tecnimap Murcia 2004

18

UNA APLICACIÓN DEL ESTÁNDAR XML. SISTEMA DE IMPRESIÓN JAVA EN SERVIDOR DE APLICACIONES WEB Y SERVIDOR UNIX

Ana Cerdeira Gutiérrez

Jefa de Servicio Centro Proceso de Datos en la Subdirección General de Informática y Estadística del Inem.
Jefa del proyecto Comunicación de la Contratación por Internet, CONTRAT@
Ministerio de Trabajo y Asuntos Sociales

RESUMEN

En esta comunicación se pretende exponer un sistema de impresión portable para cualquier máquina virtual java, que por sí misma supone un estándar en desarrollo de aplicaciones web J2EE.

Se han utilizado, para la arquitectura del sistema de impresión java, patrones de diseño estándar y la definición de los datos a imprimir se realiza con el lenguaje estándar eXtensible Markup Language (XML).

INTRODUCCIÓN

El INEM se ha planteado la revisión de la arquitectura de sus aplicaciones corporativas que se.

basan en una arquitectura server-appc, con base de datos centralizada y descentralización de procesos en equipos unix. La arquitectura a la que pretende migrar va dirigida a sutituir los equipos unix por servidores de aplicaciones web manteniendo los ordenadores personales, pero en lugar de estar conectados a los equipos unix serán clientes del servidor de aplicaciones, tanto a traves de internet como de la intranet corporativa, manteniendo la base de datos corporativa en el mainframe.

La definición del diseño de las aplicaciones presentes y futuras, presta especial énfasis al establecimiento de unos estándares de calidad en todos sus desarrollos de software así como a la definición de una arquitectura adecuada para sus aplicaciones web J2EE. Siguiendo la pauta de definición de estándares se ha diseñado el Sistema de Impresión en java que será objeto de este documento.

Se comienza con una breve exposición de la arquitectura del sistema de impresion, centrándose en primer lugar en los objetivos de diseño y como se han dispuesto los distintos componentes de software para cumplir los mismos. A continuacion se discutirá el patron de diseño MVC (Modelo Vista Controlador) , en el que se ha basado el diseño de esta solución, continuando con un examen funcional de los distintos modulos y capas en los que se divide. Al ser un examen de arquitectura, se omiten menciones al código fuente.

Se debe destacar en este sistema de impresión la utilización de datos en formato XML (eXtensible Markup Language), definidos mediante DTD o esquemas, así como, la existencia de un módulo de generación de impresiones que trata en modo nativo los documentos XML a imprimir.

El modelo de impresión java que se va a exponer responde a la necesidad de crear múltiples tipos de salida de datos, pdf, xml, fdf, post script, txt, etc de la forma mas eficiente posible y de acuerdo a unos patrones estándares. Los distintos tipos de salida se pueden implementar tanto en Servidor de Aplicaciones Web cómo en servidores de cualquier sistema operativo que tenga una Máquina Virtual Java 1.3 o superior, esto hace que sea un sistema versátil y de fácil mantenimiento.

El INEM ha aplicado este sistema de impresión en una aplicación web, CONTRAT@ que está operativa en la página principal www.inem.es y que se puede enlazar desde las páginas de las Comunidades Autónomas que lo deseen, esta aplicación se ejecuta en un Servidor de Aplicaciones Web. Tambien está en fase de desarrollo la aplicación de este sistema de impresión a algunos documentos que se obtienen desde aplicaciones modo carácter cuyo servidor es un equipo unix con máquina virtual java, con el fin de mejorar y modernizar la calidad de la impresión.

DISEÑO DEL SISTEMA DE IMPRESIÓN SIGUIENDO LOS PRINCIPALES PATRONES

Unas notas de historia

El origen de los Patrones de Diseño data de finales de los 70. Un arquitecto, Christopher Alexander, escribio dos tratados, *Un lenguage de patrones y La forma intemporal de construccion de edificios,* que

fundaron las bases descriptivas de lo que hoy en dia en informática se conoce como Patrones de Diseño, mediante ejemplos, explicaba las soluciones mas eficientes que se habían dado a un conjunto de problemas a lo largo del tiempo.

A mediados de los 90, la "banda de los cuatro": Gamma, Helm, Johnson y Vlissides publicaron el famoso texto *Patrones de Diseño: elementos de un software orientado a objetos reutilizable*. Estudiaron un conjunto de problemas que se presentaban en las aplicaciones orientadas a objeto y las soluciones mas racionales que se podían aplicar en ese contexto.

