Apache Cocoon 2

Motivación, Introducción y Explicación

Saúl Zárrate Cárdenas

Apache Cocoon 2: Motivación, Introducción y Explicación

por Saúl Zárrate Cárdenas

Historial de revisiones
Revisión 0.0 6 de Mayo de 2002 Revisado por: szc
Creación

Historial de revisiones
Revisión 0.1 5 de Junio de 2002 Revisado por: jidl
Correcciones de ortografía y marcado

Historial de revisiones
Revisión 0.2 17 de Julio de 2002 Revisado por: szc
Adición de formato de reunión semanal como apéndice y organización de directorios para las imagenes y los fuentes

Historial de revisiones
Revisión 0.3 31 de agosto de 2002 Revisado por: fjfs
Cambio de formato de imágenes a png y correcciones ortográficas
Revisión 1.0 18 de Octubre de 2002 Revisado por: jid
Conversión a xml, correccion de errores menores de marcado

Tabla de contenidos

Tabla de figuras

Tabla de ejemplos

Capítulo 1. ¿Por qué Cocoon?

1.1. Motivación

Hoy en día, la presencia en el Web es cada vez más relevante e importante para las empresas. Día a día se demandan más servicios en Internet. Por esto, son requeridos sistemas con gran capacidad de transformación y asimilación de las nuevas tendencias, de las nuevas tecnologías y del constante cambio del propio mercado.

Una característica importante de este tema es lo que atañe a la presentación de la información en múltiples formas y dispositivos. Si tenemos en cuenta el manejo de la información como hasta actualmente se está haciendo y analizamos lo que significa para una empresa tener toda su presentación en páginas HTML, podemos notar que imponer un cambio de imagen en las propias páginas es una labor dispendiosa y debido a que se trata de "remendar" algo ya hecho, se torna en una tarea poco agradable.

Peor aún, si se trata de presentar la información de la empresa en un formato distinto al HTML, ya que además de crear la presentación se debe recolectar la información de la presentación en el formato que está manejando, es decir, el HTML.

Como usted ya se habrá dado cuenta, el tener la información en la misma capa en la que se presenta, genera inconvenientes grandes y desencadena una cantidad de factores negativos para las organizaciones tales como gastos en mantenimiento, mayor tiempo en los desarrollos, pérdida de tiempo y dinero en la capacitación de personas para que conozcan la estructura de presentación, etc.

Para las empresas en las cuáles los datos de presentación se generan de forma dinámica, el problema pasa del diseñador al programador. En estos casos el programador tiene que ir al código y cambiar la presentación, pero pensemos que en realidad ésto no es el trabajo de un programador, es precisamente la presentación, trabajo del diseñador. Es claro que el diseñador no puede hacerlo (y tampoco debe, por que no es su oficio) ya que su línea de trabajo no está pensada para esto. Ésto se da mucho en generación dinámica de aplicaciones que utilizan tecnologías del estilo de Servlets. Sin embargo nótese que el programador, al tener que modificar en su código fuente aspectos de presentación corre un riesgo alto de alterar el funcionamiento de la aplicación, lo cual cae en una solución peor e improductiva.

Una solución mejorada de los Servlets salió con la tecnología J2EE. En los *J2EE Beans* (que son la forma de presentar la información) se debería tener el mínimo código, es decir, el necesario para las clases que contienen toda la lógica del negocio. Por otro lado, con los *taglibs* (etiquetas XML para definir código) se posibilita crear páginas que no tienen una sola línea de código.

Bien, ésto mejora las cosas; sin embargo se siguen teniendo básicamente tres problemas:

1. Si se quieren obtener distintas presentaciones, es necesario modificar el código Java que se encarga de dar formato a los datos

2. El mantenimiento puede no ser tan transparente ya que un diseñador puede alterar el código embebido en el HTML.
3. En ocasiones puede ocurrir que se incluya código Java mas allá del necesario para que puedan funcionar correctamente los *beans*.

Sería óptimo poder tener productos de información que fueran independientes de las distintas formas de presentación y que si ese contenido se generara dinámicamente, ese dinamismo también fuera totalmente independiente. En pocas palabras se quiere *separar Contenido, Lógica y Presentación.*

Si se tuviera en un repositorio toda la información sin ninguna característica que la atara a algún tipo de presentación, se podrían implantar cada una de las vistas que se desearan sin que se debiera tener en cuenta alguna de las otras. Luego, el cambio o el mantenimiento de una de las vistas no le afectaría, sino a ella misma.

1.2. Entornos de publicación web (*Web Publishing Framework*)

Es aquí, donde surgen los *Entornos de Publicación Web Basados en XML y XSL*. En este tipo de aplicación se tienen las ventajas de la tecnología XML, tales como ser un estándar, ser una meta común para las empresas de tecnología, facilidad en la transformación con el apoyo de la tecnología XSL, separación total entre datos y presentación de los mismos, separación entre el rol del programador y el rol del diseñador (y por lo tanto más productividad, menos costos y más paralelismo de trabajo), mejor y más fácil tratamiento al mantenimiento y ser compatible con el resto de tecnologías.

Hasta este punto un entorno de publicación web en xml resuelve el problema contenido-presentación. ¿Pero y la lógica de la aplicación?

Bien, para esta parte existen varias propuestas, pero la más interesante es un proyecto del grupo Apache que denominan XSP (*eXtensible Server Pages*). Para conocer un poco más de XSP vea el Capítulo 4

Como vemos, ya se explicó a grandes rasgos que el entorno de publicación web basado en XML es la mejor solución al problema planteado: Separar Contenido, Lógica y Presentación. Es aquí en donde entra el proyecto del grupo Apache llamado por ellos Apache Cocoon (http://xml.apache.org/cocoon).

> **Importante:** Es importante resaltar que esta solución tiene un problema: Es muy poco madura y aun anda en proceso de prueba lo cual genera expectativas de todo tipo. Cocoon es hasta el momento entre este tipo de soluciones, la más desarrollada y cuenta con gran credibilidad en este momento.

Capítulo 2. Cocoon

2.1. ¿Qué es Cocoon?

Cocoon es un sistema de publicación Web, basado en XML/XSL. Cuenta con desarrollo total en Java por lo cual se puede ejecutar desde cualquier servidor que pueda contener Servlets; y al ser un Servlet cuenta con las ventajas de éstos, es decir, se ejecutan como *threads* de forma simultánea en el mismo contexto y no tienen que llamar a métodos auxiliares como lo hacen tecnologías del estilo CGI.

Cocoon es *Open Source*. Es bastante configurable y personalizable. Además adopta características para escribir páginas de servidor en XML (XSPs). Permite diferenciar el procesamiento del documento para tenerlo en distintos formatos, dependiendo del tipo de software que hace la petición y cuenta con un sistema de caché para tener un mejor rendimiento. Un elemento adicional y clave para tener en cuenta es que es un producto gratuito y por lo tanto no tendrá que gastar dinero para su adquisición.

Su usted desea separar contenido, presentación y lógica en su aplicación, una buena alternativa es adoptar Cocoon.

2.1.1. Funcionamiento a nivel de usuario

Cuando un usuario hace una solicitud, en Cocoon ocurren una serie de fases que consisten en:

1. El usuario solicita un documento de cualquier tipo al servidor.
2. La solicitud se analiza para concluir si se puede atender o no. Si no se puede atender se produce un mensaje de error.
3. Si se puede atender se analiza a qué productor XML corresponde. Se genera un documento XML con el cual se trabajará.
4. Se extraen las instrucciones del XML generado en el paso anterior y éstas se le pasan al procesador apropiado para que se le apliquen al XML. Al procesar el XML podría salir un XML con más instrucciones que serán tratadas en algún otro ciclo.
5. El XML procesado se le pasa al elemento que aplica el formato. Si el documento es un documento final,XML aplica el formato y le envía el documento formateado al cliente. En el caso que el documento XML procesado, sea código que deba ejecutarse (como en el caso de una XSP ya compilada), éste se pasa como productor de XML y se vuelve a procesar hasta que se llega a un documento XML final.

2.2. Cocoon 1 *Vs* Cocoon 2

Cocoon está siendo desarrollado por una parte del equipo *Apache XML*. Cocoon 2 tiene cambios tan significativos con respecto a Cocoon 1, que se podría decir casi que fue escrito de nuevo.

Los desarrolladores de Cocoon 2 dicen que lo que han hecho es aprender de lo que vivieron durante el desarrollo de Cocoon 1, y lo implementaron para mejorar la eficiencia y la escalabilidad del proyecto.

Cocoon 1 trabajaba sobre DOM (*Document Object Model*) para poder pasar los documentos XML entre componentes. El problema es que el trabajo con árboles DOM se torna ineficiente ya que el procesamiento de un árbol consume mucha más memoria que el documento XML original.

Cocoon 2 está construido sobre el API SAX que es mucho más eficaz cuando se trata de manipular documentos XML.

Por otro lado, el manejo de la aplicación cambia bastante de Cocoon 1 a Cocoon 2. Mientras que en Cocoon 1, en los documentos XML se debían incluir las instrucciones para hacer el procesamiento del documento (atando el documento XML a Cocoon), en Cocoon 2 se puede configurar para determinado fichero XML que transformación debe aplicársele, fuera del mismo fichero. Note que ésto es una gran ventaja con respecto a la flexibilidad del sistema, ya que en la versión 1 de Cocoon la reutilización de código se disminuye considerablemente y la capa que separa el contenido de la lógica y la presentación se vuelve casi imperceptible.

