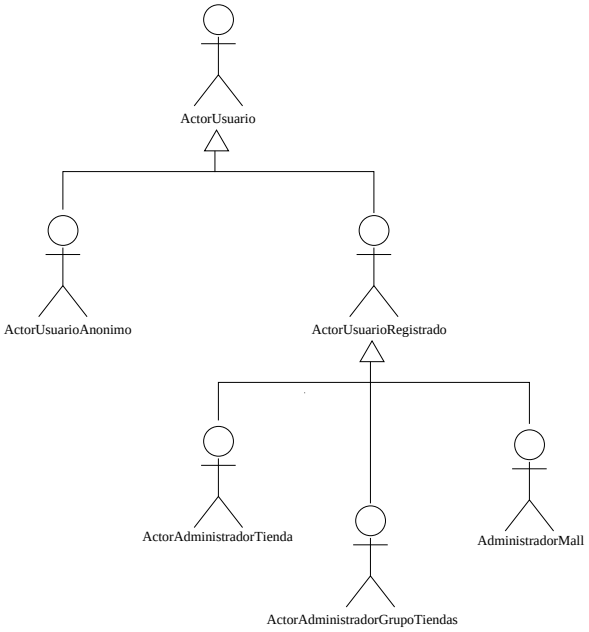

Figura 1: Actores

Los actores que componen la jerarquía van a tener unas responsabilidades determinadas, que se van haciendo cada vez más específicas a medida que se descienden niveles en la jerarquía. Así, las responsabilidades específicas de cada actor (a las que hay que añadir aquellas propias de sus generalizaciones en la jerarquía) son determinadas en detalle, y todos los casos de uso asociados son identificados.

Con el fin de presentar el nivel de detalle alcanzado en esta fase del desarrollo, se presenta a continuación la descripción de uno de los actores, el usuario registrado, y de uno de sus casos de uso más interesantes para la aplicación, el correspondiente a la acción de compra de productos.

3.1. Usuario registrado

- Descripción: Este actor representa a aquellos usuarios que están registrados en el sistema y, por lo tanto, se dispone de sus datos y de su historia previa, pudiendo ofrecer unos contenidos personalizados en función de su perfil.
- Opciones: Además de las opciones propias del actor usuario, este actor puede realizar compras de productos, utilizar un *carrito de la compra*, realizar regalos, manejar una lista de deseos, comentar productos, consultar sus pedidos realizados, etc.
- Casos de uso:

Comprar productos
Añadir un producto al *carrito de la compra*
Recalcular el *carrito de la compra*
Eliminar un producto del *carrito de la compra*
Regalar un producto
Felicitar un regalo
Vaciar el *carrito de la compra*
Realizar un pedido
Actualizar los datos personales
Cambiar de *password*
Salir del sistema
Ver la lista de deseos
Añadir un producto a la lista de deseos
Ver los pedidos realizados
Obtener información detallada de un pedido
Cancelar un pedido
Realizar una crítica de un producto

3.2. Caso de uso del usuario registrado: Comprar productos

- Tipo: Primario.
- Descripción: 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. Esta es la principal funcionalidad que debe satisfacer una aplicación de comercio electrónico, y en la cual debe centrarse el mayor esfuerzo para intentar ofrecer al usuario todas las ventajas posibles que faciliten las compras que quiere realizar.

  Así, muchos de los otros casos de uso que se detallan tienen como finalidad tratar de conseguir que el usuario realice la compra del mayor número de productos posible. Por ejemplo, las búsquedas que ofrezca el sistema van a influir en gran medida en que el usuario encuentre un producto o no. De esta forma, la variedad de estilos de búsqueda (por palabras clave, por categoría, etc.) y su rendimiento, está orientado a facilitar las compras del usuario e incrementar su número. De la misma manera, la personalización de los contenidos, disponer de un *carrito de la compra*, las listas de deseos, etc. tienen como fin último favorecer al caso de uso de comprar productos.

  Al tratarse de un caso de uso tan importante, podemos establecer una serie de subcasos de uso y se desarrolla un caso de uso extendido para un escenario concreto, lo que permite especificar de una forma más detallada cómo se desarrolla la acción de realizar una compra por parte de un usuario del sistema.

(a) Caso de uso extendido: Comprar productos contra reembolso.

- Propósito: Realizar la compra de una serie de productos utilizando el pago contra reembolso.
- Descripción: 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 utilizando como forma de pago el pago contra reembolso.
- Tipo: Primario.