De esta forma, proporcionaron a la comunidad informática un lenguage común ademas de toda una batería de construcciones para dar respuesta a la mayoría de sus problemas. Siguiendo esta línea, la arquitectura debe usar, en la medida de lo posible, estos Patrones de manera extensiva.

Arquitectura del sistema de impresión JAVA

En la actualidad la tendencia mas aceptada es la utilización de patrones de diseño de arquitectura que dividen la responsabilidad en distintas capas que interaccionan unas con otras a través de sus interfaces. Se trata de los sistemas denominados multicapas.

La arquitectura elegida para el sistema de impresión java es un desarrollo multicapa en el que se han empleado principalmente dos patrones de diseño, el patrón Modelo Vista Controlador (MVC) y el patrón Delegación de Negocio. Tambien se han empleado otros patrones a los que se hará mención mas adelante pero su uso no ha determinado, a diferencia de los citados, la solución final de la arquitectura de impresión java.

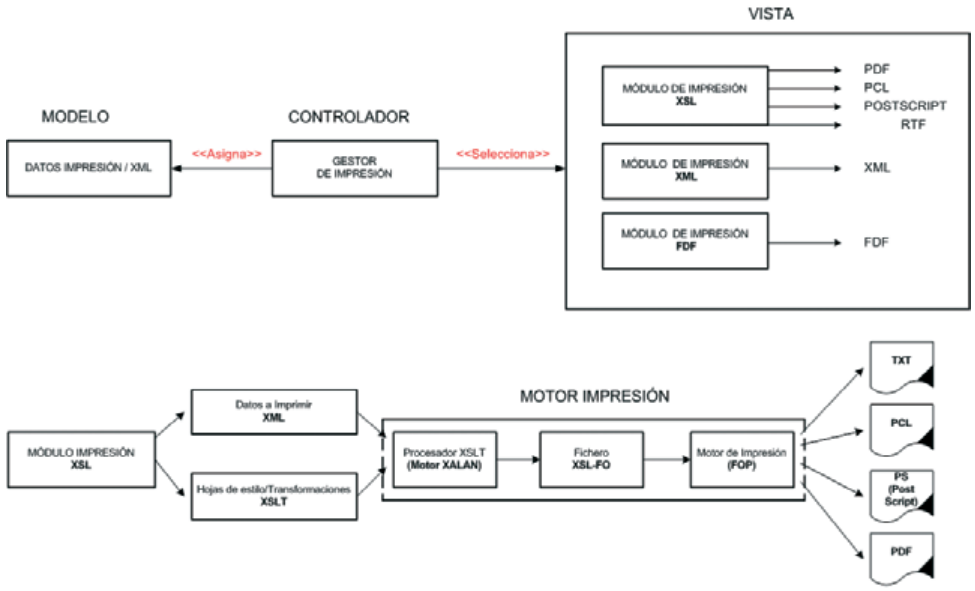

Figura 1. Sistema de impresión Java

Patrón Modelo-Vista-Controlador (MVC)

El patrón MVC separa tres formas distintas de funcionalidad:

- El Modelo que representa las estructuras de datos, o mas adecuado en el lenguaje de diseño orientado a objetos, los objetos de datos que entiende el sistema.
- La Vista ó vistas, son el conjunto de las diferentes salidas que pueden generarse, en nuestro caso serán las distintas formas de impresión.
- El Controlador, interpreta, es decir, ejecuta las peticiones de impresión del cliente, encargándose de seleccionar y encontrar los módulos de impresión oportunos, ya sea en funcion del tipo de datos a imprimir, en función de la petición del cliente o en función de ambas cosas.

Patrón Delegación de Negocio

Aísla al cliente de la problemática del manejo de los módulos de impresión. En aplicaciones distribuídas, la búsqueda y el tratamiento de excepciones en componentes remotos es complejo, un patrón de delegación oculta el manejo de este tipo de componentes al cliente por medio de clases intermedias que facilitan su tratamiento. La complejidad de este tipo de tratamiento se delega al Controlador.