Capítulo 3. Estructura y arquitectura de Cocoon

En este apartado me dedicaré a hablar un poco de la estructura interna y arquitectura en la que se basa Cocoon.

3.1. Conceptos claves

Antes de entrar en detalles es recomendable mostrar tres conceptos claves de la estructura de Cocoon. Éstos son:

Pipeline(o tubería)

La idea es dividir el procesamiento de un documento XML en varios pasos más elementales. Un *pipeline* consiste en una entrada seguida de un conjunto de procesos de tratado de la entrada y una salida. Realmente es un concepto muy sencillo pero a la vez muy potente y hace que la programación sea más fácil y más escalable.

Componentes del *Pipeline*

Son los que se encargan de llevar a cabo una tarea en particular en el *pipeline* como generar un documento XML, aplicar una transformación o producir una salida, entre otros. Estos componentes se pueden personalizar y pueden ser creados por el propio desarrollador.

Existen cuatro grupos generales de componentes. Éstos son:

Entradas del *Pipeline*

Son los generadores y los lectores (Ver Sección 3.1.1) .

Procesadores

Son los que llevan a cabo las transformaciones y las acciones (Ver Sección 3.1.1).

Salidas del *Pipeline*

Son los serializadores (Ver Sección 3.1.1).

Procesamiento Condicional

Es la parte encargada de hacer las selecciones y el proceso de *match* (Sección 7.1.1)

Atender una solicitud

Esto incluye una serie de pasos, como identificar de forma selectiva el *pipeline* correcto que debe atender la solicitud pedida, cerciorarse de que el *pipeline* se lleve a cabo y producir el resultado al cliente que hizo la solicitud.

3.1.1. Estructura

Estructuralmente hablando Cocoon está compuesto de:

Productores

Son los ficheros fuentes de donde proviene el XML. Estos pueden ser estáticos o dinámicos (es decir creados mediante XSP). La operación de un productor se basa en transformar los datos del fichero en eventos SAX.

Procesadores

Atrapan el XML de los productores para aplicarle diversos procesos, como por ejemplo hacer conectividad a una base de datos, aplicar transformaciones XSL a los documentos XML, convertir los XSP en clases Java, etc. Son el proceso principal del *Pipeline*. El más común es el transformador XSLT

Cuando de contenido dinámico se habla, entran las acciones, es decir, procesos que sólo se pueden llevar a cabo y de los que sólo se puede saber el resultado en tiempo de producción, tales como interacción con bases de datos, validaciones, envío de correo electrónico, etc.

Reactor

Es la central estructural. Extrae del XML del productor, las instrucciones para determinar qué procesadores actuarán en el documento.

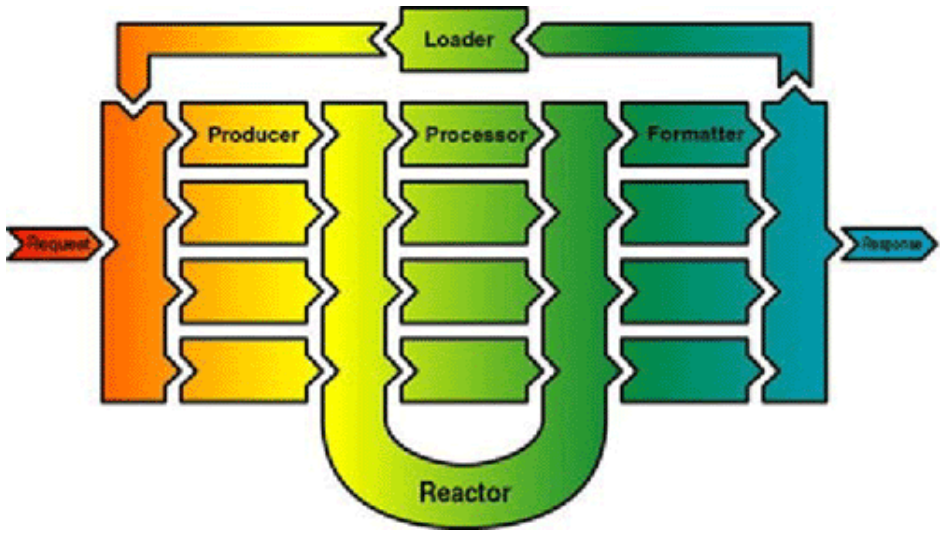
Formateadores

Son el punto final en un *Pipeline*. Recogen la representación interna del XML resultante (que está dada en eventos SAX) y la preparan para enviar como respuesta al cliente en el formato adecuado.

El formateador o serializador más común es el serializador XML que simplemente obtiene los eventos SAX y los lleva a un documento XML.

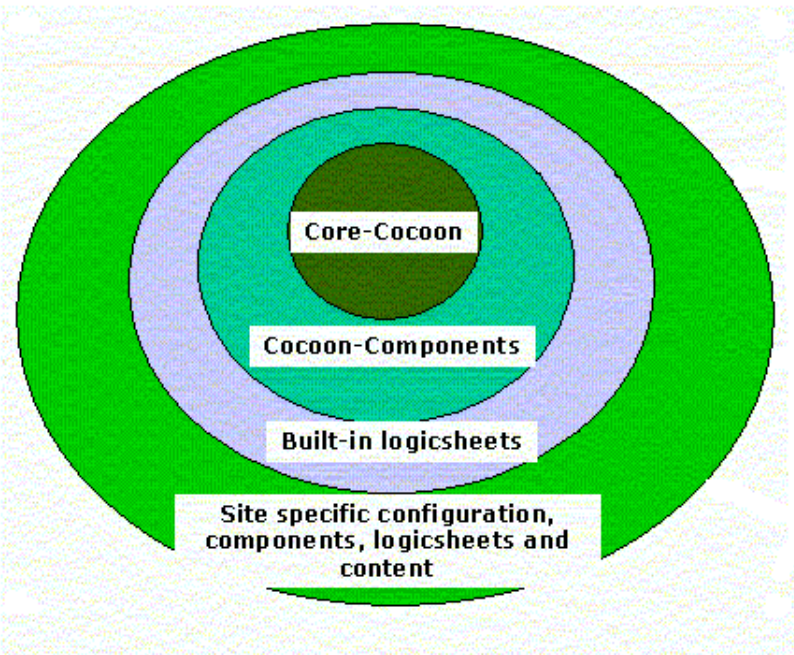
La anterior información se puede apreciar con el siguiente gráfico.

Figura 3-1. Cocoon desde un punto de vista estructural



3.1.2. Arquitectura

Figura 3-2. Arquitectura de Cocoon



Core Cocoon

Es el corazón de Cocoon. Encontramos un entorno para el control de sesiones, ficheros para configuración de Cocoon, para hacer manejo de contextos, aplicar mecanismos de caché, *Pipeline*, generación, compilación, carga y ejecución de programas.

Cocoon Components

En esta capa encontramos los generadores de XML, transformadores de XML, *matchers* de ficheros y serializadores para formatear los ficheros.

Built-in Logicsheets

Son hojas lógicas que necesita Cocoon para ficheros como `sitemap`, xsp, esql, *request*, *response*.

Site specific configuration, components, logicsheets and content

Es el nivel más externo en el cual un desarrollador puede hacer configuración, creación de componentes, creación de hojas lógicas y contenido definido por el usuario de Cocoon para su aplicación

Capítulo 4. Cocoon y las XSPs

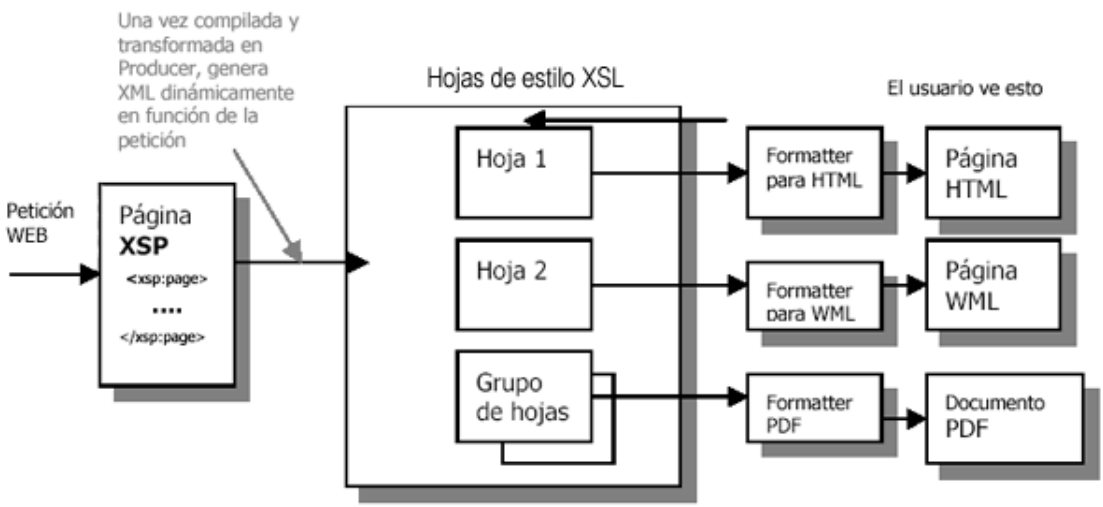
4.1. Introducción

Las XSPs manejan la misma idea de las JSPs, es decir, páginas de servidor, con lo cual se tiene dinamismo con posibilidad de conectividad a bases de datos y con las ventajas del XML.