Desarrollo típico de eventos:

Acción del actor	Respuesta del sistema
1. Este caso de uso empieza cuando un usuario se conecta al sistema y se registra.	
2. El usuario realiza búsquedas según diferentes criterios y obtiene información de productos como ayuda para sus compras.	
3. El usuario elige productos para añadir al *carrito de la compra*.	4. Tras cada producto añadido al carrito, el sistema debe mostrar el estado actual del carrito, recalculando el importe de la compra y ofreciendo todos los precios en la moneda del usuario.
5. En cualquier momento, el usuario puede modificar el número de unidades de un producto del carrito o eliminarlo.	6. El sistema debe recalcular el importe del producto y los totales.
7. El usuario puede buscar a un usuario por su nombre o por su dirección de correo electrónico.	8. El sistema ofrece una lista de los usuarios que son compatibles con la búsqueda realizada.
9. Consulta la lista de deseos de otro usuario y añade un producto de la lista a su *carrito de la compra* para regalárselo.	10. Muestra el carrito y recalcula el importe.
11. Cuando ha finalizado la selección de productos, el usuario indica que quiere realizar la compra.	12. Se presenta el estado final del carrito junto con el importe y se pide confirmación para procesar el pedido.
13. Confirma la operación.	14. Presenta los datos del usuario que se utilizarán para la facturación y solicita confirmación sobre la corrección de los mismos.
15. Confirma los datos.	16. Agrupa los productos de la compra en pedidos según el lugar de destino, y pregunta por la forma de envío a usar en cada uno de ellos y por la forma de pago que se va a utilizar.
17. Responde con la forma de pago (contra reembolso) y con la forma de envío usada para cada uno de los pedidos.	18. Genera una factura con todos los pedidos asociados y almacena estos datos de forma persistente.
	19. Muestra la factura y los datos de los pedidos al usuario.
	20. Genera un recibo con la operación realizada y lo envía a la dirección de correo electrónico del cliente.
21. El cliente abandona la aplicación.	

Rutas alternativas:

- Línea 5: Si se introducen números incorrectos, el sistema debe rechazar la operación y mantener el carrito en su estado actual.
- Línea 15: Si los datos proporcionados por el usuario están incompletos, se debe indicar el error y solicitar que se completen.
- Línea 17: Si los datos de pago o envío son incorrectos, el sistema debe indicar el error y solicitar que se corrija.

(b) Subcaso de uso: Añadir un producto al *carrito de la compra*.

- Tipo: Primario.
- Descripción: El usuario selecciona un producto que le interesa, añadiéndolo de esta forma a su *carrito de la compra* para que entre en el siguiente pedido que realice el usuario.

(c) Subcaso de uso: Recalcular el *carrito de la compra*.

- Tipo: Secundario.
- Descripción: El usuario puede establecer, para cualquiera de los productos que tiene actualmente en su *carrito de la compra*, el número de unidades que quiere comprar. Con la opción de recalcular, estos cambios se hacen permanentes y se ofrece al usuario el *carrito de la compra* en su nuevo estado.

(d) Subcaso de uso: Eliminar un producto del *carrito de la compra*.

- Tipo: Secundario.
- Descripción: Un usuario puede eliminar un producto que se encuentre actualmente en su *carrito de la compra*, si este ya no le interesa.

(e) Subcaso de uso: Regalar un producto.

- Tipo: Secundario.
- Descripción: Un usuario, tras consultar la lista de deseos de otra persona, mete un producto en su *carrito de la compra*, como regalo a uno de los deseos de ese usuario.

(f) Subcaso de uso: Felicitar un regalo.

- Tipo: Secundario.
- Descripción: Un usuario, cuando selecciona un producto para regalárselo a otro, tiene la posibilidad de escribir un texto a modo de felicitación para que el beneficiario lo lea cuando consulte su lista de deseos.

(g) Subcaso de uso: Vaciar el *carrito de la compra*.

- Tipo: Secundario.
- Descripción: Un usuario puede eliminar todos los producto que se encuentren en su *carrito de la compra*, si desea comenzar la selección nuevamente.

(h) Subcaso de uso: Realizar un pedido.

- Tipo: Primario.
- Descripción: Un usuario confirma que desea comprar todos los productos que se encuentran actualmente en su *carrito de la compra*. La compra realizada se va a agrupar en una serie de pedidos cada uno de ellos con un destino diferente (debido a que en su compra pueden existir regalos dirigidos a personas situadas en diferentes puntos). Para cada uno de los pedidos generados, el usuario debe proporcionar los datos de envío y especificar el método de envío que se utilizará de entre los disponibles en el país de destino del pedido. Además, deberá proporcionar los datos que se utilizarán para la facturación de los pedidos, así como la forma de pago que se va a usar.