Una XSP es simplemente un documento XML en donde se puede incluir contenido tanto estático como dinámico para generar XML de forma dinámica. Cabe anotar que el uso de *taglibs* se puede implementar sin problemas (es más, de manera más intuitiva que en JSP, ya que los *taglibs* son etiquetas XML) en las XSP, una evidencia adicional de las ventajas de esta nueva tecnología.

La siguiente gráfica muestra el flujo de operación en una solicitud XSP.

Figura 4-1. Flujo en XSP



4.2. Tipos de páginas XSP

De acuerdo a la forma como se programan, las XSPs se pueden dividir en tres grupos:

1. Con la lógica embebida en la presentación
2. Con hojas de estilos

3. Con bibliotecas etiquetas

4.2.1. XSP con la lógica embebida en la presentación

En este tipo de páginas se escribe el código en la propia página XML. Ésta es la práctica de programación menos recomendada y no debería utilizarla nunca, ya que aunque puede funcionar, el mantenimiento se torna muy complicado y la reutilización se minimiza brutalmente.

Para hacer el tratamiento de este tipo de XSP, el procesador XSP analiza el XML y convierte la página en un Servlet compilado. Esto lo hace llevando el XML a Java mediante un árbol DOM. Una vez llevado a código Java, se procesa y al resultado final se le aplica una transformación XSLT para llevar el resultado final a una página HTML.

Como podemos ver esta forma de programación, degrada el código XML ya que lo combina con el Java, por lo tanto la separación entre contenido y presentación de la cual hemos hablando no se hace presente. Este tipo de forma de programar no debería ser utilizada en ningún caso a menos que sea estrictamente necesario (aunque a decir verdad nunca debería ser estrictamente necesaria).

4.2.2. XSP con hojas de estilos

Esta forma de programar las XSP es mucho más recomendable que la anterior. En ésta, la página XSP original sería vista como un documento XML que se vale de hojas de estilos para aplicar la lógica de la programación, es decir el código Java. Cuando el documento XML original es procesado, se le aplica la transformación y como resultado se tiene una página XSP con el código embebido.

Note que en este caso el mantenimiento de la página mejora bastante con respecto al modelo que se expuso anteriormente, sin embargo la reutilización es muy pobre ya que el código fuente Java que se necesite para otra página XSP se debe incluir en otra XSL distinta.

4.2.3. XSP con bibliotecas de etiquetas

La idea de esta forma de implementar XSP es tener en bibliotecas especializadas, etiquetas que se encarguen de ejecutar cierto proceso, cierta función o procedimiento escrito en un lenguaje de programación (como por ejemplo Java) para que dichas bibliotecas puedan ser incluidas mediante espacios de nombres en los ficheros XML que las necesitan y así mismo se puedan utilizar las funciones que proveen dichas bibliotecas.

Estas bibliotecas deberían agruparse por roles o por tipos de funciones o servicios que proveen.

Como vemos en este caso el problema que aun teníamos, reutilización de código se vuelve imperceptible y no existe, ya que las bibliotecas y los servicios que proveen son independientes del fichero XML que las utiliza.

4.3. Conectividad a bases de datos

Con las XSP y Cocoon se puede tener acceso a una bases de datos de cualquier tipo, con lo cual usted puede tener la persistencia de su aplicación en un sistema manejador de bases de datos y aprovechar las ventajas tanto del manejador como de las XSP.

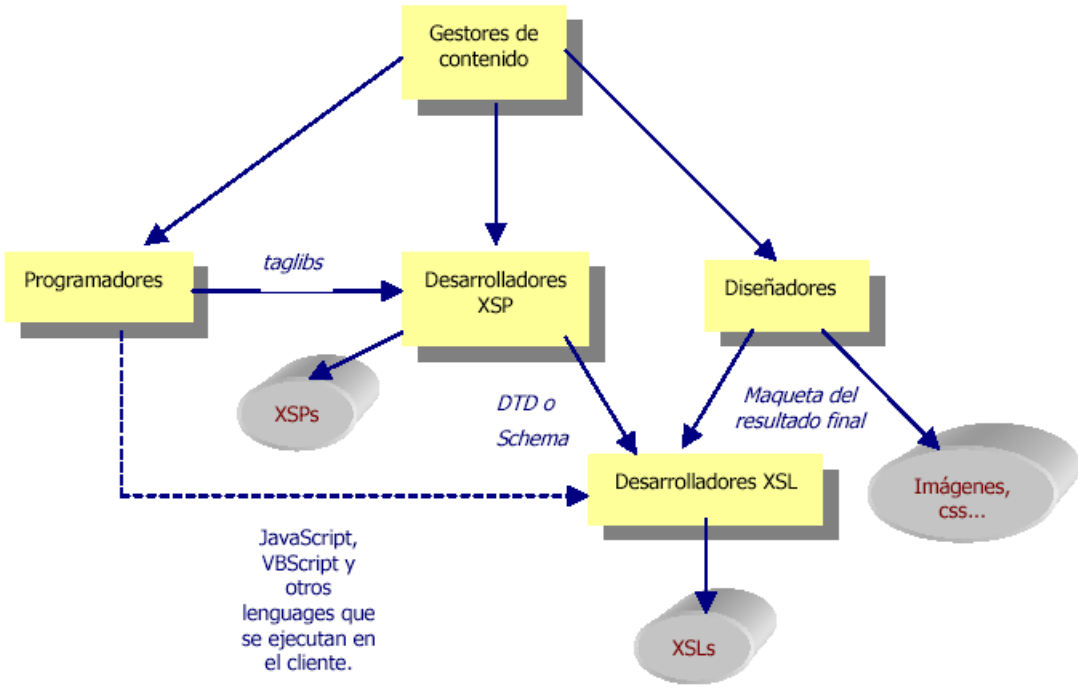
Para poder utilizar una XSP para conectarse a una base de datos, usted tiene que cargar el *driver* de su base de datos, definir las variables típicas de conexión a base de datos, tales como url, nombre de usuario, contraseña, etc.

Con XSP en Cocoon usted puede tener *prepared statements*, manejo de múltiples *resultsets* y *pool* de conexiones.

Nota: Para más información de XSP y acceso a Bases de Datos vaya a la Sección 8.2.2.

Capítulo 5. Paralelismo con Cocoon

Figura 5-1. WorkFlow en Cocoon



Como ya nos hemos podido dar cuenta el paralelismo con Cocoon se incrementa de forma enorme. Esto mejora tanto la calidad de los productos (ya que las personas se especializan en un área en particular) como los tiempos de desarrollo de trabajo y de respuesta de los mismos. Aquí se explica como se puede hacer *Workflow* con Cocoon. En éste se deben tener en cuenta estos 5 perfiles.

Gestores de Contenido

Estudian el tipo de contenido al cual se quieren o deben referir. Una vez que identifican los contenidos, notifican a los programadores para la construcción de *taglibs*. La especificación de contenidos la reciben también los desarrolladores XSP y los diseñadores.

Programadores

Proporcionan *taglibs* que al ejecutarse generan el contenido acordado con los gestores.

Desarrolladores XML

Estructuran los contenidos en los XMLs que crean según lo acordado con los gestores. Introducen contenido estático en las páginas y las etiquetas *taglibs* donde corresponda para el contenido dinámico.

Diseñadores

: Se encargan de construir el esquema de la interfaz. Generan entonces la forma de presentación de los datos de cada vista, de cada formato y cada uno de los elementos estéticos.

Desarrolladores XSL

: Se dedican a elaborar documentos XSL para obtener las vistas hechas por los diseñadores a partir de los XMLs hechos por los desarrolladores XML.

Capítulo 6. Instalación de Cocoon 2

En este apartado nos adentramos en un campo un poco más denso, explicando cómo hacer la instalación de Cocoon.

6.1. Requisitos para la instalación

Ya que Cocoon es una aplicación Web hecha en Java, debe ejecutarse sobre un motor de Servlets. Como estamos hablando de Java, Cocoon puede ser ejecutado sobre cualquier motor de Servlets. Para este caso en particular se ha utilizado Tomcat 4.0.3

6.1.1. Instalación de Tomcat

La instalación de Tomcat es realmente muy sencilla.