4. Diseño

Tras el análisis de requisitos, que se completa con un modelo conceptual que sitúa los conceptos relevantes extraídos del análisis del dominio y las relaciones existentes entre ellos, se refina dicho modelo hacia su posterior implementación mediante un diagrama de clases, ambos presentados en [6].

Para detallar la interacción entre los distintos elementos del modelo, se utilizaron diagramas de interacción. Como la mayoría son muy sencillos y carece de importancia detallarlos de manera exhaustiva, se incluye sólo el referente a la compra de productos (figura 2).

Revisando los conceptos del dominio que tienen interés, se obtuvo el análisis de requerimientos para el diseño de la base de datos, y posteriormente el modelo entidad-relación y el diseño de todas las tablas propias de la implementación del prototipo, optando por algunos casos de desnormalización para favorecer la eficiencia. Tampoco se incluyen detalles de este desarrollo que se explica en [6].

4.1. Patrones de diseño utilizados

Los patrones de diseño son soluciones a problemas comunes, que han resultado exitosos en proyectos previos y que, por lo tanto, podrían ser reutilizados ante otros problemas similares. Durante el desarrollo del sistema, han surgido problemas que se han resuelto mediante patrones de diseño de GoF [5] y que se detallan a continuación:

(a) Representación de la jerarquía formada por las tiendas del *mall*.

- Patrón: composición.

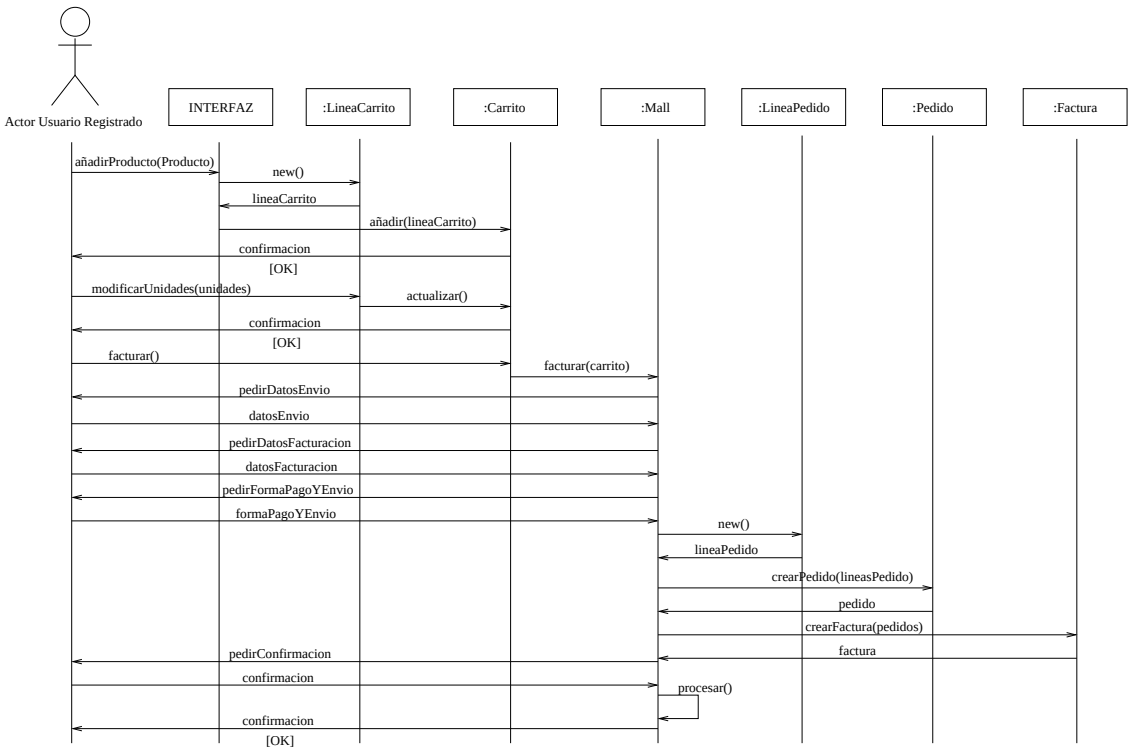

Figura 2: Compra de productos

- Descripción: Un *mall* está compuesto por una serie de grupos, los cuales, a su vez, contienen varias tiendas cada uno de ellos. Esto se puede representar por medio de una clase `Tienda` y una clase `GrupoTienda` que se generalizan en una clase `Mall`. Un `GrupoTienda` tiene asociado un conjunto de elementos de la clase `Mall`. De esta manera, le damos al *mall* una estructura jerárquica y nos permite trabajar con él como una unidad independientemente de su composición.

(b) Representación de los productos.

- Patrón: composición.
- Descripción: Un producto puede ser simple o estar formado por una serie de productos simples. Esto se puede representar por medio de una clase `ProductoSimple` y una clase `ProductoCompuesto` que se generalizan en la clase `Producto`. Un `ProductoCompuesto` estará formado por una serie de elementos de la clase `Producto`. De esta manera, se pueden manejar agrupaciones de productos de cualquier complejidad y trabajar con ellas como una unidad independientemente de su composición.

(c) Configuración de productos.

- Patrón: decorador.
- Descripción: Un producto puede ofrecerse al público con una serie de variaciones que apenas afectan a la entidad como, por ejemplo, los colores a los que se pone a la venta un determinado producto. Para poder manejar productos a los que se aplique cualquier número de variantes, se crea la clase `DecoradorProducto` de la cual colgarán cada una de estas variantes. El `DecoradorProducto` tendrá asociado a un producto, el cual puede a su vez tener cualquier grado de complejidad. Así, mantenemos el tratamiento de un producto como una unidad, y el `DecoradorProducto` puede modificar ciertos aspectos del comportamiento del mismo.

(d) Utilización de diferentes formas de pago.

- Patrón: estrategia.
- Descripción: El pago de una factura puede realizarse utilizando distintas formas de pago. Sería interesante poder cambiar la forma de pago simplemente asociando la clase que necesita realizar el pago con el módulo de pago adecuado. En este caso, se dice que el módulo de pago se comporta como una estrategia.

(e) Utilización de diferentes formas de envío.

- Patrón: estrategia.
- Descripción: El precio que se añade a un pedido debido a los gastos de envío, depende de la forma de envío utilizada. Se puede utilizar una estrategia también para la forma de envío y que simplemente haya que asociar la clase que necesita calcular el coste con el módulo de forma de envío adecuado.

(f) Representación de elementos que deben ser únicos y conocidos por todos.

- Patrón: instancia única (o `Singleton`).
- Descripción: Muchos de los elementos que aparecen en el sistema tienen el comportamiento que representa el patrón `Singleton`. Por ejemplo, el concepto de *mall* se puede representar como una instancia única. Desde muchos elementos del sistema, se necesita invocar operaciones asociadas al *mall* y, por lo tanto, este debe ser accesible y conocido por todos estos elementos. También pueden considerarse `Singletons` el subsistema encargado de construir el documento XML de la página y el subsistema encargado de realizar las transformaciones XSL.

(g) Diseño de la arquitectura distribuida y tolerante a fallos.

ERLANG dispone también de una serie de patrones, a los cuales designa con el nombre de `behaviours` [3]. Dos de ellos (*servidor genérico* y *aplicación*), se utilizaron en la implementación del sistema, desempeñando un papel importante en la arquitectura final diseñada. A continuación se pasa a detallar el funcionamiento de cada uno de ellos:

- *behaviour application* (aplicación): Sirve para representar un conjunto de módulos como una aplicación, de forma que se puedan manejar de manera unitaria. La interfaz que debe implementar consta de únicamente dos funciones fundamentales: una para arrancar la aplicación y otra para detenerla. Las aplicaciones se configuran por medio de un fichero en el cual se especifican sus características como pueden ser los parámetros por defecto que utilizará la aplicación cuando arranca, las fases de que consta el arranque, los módulos de que consta, las aplicaciones que deben estar corriendo en el nodo para que todo funcione correctamente, etc.

  ERLANG permite especificar propiedades de distribución de aplicaciones, como un caso particular del patrón aplicación comentado. Así, se define en ERLANG el concepto de aplicación distribuida, que consiste en una aplicación que esté corriendo en cualquiera de una serie de nodos. En el caso de que el nodo donde esté actualmente la aplicación se caiga, esta se regenerará en cualquiera de los otros nodos disponibles.