Lo primero que debe hacer es descargar el instalador. Ésto lo puede hacer desde este enlace (http://jakarta.apache.org/builds/jakarta-tomcat-4.0/release/) en el cuál encontrará la última versión de este producto de licencia libre. Para el momento en el que este documento estaba siendo elaborado la versión más reciente de Jakarta Tomcat era la 4.0.3 y ya existía una alfa para la versión 4.1.0

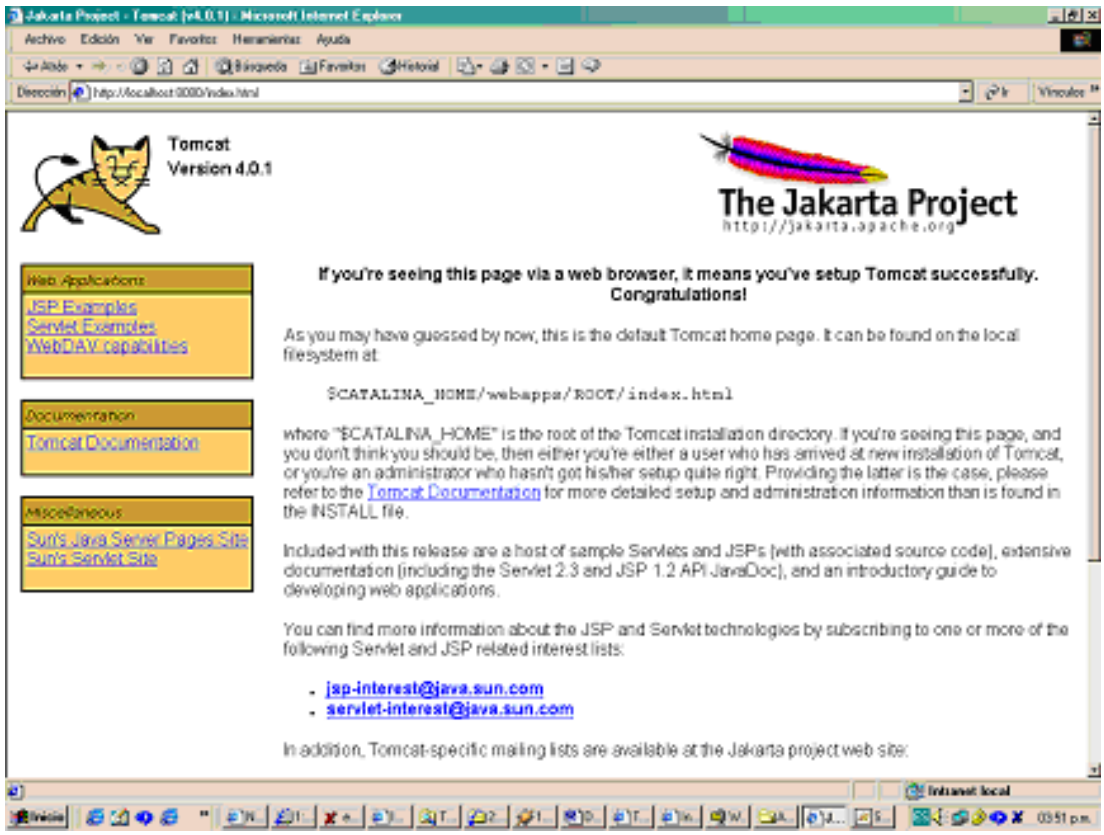
Una vez descargado el instalador, ejecútelo. Los pasos a seguir son bastante intuitivos y no presentan problema alguno.

En el directorio donde instaló Tomcat, es decir, donde está la raíz de la aplicación, lo llamaremos `CATALINA_HOME` (CATALINA es el nombre del contenedor de Servlets de Tomcat, el cuál tiene una implementación totalmente distinta desde la versión 4).

Para subir y bajar Tomcat vaya al directorio `CATALINA_HOME/bin`. Ahí encontrará dos *scripts* para llevar a cabo esta operación (*startup* y *shutdown* respectivamente).

Tomcat se ejecuta por omisión en el puerto 8080, así que una vez que haya arrancado Tomcat puede probar la instalación abriendo en el navegador la dirección *http://localhost:8080*. Si la instalación no tuvo problemas se le mostrará una página de bienvenida semejante a ésta:

Figura 6-1. Ventana de bienvenida de Tomcat



6.1.2. Ambiente Java

Para poder ejecutar tanto Tomcat como Cocoon usted necesita tener instalado el kit de desarrollo de Java el cual se encuentra actualmente en la versión 1.4.0 y puede ser descargado de forma gratuita desde este enlace (http://java.sun.com/j2se/).

6.2. Instalando Cocoon

6.2.1. Instalación Rápida De Cocoon

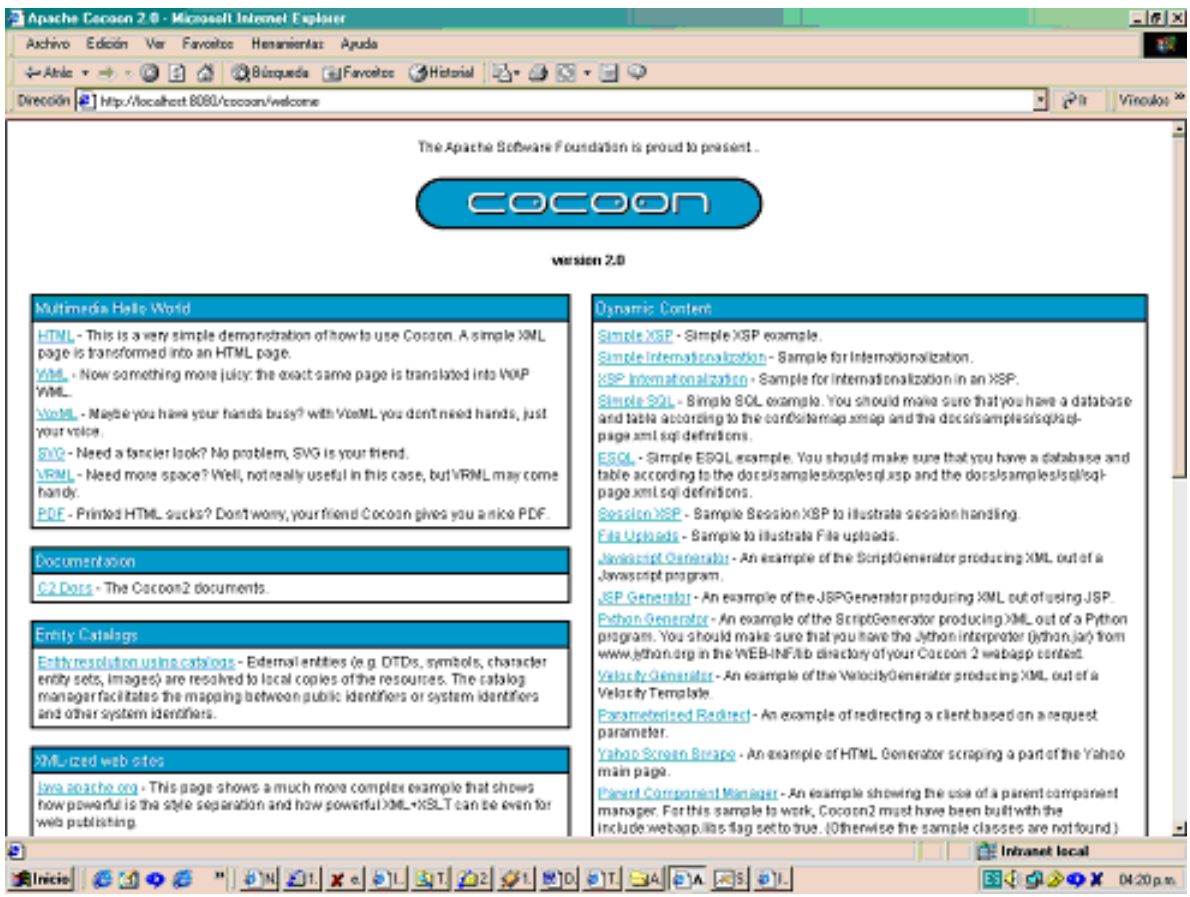
De Cocoon se pueden obtener dos distribuciones. La que trataremos en esta parte es la distribución en binario que puede ser descargada de este enlace (http://xml.apache.org/cocoon/dist/). Con esta distribución lo único que usted debe hacer es descargarla y descomprimirla en cualquier directorio. En el

directorio que usted eligió deberá haber quedado el fichero *cocoon.war*. Este fichero es el de la aplicación Cocoon.

Para que Tomcat y Cocoon se puedan comunicar, usted debe copiar el `cocoon.war` en el directorio `CATALINA_HOME/webapps` e iniciar Tomcat.

Cuando usted inicia Tomcat puede darse cuenta que el fichero es descomprimido automáticamente en el directorio `CATALINA_HOME/webapps/cocoon/`, el cual llamaremos de ahora en adelante COCOON_HOME. Para probar si cocoon está funcionando puede abrir la dirección *http://localhost:8080/cocoon/* en el *browser*, en la cual debe mostrársele una página de bienvenida de este estilo.

Figura 6-2. Ventana de bienvenida de Cocoon



6.2.2. Instalación a partir de los fuentes

En ocasiones es recomendable tener una copia local del código de Cocoon y compilar la aplicación de forma local. Para ésto, lo que usted debe hacer es descargar el código fuente de Cocoon. Esto lo puede realizar a través del servidor de CVS (Current Versioning System) de Apache.

Primero det todo, usted debe tener instalado CVS. Si usted no lo ha instalado aún en su máquina, puede consultar el sitio web de CVS (http://www.cvshome.org/) para más información.

En el momento que tenga instalado el CVS, ingrese al servidor de CVS de Apache de la siguiente forma:

```
$ cvs -d:pserver:anoncvs@cvs.apache.org:/home/cvspublic login
```

Cuando se le pregunte por una contraseña escriba *anoncvs*. Luego escriba lo siguiente:

```
$ cvs -d:pserver:anoncvs@cvs.apache.org:/home/cvspublic -z3 checkout -r cocoon_20_branch xm
```

Una vez hecho esto se inicia la descarga de todo el código necesario para la compilación de Cocoon.