  En el sistema final implementado, tanto el subsistema generador del documento XML (`sced`) como el encargado de realizar las transformaciones (`erlatron`), funcionan según el concepto de aplicación distribuida, lo cual confiere a la arquitectura la distribución y tolerancia a fallos buscados.
- *behaviour gen_server* (servidor genérico): Sirve para representar una arquitectura cliente-servidor de manera sencilla, y con todas las características que este tipo de sistemas necesitan (tolerancia a fallos, especialmente). El servidor cuenta con un estado interno y recibe mensajes procedentes de diversos clientes. Ante estos mensajes, el servidor debe responder y, si es necesario, modificar su estado interno.

  Existen diversos módulos en el sistema que cumplen el comportamiento de servidor genérico como pueden ser los maestros y esclavos, tanto del subsistema `sced` como de `erlatron`.

5. Implementación del prototipo

Un prototipo del sistema fue implementado incrementando paulatinamente su nivel de distribución hasta alcanzar su arquitectura definitiva.

En primer lugar se estableció un prototipo inicial que corría sobre un único nodo, identificándose los siguientes procesos: (a) Un servidor HTTP (INETS); (b) El proceso gestor de base de datos (MNESIA) con (c) la aplicación para el lenguaje de consultas (MNEMOSYNE); (d) Los procesos que implementan la lógica del sistema `sced_app` utilizando una arquitectura cliente/servidor con un maestro (`sced_master`) y un único esclavo (`sced_slave`); (e) El motor de transformaciones XSL, configurado con un maestro (`erlatron_master`) y un único esclavo a cargo de realizar las transformaciones (`erlatron_slave`).

Con este primer modelo, únicamente se pretendía comprobar que el sistema proporcionaba las funcionalidades que necesitan las aplicaciones de comercio electrónico y que ofrecía todas las características mencionadas de personalización de contenidos y de visualización. Sin embargo, este modelo presenta una serie de inconvenientes:

- Sólo existe un único servidor HTTP y, por lo tanto, un único punto de acceso al sistema. En el caso de que este servidor fallase, sería imposible acceder al sistema.
- Ante la caída del nodo, el estado de las distintas sesiones de los usuarios se perdería de manera irrecuperable.
- Todo el procesamiento, tanto para la generación del documento XML como para su posterior transformación XSL, la debe realizar un único nodo. El tratamiento de las peticiones se producirá, por lo tanto, de manera secuencial y será muy sensible al incremento de los niveles de concurrencia.

En un segundo paso, se distribuye la aplicación sobre dos nodos, de manera que la distribución de procesos y aplicaciones sobre ellos se plantea de la siguiente forma:

- En cada uno de los nodos está ejecutándose un servidor HTTP (INETS), proporcionando más de un punto de acceso al sistema, eliminando el cuello de botella detectado en la propuesta previa.
- La base de datos (MNESIA) se distribuye entre los dos nodos, proporcionando redundancia frente a caídas. Cuando el nodo caído se recupera, éste actualiza el contenido de sus tablas de forma transparente a los clientes del sistema. La aplicación para el lenguaje de consultas (MNEMOSYNE) va a estar corriendo en cada uno de los nodos.
- La aplicación `sced_app` se configura como una aplicación distribuida entre los dos nodos, con un único maestro escuchando peticiones y un esclavo en cada uno de ellos. La caída del nodo en el que corre la aplicación distribuida provoca la aparición de un proceso con la misma funcionalidad en el nodo de respaldo (*failover*).
- Existe un maestro de `erlatron` en uno de los nodos y un esclavo en cada uno de ellos, repartiendo trabajo entre los dos nodos y soportando, pues, mayores niveles de concurrencia.

La arquitectura final del prototipo puede verse en la figura 3. En este modelo se ofrece una descomposición del sistema global en una serie de subsistemas independientes que intercambian mensajes.

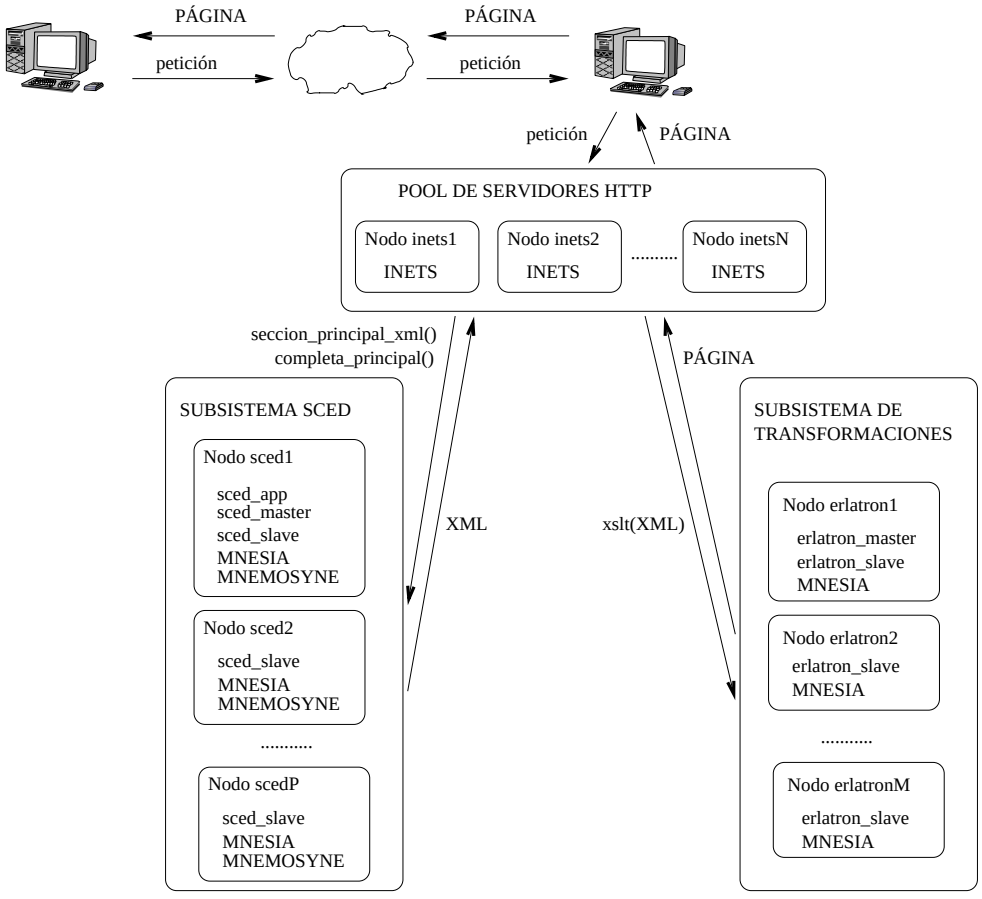

Figura 3: Arquitectura final del prototipo

En el sistema implementado existen tres subsistemas perfectamente diferenciados:

1. Subsistema de servidores HTTP:

   En un conjunto de nodos están corriendo servidores HTTP (INETS). Este subsistema permite 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 el cliente obtiene la página solicitada en el formato deseado. 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 demás. 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 de Comercio Electrónico Distribuido (`sced`):

   Sobre una serie de nodos está ejecutándose 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, existe un `sced_slave`, que es el encargado de construir el documento XML. En uno sólo de esos nodos está situado el `sced_master`, que es globalmente accesible, y que delega las peticiones que reciba en los `sced_slave` del subsistema para que éstos realicen el trabajo.

3. Subsistema de transformaciones (`erlatron`):

   Para la implantación de este subsistema se utilizó una aplicación disponible (`erlatron`), que corre 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 es elegido en función de un conjunto 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.

6. Configuración de páginas

La separación entre el contenido y la visualización mediante la utilización de XML y hojas de estilo XSL, supone una ventaja importante frente a otros sistemas. El sistema, ante las peticiones que recibe, genera un documento XML en función de la página actual, el contexto, el perfil del usuario y el contenido de la base de datos. El diseñador puede saber para cualquier página de la aplicación el documento XML que le va a proporcionar el sistema, simplemente con conocer el DTD asociado a esa página.

Como no existe ningún tipo de acoplamiento entre el contenido y la visualización, a partir del mismo documento XML se pueden obtener diversas visualizaciones alternativas, tanto para ser usadas por un mismo navegador como para tener como clientes dispositivos muy diferentes que acepten cualquier tipo de formato existente (HTML, WML) e incluso formatos diseñados específicamente para ese dispositivo.