Cuando tenga los fuentes de Cocoon descargados, debe compilarlos para crear el fichero *cocoon.war*. Para empezar a ejecutar el proceso de compilación utilice la siguiente instrucción:

```
$ ./build.sh -Dinclude.webapp.libs=yes webapp
```

Ésto creará un directorio con el código compilado, las bibliotecas y el fichero *cocoon.war*. Una vez termine el proceso copie el *cocoon.war* en el directorio `CATALINA_HOME/webapps` y reinicie Tomcat. De esta forma Cocoon estará ejecutándose en http://localhost:8080/cocoon/.

> **Sugerencia:** Si usted está interesado en hacer pruebas con Cocoon es útil crear una aplicación aparte para este fin. Ésto lo puede hacer creando un directorio nuevo bajo `CATALINA_HOME/webapps`. Supongamos que a dicho directorio se le pone como nombre `pruebasCocoon`. Lo que usted debe hacer es copiar el fichero `COCOON_HOME/cocoon.xconf` y la carpeta `COCOON_HOME/WEB-INF` en `CATALINA_HOME/webapps/pruebasCocoon/`. Ésto ya es suficiente para empezar a hacer sus pruebas y sus desarrollos ya que en `WEB-INF` están todas las clases necesarias para hacer que Cocoon pueda funcionar correctamente. Cree también su propio `sitemap` en `CATALINA_HOME/webapps/pruebasCocoon`(con lo cual no corre el riesgo de alterar los ejemplos y la documentación que ya existan) y cargue su aplicación en http://localhost:8080/pruebasCocoon/

Capítulo 7. Configuración y personalización en Cocoon

Cocoon cuenta con varios ficheros para hacer la configuración y personalización del mismo. Entre éstos, el más importante a nivel de usuario es el `sitemap.xml`. En este fichero se lleva a cabo el proceso de selección y *match*.

7.1. El *sitemap*

7.1.1. Selección y match en Cocoon

Casi todo *Pipeline* tiene secciones condicionales. Una sección condicional sirve para decirle a Cocoon qué tipo de solicitudes puede atender y cómo debe atenderlas.

Los *matcher* y los selectores desempeñan la misma función en Cocoon, condicionar un requerimiento como lo haría una instrucción **if** y analizar si la condición se cumple o no para poder llevar a cabo una tarea en particular. La diferencia entre un selector y un *matcher* radica que mientras el primero enumera todos los posibles valores, el segundo trabaja con expresiones regulares para evaluar la condición

7.1.2. Funcionalidad del *sitemap*

En el `sitemap` es en donde se lleva a cabo la parte Web de Cocoon. Éstee tiene dos funciones básicas:

- Declarar los componentes que serán utilizados por cualquier *pipeline*.
- Declarar los *pipelines* necesarios para el funcionamiento de las aplicaciones.

7.1.3. Estructura básica del `sitemap`

El `sitemap` puede encontrase en el directorio de la aplicación Web Cocoon, es decir es `COCOON_HOME/sitemap.xml`

El `sitemap` tiene tres partes básicas. La primera es la declaración del espacio de nombres, la segunda la declaración de los componentes y la tercera es la declaración de los *pipelines*. Un fichero `sitemap.xml` es entonces de este estilo:

Ejemplo 7-1. Ejemplo de un sitemap básico

```
<map:sitemap
  xmlns:map="http://apache.org/cocoon/sitemap/1.0">
```

```
  <map:components>
    <map:generators/>
    <map:readers/>
    <map:transformers/>
    <map:actions/>
    <map:serializers/>
    <map:matchers/>
    <map:selectors/>
  </map:components>

  <map:pipelines>
  </map:pipelines>

</map:sitemap>
```

Capítulo 8. Desarrollo en Cocoon

8.1. Contenido estático

Para poder implantar sus aplicaciones de contenido estático en Cocoon usted debe seguir varios pasos:

Vamos a suponer que usted tiene un fichero para presentar información acerca de su empresa y es la página inicial de la misma. En este caso ese fichero, por ser el inicial lo llamaremos `index`.

Esto quiere decir que deberá tener un fichero llamado `index.xml` (con su respectiva DTD, supongamos su nombre como `index.dtd`) en el cual tendrá la información necesaria para mostrar la página principal de la empresa y además deberá tener un fichero index.xsl con el cual se aplicará formato HTML al documento XML.

> **Sugerencia:** Es de anotar que no tiene porque crear un fichero XSL por cada fichero XML que tenga en su aplicación, sólo que para efectos de un ejemplo de muestra basta con hacerlo de esta forma.

El fichero `sitemap.xmap` de Cocoon nos servirá para decirle a Cocoon dónde encontrar los fuentes, como procesarlos y cómo presentarlos. Este fichero lo puede encontrar en `COCOON_HOME/`.

Lo que tiene que hacer es editar este fichero para añadirle un *pipeline* con el cual se pueda atender una solicitud que muestre el fichero que desea presentar.

Para nuestro ejemplo vamos a suponer que el fichero `index.xml` se encuentra en la ruta `$MiAplicacion/XML/`, que el fichero `index.dtd` se encuentra en la ruta `$MiAplicacion/DTD` y el fichero `index.xsl` que transforma el XML en un HTML se encuentra en la ruta `$MiAplicacion/XSL/HTML/`

Atención

Tenga en cuenta la ruta en la que guarda su DTD para que el fichero XML la pueda reconocer.

> **Sugerencia:** Es recomendable manejar rutas relativas en la declaración de la DTD para mejorar la portabilidad de la aplicación.

> **Sugerencia:** Cuando este construyendo aplicaciones en Cocoon es bastante útil definir directorios para guardar sus ficheros XML, XSL, sus DTD, sus fuentes, sus clases, etc.

Bien, el *pipeline* que usted debe añadir es de este estilo:

Ejemplo 8-1. Código para funcionamiento de un solicitud de un fichero XML presentado como un HTML

```
<map:match pattern="MiAplicacion/index.html">
 <map:generate type="file" src="$MiAplicacion/XML/index.xml"/>
 <map:transform src="$MiAplicacion/XSL/HTML/index.xsl"/>
</map:match>
```

Analicemos un poco más detalladamente esto. La línea **`match pattern="MiAplicacion/index.html"`** le indica a Cocoon que cuando llegue una solicitud del tipo **`http://localhost:8080/cocoon/MiAplicacion/index.html`** la atienda obteniendo los datos del fichero XML `$MiAplicacion/XML/index.xml` (esto se le indica mediante la línea **`generate type="file" src="$MiAplicacion/XML/index.xml"`**) y aplicándole la transformación dada por el fichero XSL ubicado en `$MiAplicacion/XSL/HTML/index.xsl` (lo cual se le dice mediante la línea **`transform src="$MiAplicacion/XSL/HTML/index.xsl"`**).

Nota: Para un ejemplo un poco más detallado consulte el Apéndice A.

8.2. Contenido Dinámico

8.2.1. Dando lógica con programación en Java

Atención

En Construcción

8.2.2. Acceso a bases de datos

Para acceder una base de datos usted debe tener en cuenta tres pasos:

1.

Configurar el *Data Source* para acceder la base datos.

Ésto lo debe hacer en al fichero `cocoon.xconf` añadiendo las siguientes líneas en la etiqueta **`datasources`**

Ejemplo 8-2. Código para definir un *Data Source* para acceso a una base de datos

```
<jdbc name="nombreBD">
  <pool-controller min="0" max="10"/>
  <dburl>urlParaConexionABaseDeDatos</dburl>
  <user>NombreUsuario</user>
  <password>contraseñaUsuario</password>
</jdbc>
```

Donde *`nombreBD`* es el nombre que se le dará al *Data Source*, *`NombreUsuario`* es el nombre de un usuario registrado en la Base de Datos con el cual se llevará a cabo la conexión y *`contraseñaUsuario`* es la contraseña del usuario *`NombreUsuario`* con la cual se validará dicho usuario

2.

Configurar el fichero `web.xml`

Para que cargue el *driver* e incluir el *driver* de tal forma que Cocoon tenga un lugar desde donde cargarlo.

Para configurar el `web.xml` con ayuda de la etiqueta `init-param` y la etiqueta hija de ésta, `param-name` con valor **`load-class`** enunciando dentro de esta última el nombre del *driver* y separando el nombre de los distintos *drivers* por coma o espacio. Por ejemplo, para incluir un *driver* para Oracle y otro para *IBM WebSphere* las líneas de código que deberían verse en el fichero `web.xml` serían:

Ejemplo 8-3. Código para cargar clases para acceso a bases de datos.

```
<init-param>
  <param-name>load-class</param-name>
  <param-value>

    <!-- Para Oracle: -->
    oracle.jdbc.driver.OracleDriver

    <!-- Para IBM WebSphere: -->
    com.ibm.servlet.classloader.Handler
```

```
<!-- For parent ComponentManager sample:
        org.apache.cocoon.samples.parentcm.Configurator
        -->
      </param-value>
    </init-param>
```

> **Nota:** Si usted está utilizando la Base de Datos que viene con Cocoon (hsql)este paso no es necesario

3. Si va a utilizar hsql debe añadir las instrucciones de base de datos que necesite su aplicación, tales como sentencias de autenticación, de creación de tablas, de inserciones de datos, etc. Esto lo debe hacer en el fichero `cocoondb.script` ubicado en la ruta COCOON_HOME/WEB-INF/db/

   Para nuestro caso se añadieron las siguientes líneas:

   Ejemplo 8-4. Ejemplo de Código de Base de Datos necesario a incluir con la Base de Datos hsql

```
CREATE USER usuario PASSWORD "contrasena" ADMIN
CONNECT USER usuario PASSWORD "contrasena"
CREATE TABLE PRUEBAS(ID INTEGER,NAME VARCHAR,UNIQUE(ID))
INSERT INTO PRUEBAS VALUES(1,'Prueba 1')
INSERT INTO PRUEBAS VALUES(1,'Prueba 2')
```

   con lo cual se está dando la posibilidad al usuario *`usuario`* con contraseña *`contrasena`* hacer operaciones sobre la tabla Pruebas, la cuál tiene 2 registros.

8.2.2.1. Etiquetas SQL y ESQL

Para la construcción de páginas XSP, contamos con dos tipos de etiquetas, SQL y ESQL.

La diferencia radica en que ESQL siendo más nuevo, presta mayores funcionalidades como combinar distintos tipos de hojas de estilos, soporte para *prepared statements* y manejo de varios *resultsets* en una sola sentencia, entre otras cosas. De ahí su nombre, *Extended* SQL.

A continuación presentaré dos ejemplos con estas tecnologías para analizar y tener en cuenta cómo funciona cada una.

8.2.2.1.1. Ejemplo con uso de etiqueta SQL

1. Añada un *pipeline* en el `sitemap` que sea de la forma:

Ejemplo 8-5. *Pipeline* necesario para una XSP con etiquetas SQL y acceso a una Base de Datos

```
<map:match pattern="MiXSP/MiEjemploXspSql">
 <map:generate src="$MiAplicacion/MiEjemploXspSql"/>
 <map:transform type="sql">
   <map:parameter name="use-connection" value="MiConexion"/>
 </map:transform>
 <map:transform src="stylesheets/simple-sql2html.xsl"/>
 <map:serialize/>
</map:match>
```

Nota: Para este caso, estamos indicando que el transformador es de tipo sql y que se debe usar una conexión llamada MiConexion. Es decir, estamos indicando desde el `sitemap` el nombre de la conexión

Teniendo en cuenta todo lo anteriormente expuesto, se pueden escribir páginas con etiquetas sql.

Ejemplo 8-6. Código de una XSP con conexión a Base de datos con etiqueta SQL

```
<?xml version="1.0"?>

<page xmlns:sql="http://apache.org/cocoon/SQL/2.0">

 <title>Una Prueba con SQL</title>
 <content>
  <para>Una página con SQL</para>

  <execute-query xmlns="http://apache.org/cocoon/SQL/2.0">
   <query>
select id, name from PRUEBAS
   </query>
   <execute-query>
    <query>
select id, name from PRUEBAS
    </query>
   </execute-query>
  </execute-query>

 </content>
</page>
```

8.2.2.1.2. Ejemplo con uso de etiqueta ESQL

1. Añada un *pipeline* en el `sitemap` que sea de la forma:

   Ejemplo 8-7. *Pipeline* necesario para una XSP con etiquetas ESQL y acceso a una Base de Datos

```
<map:match pattern="MiXSP/MiEjemploXspEsql">
 <map:generate type="serverpages" src="$MiAplicacion/MiEjemploXspEsql.xsp"/>
 <map:transform src="stylesheets/dynamic-page2html.xsl">
 </map:transform>
 <map:serialize/>
</map:match>
```

   Nota: Para este caso, estamos indicando que el generador es de tipo *serverpages*.

Teniendo en cuenta todo lo anteriormente expuesto, se pueden escribir páginas con etiquetas sql.

Ejemplo 8-8. Código de una XSP con conexión a Base de datos con etiqueta ESQL

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<xsp:page
  language="java"
  xmlns:xsp="http://apache.org/xsp"
  xmlns:esql="http://apache.org/cocoon/SQL/v2"
>

  <page>

   <title>Una prueba con ESQL</title>

   <content>

   <esql:connection>
     <esql:pool>MiConexion</esql:pool>
     <esql:execute-query>
       <esql:query>select * from PRUEBAS</esql:query>
       <esql:results>
 <esql:row-results>
   <para><esql:get-string column="name"/></para>
 </esql:row-results>
       </esql:results>
     </esql:execute-query>
```

```
   </esql:connection>

   </content>
  </page>
</xsp:page>
```

Nota: Note que en este caso, es en la página XSP en donde se define el nombre de la conexión.

Como usted ya se habrá podido dar cuenta, la diferencia en implementación entre ambas tecnologías es mínima. Dependiendo de las necesidades de su aplicación puede escojer entre ambas, teniendo en cuenta las potencialidades de ESQL y el desconocimiento que existe aún por su poco tiempo de vida en el mundo del *software*.

8.3. *Deployment* en Cocoon

8.3.1. Condiciones mínimas

Es muy común tener varias aplicaciones bajo Cocoon, en cuyo caso es recomendable tener ficheros de configuración aparte para el momento en el que se debe hacer *deployment* a cada aplicación. Esto mejora la portabilidad y la escalabilidad de los productos.

Para poder lograr esto Cocoon provee una herramienta poderosa, el concepto de *SubSitemap*.

Un *subsitemap* no es más que un fichero `sitemap` para una parte en particular de una aplicación de Cocoon.

Para poder utilizar esta técnica sólo se deben tener en cuenta dos cosas:

- Incluir en el `sitemap` general de Cocoon el `subsitemap` (y especificar a qué y en dónde se aplica ese `subsitemap`).
- Incluir el `subsitemap` en el lugar correcto.

Para esta parte voy a trabajar con el ejemplo de la sección referente a contenido estático desarrollada al inicio de este capítulo (ver).

Para esta aplicación vamos a construir entonces un `subsitemap` en el directorio `$MiAplicacion/`, es decir, el fichero quedará en la ruta `$MiAplicacion/sitemap.xmap`.

8.3.2. Inclusión de un *subsitemap* en el *sitemap* de Cocoon

En el fichero `sitemap.xmap` de Cocoon se deben añadir las siguientes líneas:

Ejemplo 8-9. Código para incluir un *subsitemap*

```
   <map:match pattern="MiAplicacion/*">
    <map:mount uri-prefix="MiAplicacion" check-reload="yes" src="$MiAplicacion/sitemap.xmap
reload-method="synchron"/>
   </map:match>
```

Bien, miremos un poco este código para comprenderlo mejor:

- En la línea *match pattern="MiAplicacion/*"* se le está indicando a Cocoon que cualquier recurso que intente ser accedido por el URL http://localhost:8080/cocoon/MiAplicacion/ debe ser atendido con el fichero ubicado en `$MiAplicacion/sitemap.xmap` (esto se le está indicando con el atributo *src* de la etiqueta `mount`, es decir, en la línea **`mount uri-prefix="MiAplicacion" check-reload="yes" src="$MiAplicacion/sitemap.xmap"`**)
- Con el atributo *uri-prefix* le estamos diciendo que cuando siga el flujo del requerimiento del usuario al *subsitemap* no le pase la cadena MiAplicacion. Esto es lógico, ya que para el `subsitemap` de MiAplicacion, todos los requerimientos que va a atender son de la aplicación MiAplicacion, por lo cual incluirlo en cada uno de los *pipelines* del `subsitemap` sería redundante.
- Con el atributo *check-reload* damos la opción de que Cocoon verifique cuando el *subsitemap* de MiAplicacion sea modificado para que lo vuelva a cargar. Si el valor del atributo es *yes*, chequea cada vez que sea modificado, si el valor es *no* sólo carga el *subsitemap* cada vez que sea cargado Cocoon.
- Por último, con el atributo *reload-method* le indicamos el modo como debe recargar el `subsitemap`. Si el valor es **`synchron`**, recarga el *subsitemap* cuando se le haga una solicitud y no muestra el resultado del requerimiento hasta que es recargado por completo, caso contrario a cuando el valor es **`asynchron`** ya que en este caso, aunque también recarga en el momento que se haga el requerimiento, deja mostrar el resultado del requerimiento mientras va recargando el fichero. Es de tener en cuenta aquí, que como el `subsitemap` no ha sido recargado en el momento que se muestra el resultado de la solicitud del usuario (cuando muestra el resultado empieza a ejecutar el proceso que lo recarga), el resultado mostrado no estará actualizado con respecto al estado actual de la aplicación, sólo hasta que sea pedido en la siguiente ocasión.

  Sugerencia: En ambientes de desarrollo es bastante útil tener la opción de que cada vez que se haga un cambio, éste se pueda reflejar de forma inmediata. Sin embargo en ambientes de producción es mejor tener configurado que los cambios se reflejen una vez el servicio se baje y se vuelva a restaurar; ésto es para no perjudicar a los usuarios de la aplicación quienes podrían tener la impresión de una aplicación lenta.

  Mejor aún si crea una copia de la aplicación, para tener una en producción y otra en desarrollo para hacer las pruebas. Para conocer como crear una aplicación en Cocoon consulte la sugerencia que está al final de la sección Sección 6.2.2

8.3.3. Código del `subsitemap`

El `subsitemap`, el cual debe estar ubicado como ya se dijo en la ruta `$MiAplicacion/` debe seguir el siguiente estilo:

Ejemplo 8-10. Código básico de un *subsitemap*