La manera de indicar al sistema la existencia de una nueva visualización, es por medio de reglas que `erlatron` utilizará para seleccionar la transformación adecuada. Por ejemplo, estas son algunas reglas utilizadas para las páginas de inicio de dos tiendas:

```
{sced_inicio, r1, "text/html", "SCED Inicio en HTML",
  "file://LALUZ/XSL/laluz_inicio.xsl",
  {all, [netscape, {equal, {"id_tienda", "LaLuz"}}]}}.

{sced_inicio, r101, "text/html", "SCED Inicio en HTML",
  "file://SESAMO/XSL/sesamo_inicio.xsl",
  {all, [netscape, {equal,{"id_tienda", "Sesamo"}}]}}.

{sced_inicio, rw1, "text/vnd.wap.wml", "SCED Inicio en WML",
  "file://LALUZ/XSL/WML/wap_inicio.xsl",
  {all, [wap, {equal, {"id_tienda", "LaLuz"}}]}}.

{sced_inicio, rw101, "text/vnd.wap.wml", "SCED Inicio en WML",
  "file://SESAMO/XSL/WML/wap_inicio.xsl",
  {all, [wap, {equal,{"id_tienda", "Sesamo"}}]}}.
```

De esta forma, el subsistema de transformaciones XSL seleccionará el estilo `file://LALUZ/XSL/laluz_inicio.xsl` para el documento `sced_inicio` cuando se cumplan tanto el predicado `netscape` como que el identificador de la tienda sea `LaLuz`.

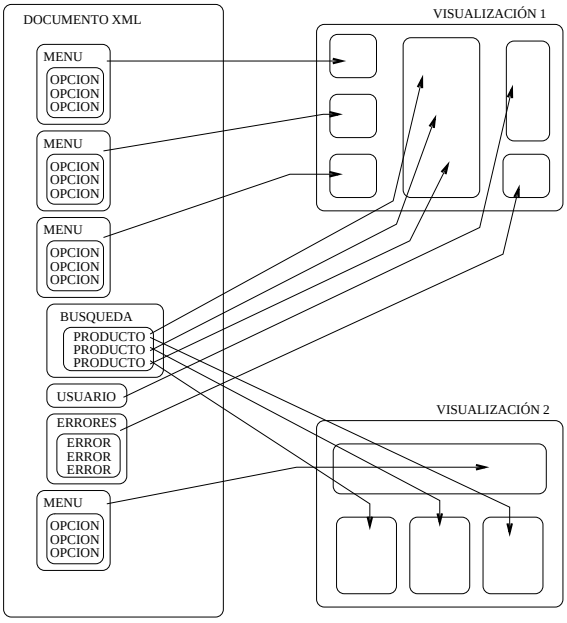

Figura 4: Configuración de páginas

Figura 5: Página inicial de LA LUZ

7. Escalabilidad

A continuación se presentan los resultados de una prueba del sistema variando los nodos disponibles y el nivel de concurrencia en las peticiones. Para el experimento se utiliza el siguiente esquema de configuración en el *cluster* de computadores de consumo descrito en [1] en el que se dispone de un nodo frontal (*Borg*, Pentium II dual 350MHz, 384MB RAM) y varios nodos (AMD K6 300MHz, 96MB RAM):

- 1 INETS en `Borg`.
- 1 `erlatron_master` y 1 `erlatron_slave` en `Borg`.
- 1 `sced_app`, 1 `sced_master` y n `sced_slave` en los nodos del *cluster* `Borg`.

La figura 6 muestra que mientras exista un número de nodos `sced` suficiente para atender a todas las peticiones, el tiempo medio de respuesta apenas aumenta. Ello es debido a que todas estas peticiones se pueden procesar en paralelo utilizando cada una de ellas un `sced_slave` diferente. Cuando se supera el número de nodos disponibles, se aprecia que el tiempo medio de servicio aumenta de manera proporcional, debido a que las peticiones que no se pueden atender deben esperar.

	1 nodo	2 nodos	4 nodos	8 nodos
Concurrencia 1	102 / 9.66	102 / 9.68	103 / 9.64	107 / 9.22
Concurrencia 2	177 / 11.25	117 / 16.94	116 / 17.11	120 / 16.60
Concurrencia 4	356 / 11.22	197 / 20.23	137 / 28.95	148 / 26.86
Concurrencia 8	734 / 10.89	389 / 20.45	229 / 35.21	203 / 39.15
Concurrencia 16	1466 / 10.82	783 / 20.14	469 / 34.27	400 / 39.72
Concurrencia 25	2318 / 10.73	1253 / 19.74	733 / 34.17	599 / 41.20

Figura 6: Tiempo medio de respuesta (ms) / Peticiones por segundo utilizando los discos locales

Como puede observarse en la misma figura (segundo valor en las celdas de la tabla), el número de peticiones por segundo que puede atender el sistema aumenta mientras hay nodos `sced` libres que puedan satisfacer esas peticiones.

8. Conclusiones

El planteamiento utilizado para conseguir la separación entre el contenido y la visualización, mediante XML y hojas de estilo XSL, parece una solución muy adecuada, frente a otros métodos en los que no existe una frontera bien delimitada.