```
<?xml version="1.0"?>
<map:sitemap xmlns:map="http://apache.org/cocoon/sitemap/1.0">
<!-- ========================== Components ================================= -->
<map:components>
<map:generators default="file"/>
<map:transformers default="xslt"/>
<map:readers default="resource"/>
<map:serializers default="html"/>
<map:selectors default="browser"/>
<map:matchers default="wildcard"/>

</map:components>
<!-- ========================== Pipelines ================================= -->
<map:pipelines>
<map:pipeline>

                        <map:match pattern="index.html">
                               <map:generate type="file" src="$MiAplicacion/XML/index.xml"/
                               <map:transform type="xslt" src="$MiAplicacion/XSL/HTML/index
                        </map:match>

<map:handle-errors>
<map:transform src="../stylesheets/system/error2html.xsl"/>
<map:serialize status-code="500"/>
</map:handle-errors>
</map:pipeline>
</map:pipelines>
</map:sitemap>
```

Miremos un poco este `subsitemap`:

- En las líneas:

```
<!-- ========================== Components ================================= -->
<map:components>
<map:generators default="file"/>
<map:transformers default="xslt"/>
<map:readers default="resource"/>
<map:serializers default="html"/>
<map:selectors default="browser"/>
<map:matchers default="wildcard"/>
```

```
</map:components>
```

se están declarando los componentes del `subsitemap` y se está diciendo con el atributo *default* que para la aplicación en cuestión los componentes predeterminados son los que se declararon con el valor de dicho atributo en el `sitemap` general de Cocoon.

- Por otro lado, en las líneas:

```
<map:pipelines>
<map:pipeline>
                        <map:match pattern="index.html">
                               <map:generate type="file" src="$MiAplicacion/XML/index.xml
                               <map:transform src="$MiAplicacion/XSL/HTML/index.xsl"/>
                        </map:match>

<map:handle-errors>
<map:transform src="../stylesheets/system/error2html.xsl"/>
<map:serialize status-code="500"/>
</map:handle-errors>
</map:pipeline>
</map:pipelines>
```

se están declarando los *pipelines*. Esto se hace de igual forma que como se hace en el *sitemap* general de Cocoon.

> **Nota:** Fíjese que en la línea
>
> *<map:match pattern="index.html">*
>
> se está diciendo que si se hace una solicitud de la página `index.html`, tome los datos del documento `index.xml` y le aplique la transformación dada en `index.xsl`. Lo importante aquí es observar que esta página será mostrada cuando se cargue la dirección http://localhost:8080/cocoon/MiAplicacion/index.html ya que el *subsitemap* esta dentro de `$MiAplicacion` y en el `sitemap` general se dijo que la cadena **`$MiAplicacion`** sería truncada.

Apéndice A. Formato de reunión semanal

A.1. Introducción

En el presente documento se introducirán algunos de los términos comunes en la terminología XML y para ello se utilizará como ejemplo, unos formatos de reuniones semanales de Ingeniería de Software.

El formato de reunión semanal nació como una necesidad de registrar información necesaria para llevar un registro y un historial de reuniones de forma estructurada y con unos parámetros y lineamientos identificados ya de antemano.

Este formato se implementó como un documento XML para poder tenerlo de forma estructurada, aprovechar las ventajas propias del XML y conocer un poco esta tecnología.

Para las personas que no estén familiarizadas con la tecnología XML, en este documento se da una breve descripción de lo que es una DTD (Sección A.3.2.1), de lo que es un documento XML (Sección A.3) y de lo que es una XSL (Sección A.4) junto con un ejemplo de cada una de estas definiciones aplicadas al formato que tenemos como tema principal.

> **Importante:** Este documento sólo pretende ser un pequeño "abrebocas" de lo que son estas tecnologías punta. Es muy básico y la explicación está pensada para personas con muy pocos conocimientos.

A.2. Descripción formato de reunión semanal

En el formato de reunión semanal se pueden identificar tres componentes: *información general de la reunión*, *agenda de la reunión* e *informes por rol*.

Cabe anotar que mientras la información general es de carácter obligatorio, la agenda y los informes por rol pueden o no aparecer en el documento, pero si aparecen será sólo una vez.

A continuación se describen estos tres elementos

A.2.1. Elementos del formato de reunión semanal

A.2.1.1. Información General

La información general en el formato es de carácter obligatorio. En éste se registra el sitio o lugar en donde se hizo la reunión, la fecha, la hora en la cual empezó la reunión en cuestión, la hora en que finalizó la misma, el secretario de la reunión quien es el que se encarga de registrar la información en el momento de la reunión y los asistentes de la reunión.

A excepción del asistente, los demás items descritos son obligatorios y van de forma única. Los asistentes aunque es un dato obligatorio, tienen la particularidad de que pueden ser uno o varios.

A.2.1.2. Agenda

Como ya se dijo la agenda en el formato es de carácter opcional, pero sólo puede aparecer una vez. En ésta se registran los aspectos tratados contando por cada uno su respectiva descripción y sus propuestas. En cada reunión se deben tratar uno o más aspectos y por cada uno se pueden o no hacer varias propuestas. Una propuesta puede ser aprobada y puede tener una decisión.

A.2.1.3. Informes Por Rol

Esta parte es de carácter opcional y en el caso de que aparezca sólo puede hacerlo una vez. Aquí se almacenan las anotaciones que puede hacer el líder del proyecto o el personal de calidad, el de planificación, soporte o desarrollo. También se registra el rol de esa persona.

A.3. XML

A.3.1. ¿Qué es?

No es más que un conjunto de reglas para definir etiquetas semánticas para organizar un documento.

La tecnología XML es realmente sencilla y tiene alrededor otras tecnologías que la complementan y la hacen más grande y con posibilidades mayores. Entre estas se encuentran XSL y XSP.

A.3.2. Ejemplo XML con el formato de reunión semanal de Ingeniería de Software

A.3.2.1. DTD del formato de reunión semanal

A.3.2.1.1. ¿Que es una DTD?

DTD es la sigla de *Document Type Definition (Definición de Tipo de Documento)*.

Es un fichero lógico que contiene la definición formal de un tipo de documento en particular

En este se describen los nombres de los elementos, donde pueden aparecer y la interrelación entre ellos. Con éste se puede validar el documento XML.

A.3.2.1.2. DTD del formato de reunión semanal

El fuente de esta DTD la puede descargar desde este enlace (../fuentes/reunionSemanal.dtd.txt).

Ejemplo A-1. Ejemplo de una DTD para el formato de reunión semanal

```
<!ELEMENT reunionSemanal (informacionGeneral, agenda?, informesPorRol?)>

<!ELEMENT informacionGeneral (sitio, fecha, horaInicio, horaFin, secretario, asistente+)>
<!ELEMENT sitio (#PCDATA)>
<!ELEMENT fecha (#PCDATA)>
<!ELEMENT horaInicio (#PCDATA)>
<!ELEMENT horaFin (#PCDATA)>
<!ELEMENT secretario (#PCDATA)>
<!ELEMENT asistente (#PCDATA)>

<!ELEMENT agenda (aspectos)>
<!ELEMENT aspectos (aspecto+)>
<!ELEMENT aspecto (descripcion, propuestas?)>
<!ELEMENT descripcion (#PCDATA)>
<!ELEMENT propuestas (propuesta+)>
<!ELEMENT propuesta (textoPropuesta, decision?)>
<!ATTLIST propuesta aprobado (Si | No) "No">
<!ELEMENT textoPropuesta (#PCDATA)>
<!ELEMENT decision (#PCDATA)>

<!ELEMENT informesPorRol (informePorRol+)>
<!ELEMENT informePorRol (#PCDATA)>
<!ATTLIST informePorRol rol (lider | calidad | planeacion | soporte | desarrollo) #REQUIRED
```

A.3.2.2. XML del formato de reunión semanal

Aquí podemos apreciar un ejemplo de un posible documento XML del formato de reunión semanal.

Puede obtener el fuente de este ejemplo haciendo click aquí (../fuentes/reunionSemanal.xml.txt).

Ejemplo A-2. Ejemplo de un documento XML para el formato de reunión semanal

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE reunionSemanal SYSTEM "reunionSemanal.dtd">
<reunionSemanal>
  <informacionGeneral>
 <sitio> Oficina W302 </sitio>
 <fecha> 27 de Febrero de 2002 </fecha>
 <horaInicio> 4:00 pm </horaInicio>
 <horaFin> 5:32 pm </horaFin>
 <secretario> Juan Torres </secretario>
 <asistente> Saúl Zarrate Cardenas </asistente>
 <asistente> Juan Pablo Quiroga </asistente>
 <asistente> Jaime Irving Dávila </asistente>
  </informacionGeneral>
  <agenda> <aspectos>
 <aspecto>
<descripcion> Practicar DTD, XML </descripcion>
<propuestas>
  <propuesta aprobado="No">

 <textoPropuesta> Desarrollar un ejemplo con el Formato de Lanzamiento
de Ingenieria de Software </textoPropuesta>
 <decision> Como ya se presentó el Formato de
     de Lanzamiento de Ingenieria de Software,
no hay que documentarlo </decision>
  </propuesta>
  <propuesta aprobado="Si">
 <textoPropuesta> Desarrollar un ejemplo con el Formato de Reunión
Semanal de Ingenieria de Software </textoPropuesta>
  </propuesta>
</propuestas>
 </aspecto>
 <aspecto>
<descripcion> Practicar XSL </descripcion>
<propuestas>
  <propuesta aprobado="Si">
 <textoPropuesta> Desarrollar un ejemplo con el XML del Formato de
Reunión Semanal de Ingenieria de
     Software para presentación en html
</textoPropuesta>
  </propuesta>
</propuestas>
 </aspecto>
 <aspecto>
```

```
<descripcion> Practicar DocBook </descripcion>
<propuestas>
  <propuesta aprobado="Si">
 <textoPropuesta> Documentar el desarrollo del XML y el XSL del
Formato de Reunión de Ingenieria de
     Software </textoPropuesta>
  </propuesta>
</propuestas>
 </aspecto> </aspectos>
  </agenda>
  <informesPorRol>

 <informePorRol rol="planeacion">
Se cumplió con las expectativas de las metas planeadas
 </informePorRol>
 <informePorRol rol="soporte">
Se dieron vinculos de Internet para poder tener ayudas en el desarrollo
de los temas
 </informePorRol>
 <informePorRol rol="desarrollo">
El trabajo acordado es interesante y cumple con lo buscado
 </informePorRol>

  </informesPorRol>

</reunionSemanal>
```