Resulta interesante el hecho de que una misma aplicación escuchando en un puerto de una máquina pueda responder a peticiones de distintos dispositivos y en sus formatos correspondientes. Se trata de una solución más escalable que tener aplicaciones distintas escuchando en puertos diferentes, sólo por el hecho de que la visualización cambie. La solución adoptada en el sistema desarrollado consiste en tener una aplicación que genere un documento XML (común para cualquier visualización) y transformarlo según el formato que se desee.

El funcionamiento del sistema utilizando un navegador tradicional (HTML) fue el esperado. Además, como se pudo comprobar al utilizar un emulador WAP como prueba, la adaptación a un nuevo formato es muy sencilla de realizar, siendo necesaria únicamente la definición de nuevos ficheros XSL que generasen la salida en el formato deseado (WML).

La distribución del sistema sobre un *cluster* es la solución más escalable y rentable para este tipo de aplicaciones que tienen previsiones de crecimiento futuro. Las pruebas realizadas permitieron probar el rendimiento del sistema sobre un entorno distribuido en distintas condiciones de carga y configuración. Se pudo comprobar que a medida que se añadían más nodos al sistema, aumentaba el número de peticiones que podían ser atendidas y se reducían los tiempos medios de espera de cada petición.

En este tipo de sistemas también es importante proporcionar mecanismos para que puedan crecer ofreciendo nuevos servicios a sus usuarios. En el sistema propuesto se han definido interfaces que deben implementar nuevos módulos de opciones, formas de pago, formas de envío, etc. De esta manera, cualquier programador puede implementar nuevas funcionalidades que hagan crecer al sistema.

La utilización del lenguaje funcional ERLANG para la implementación del prototipo, ha simplificado su elaboración. El uso de patrones en ERLANG simplifica el mantenimiento de los procesos. En particular, el uso de servidores genéricos para implementar sistemas cliente-servidor confieren al sistema un grado de robustez importante. El uso de una base de datos distribuida como MNESIA también ha sido de gran utilidad, pudiéndose crear bases de datos

robustas y distribuidas, sin tener que preocuparnos excesivamente de los mecanismos de distribución de las distintas tablas.

Con las pruebas realizadas se llegó a la conclusión de que el cuello de botella del sistema está en el subsistema que construye el documento XML con la información relevante de la página. El número de nodos del subsistema de transformaciones XSL, apenas influye en el rendimiento global del sistema. El esfuerzo de cara a la implementación de la arquitectura de un sistema final debe orientarse a intentar equilibrar el nivel de trabajo de ambos subsistemas: si no es posible reducir el tiempo necesario para generar el documento XML puede plantearse la creación de éste a partir de pequeños trozos de XML, generados cada uno de ellos como trabajos independientes.

Referencias

[1] Miguel Barreiro and Victor M. Gulias. Cluster setup and its administration. In Rajkumar Buyya, editor, *High Performance Cluster Computing*, volume I. Prentice Hall, 1999.

[2] G. Coulouris, J. Dollimore, and T. Kindberg. *Distributed Systems: Concepts and Design*. Addison-Wesley, third edition, 2000.

[3] The complete Erlang/OTP documentation. Disponible en `http://www.erlang.org/doc.html`, 2000.

[4] Martin Fowler. *Analysis Patterns: Reusable Objects Models*. Object Technology Series. Addison-Wesley, 1997.

[5] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison Wesley, Reading, 1996.

[6] José R. Gulías. Desarrollo de un sistema de comercio electrónico ejecutable en un cluster de ordenadores. Memoria de proyecto de fin de carrera de Ingeniería Informática, Universidad de A Coruña, 2001.

[7] Elliotte Rusty Harold and W. Scott Means. *XML in a Nutshell*. California O'Reilly, first edition, 2001.

[8] http://www.ibiblio.org/xml/books/bible/. Xml bible, 2001.

[9] Craig Larman. *Applying UML and Patterns*. Prentice Hall PTR, Upper Saddle River, NJ, 1998.

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

[11] Natanya Pitts. *XML*. Anaya Multimedia, first edition, 1999.

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

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

[14] Kevin Williams, Michael Brundaye, Jeff Gabriel, Andy Hoskinson, Michael Kay, Thomas Maxwell, Marcelo Ochoa, Johnny Papa, and Mohan Vanmane. *XML Databases*. Anaya Multimedia, first edition, 2000.