A.4. XSL

XSL es el acrónimo de *eXtensible Style Language (Lenguaje de Estilo eXtensible)*.

A.4.1. Algunos aspectos de XSL

A.4.1.1. ¿Qué es XSL?

Es una Tecnología XML de hojas de estilos que sirve para mostrar documentos XML, es decir, darles formato de presentación.

A.4.1.2. ¿Para qué sirve?

La tecnología XSL sirve para transformar documentos XML en otros XML. Éste, permite la manipulación de la información XML. *XSLT XSL Transformation*.

También sirve para definir cómo acceder cierto punto de la estructura de un documento. XPath.

Por otro lado, tiene la capacidad de definir el formato que deben tomar los objetos dentro de un documento XML. *XSLFXSL Format*.

A.4.1.3. ¿De qué se trata?

En un documento XSL se describe un conjunto de reglas para aplicarse en documentos XML, reglas encaminadas a la presentación del documento XML.

A.4.1.4. Ejemplo de XSL con el formato de reunión semanal de Ingeniería de Software para salida en HTML

Para obtener el fuente XSL puede hacer click en este enlace (../fuentes/reunionSemanal.xsl.txt).

Ejemplo A-3. Ejemplo de una XSL para el formato de reunión semanal

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/reunionSemanal">
<HTML>
<HEAD>
<TITLE>
Acta <xsl:apply-templates
select="informacionGeneral/fecha"/>
</TITLE>
</HEAD>
<BODY text="#000000" vLink="#840084"
aLink="#0000ff" link="#0000ff"
bgColor="#ffffff">
<B>
<font size="6">
Acta
<xsl:apply-templates
select="informacionGeneral/fecha"/>
</font>
<BR></BR>
<BR></BR>
<font size="4">
<xsl:apply-templates
select="informacionGeneral/secretario"/>
</font>
<BR></BR>
<BR></BR>
<HR></HR>
<BR></BR>
```

```
<BR></BR>
Tabla de Contenido
</B>
<BR></BR>
<UL>
<A href="#items">Items a discutir</A>
<BR></BR>
<A href="#decisionesTomadas">Decisiones tomadas</A>
<BR></BR>
<A href="#reportesDeRol">Reportes de rol</A>
</UL>
<HR></HR>
<BR></BR>
<B>
<FONT SIZE="6">
<A name="#items">Items a discutir</A>
</FONT>
</B>

<BR></BR>
<xsl:apply-templates select="agenda/aspectos"/>

<HR></HR>
<B>
<FONT SIZE="6">
<BR></BR>
<A
name="#decisionesTomadas">Decisiones
Tomadas</A>
</FONT>
</B>

<BR></BR>

<xsl:apply-templates
select="agenda/aspectos/aspecto/propuestas/propuesta"/>


<HR></HR>
<B>
<FONT SIZE="6">
<BR></BR>
<A name="#reportesDeRol">Reportes de rol</A>
</FONT>
</B>

<xsl:apply-templates select="informesPorRol"/>

</BODY>
</HTML>
</xsl:template>

<xsl:template match="fecha">
```

```
<xsl:value-of select='.'/>
</xsl:template>

<xsl:template match="secretario">
<xsl:value-of select='.'/>
</xsl:template>

<xsl:template match="aspectos">
<xsl:apply-templates select="aspecto"/>
</xsl:template>

<xsl:template match="aspecto">
<UL>
<LI>
<xsl:value-of select="descripcion"/>
</LI>
</UL>
</xsl:template>

<xsl:template match="propuesta">
<xsl:choose>
<xsl:when test='decision!=""'>
<UL>
<LI>
<xsl:value-of select="decision"/>
</LI>
</UL>
</xsl:when>
</xsl:choose>
</xsl:template>

<xsl:template match="informesPorRol">
<xsl:apply-templates select="informePorRol"/>
<BR>
</BR>
</xsl:template>

<xsl:template match="informePorRol">
<UL>
<LI>
Reporte de
<xsl:value-of select="@rol"/>:
<xsl:value-of select='.'/>
</LI>
</UL>
</xsl:template>

</xsl:stylesheet>
```

A.5. Usando el formato de reunión semanal en Cocoon

Para usar el formato de reunión semanal en Cocoon, usted básicamente tiene que seguir los siguiente pasos:

1. Guardar el fichero XML y su DTD en el directorio que usted escoja. Para nuestro ejemplo lo haremos en la ruta $ReunionSemanalHome/XML.
2. Guardar el fichero XSL para la transformación del documento XML en uno HTML en cualquier carpeta. Para nuestro ejemplo lo haremos en la ruta $ReunionSemanalHome/XSL/HTML.
3. En el fichero *sitemap.xml* de Cocoon añada un *Pipeline* de este estilo:

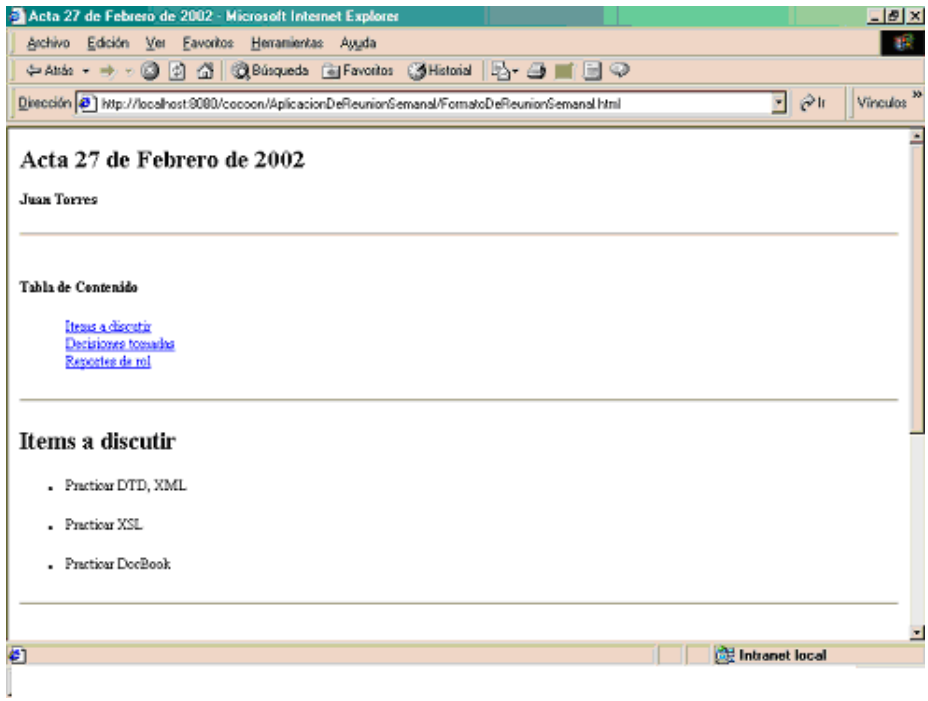
   Ejemplo A-4. Código para añadir un *pipeline* que cargue el formato de reunión semanal

   ```
   <map:match pattern="MiAplicacion/FormatoDeReunionSemanal.html">
     <map:generate src="$ReunionSemanalHome/XML"/>
     <map:serialize type="xml"/>
   </map:match>
   ```

4. Por último inicie su servidor de Servlets y cargue el documento. Para el caso de Apache Tomcat puede cargar el documento en http://localhost:8080/cocoon/MiAplicacion/FormatoDeReunionSemanal.html.

   Debería cargarle algo como:

Figura A-1. Formato de reunión semanal en Cocoon



Atención

Tenga en cuenta la ruta en la que guarda su DTD para que el fichero XML la pueda reconocer.

Sugerencia: Es recomendable manejar rutas relativas en la declaración de la DTD para mejorar portabilidad de la aplicación.

Sugerencia: Cuando esté construyendo aplicaciones en Cocoon es bastante útil definir directorios para guardar sus ficheros XML, XSL, sus DTD, sus fuentes, sus clases, etc.