DESCRIPCIÓN DE LAS DIFERENTES FUNCIONALIDADES EN EL PATRÓN MVC

Capa Modelo

El componente *Modelo,* dentro de la arquitectura *MVC* del sistema de impresión, encapsula los objetos de negocio y su API. Estos objetos, los datos a imprimir, se implementan a traves de Beans, se descarto el uso de *Entity Beans,* tanto *CMP* como *BMP,* para evitar problemas de rendimiento.

En el diseño de la capa *Modelo* se tuvieron en cuenta los siguientes puntos:

Conseguir un interfaz de datos manejable y coherente.

A medida que las aplicaciones *J2EE* crecen, tambien lo hace el número de objetos de negocio que manejan. Estos objetos a su vez van a interrelacionarse con otros objetos y componentes de los diferentes subsistemas. Debería evitarse un crecimiento ininterrumpido de las APIs, en concordancia con los principios generales de diseño.

La capa *Modelo* del sistema de impresión ha definido un solo interfaz, *DatosImpresion,* que debe implementar todos los objetos de negocio de impresión. Implementa las funcionalidades mínimas exigibles. Establece una serie de reglas, muy pocas y deja otras intencionadamente poco explícitas para permitir a los desarrolladores de las clases que lo implementen mayor libertad a la hora de ordenar sus datos.

Se pretende mantener un API mínimo y cualquier adición, en forma de nuevas funcionalidades, se delegará a las implementaciones de las mismas. Seguir un estilo consistente en el diseño de las APIs. Se han definido siguiendo el estilo e implementando la mayoria de sus métodos de un API conocida por todos y estándar en Java: las clases *Collection*. Cualquier programador que conozca el framework de *Collection* comprenderá el funcionamiento de *DatosImpresion* rápidamente.

Capa Controlador

El *componente Controlador* dentro de la arquitectura MVC gestiona el flujo de la aplicación y establece el vínculo entre la capa *Vista y Modelo,* ejecutanto la lógica de negocio en el *Modelo* a petición del cliente del sistema y ayudando en el proceso de selección de la *Vista*.

El *Controlador* separa la presentación de los datos de la lógica y objetos de negocio. Es responsabilidad del cliente la instanciación del conjunto de datos a imprimir (los objetos que implementan el interfaz *DatosImpresion*), mientras se delega al *Controlador* la eleccción y recuperación del modulo de impresión adecuado según el patrón *Delegacion de Negocio* y en último lugar el módulo de impresión creará la correspondiente *Vista*.

El componente principal de esta capa es la clase GestorImpresion, el nexo de unión entre las tres capas. GestorImpresion implementa el patrón de diseño Delegacion de Negocio en orden a facilitar al cliente el manejo y acceso a los diferentes Modulos de Impresion, se explicará en detalle más adelante. El contrato de diseño para esta capa establece:

Esta capa devuelve al cliente un objeto que implementa el interfaz ModuloImpresion. Las instancias de tipo Modulo Impresion implementan los métodos necesarios para generar impresiones. El tipo en tiempo de ejecución del objeto es transaparente para el cliente, que solamente maneja los metodos expuestos en el interfaz.

Responsabilidades del cliente: En orden a obtener impresiones, el cliente obtendrá objetos de tipo ModuloImpresion a traves de la clase GestorImpresion indicando el tipo de impresión deseado de acuerdo a las constantes enumeradas en el interfaz ModuloImpresion y proporcionando un objeto de tipo DatosImpresion o documentos XML.

El cliente no debe intentar encontrar o manejar directamente los Modulos de Impresion, es decir, las clases que implementan esta funcionalidad. Esta capa se encarga de seleccionar el módulo mas adecuado.

La capa Controlador actuando como Delegado de Negocio y resto de patrones de diseño empleados

De acuerdo a lo expuesto anteriormente, el cliente obtiene una referencia a un objeto que le va a permitir realizar las salidas de impresion. Aunque en la descripción de la capa *Vista* se abordará en mayor detalle, si podemos adelantar que la referencia que se le devuelve al cliente desde aquí no tiene por que ser la clase que realmente implementa toda la lógica de negocio del sistema de impresión, pudiendo ser incluso una simple clase accesoria. Este mecanismo de actuación es el descrito por el patrón de diseño "*Delegacion de Negocio*".

Los desarrolladores de código Java estándar que necesiten acceder a los servicios de impresión unicamente deben realizar sus peticiones de impresión a traves del GestorImpresion en esta capa.

GestorImpresion, por su parte, se ocupar de:

- Realizar la búsqueda del módulo de impresión adecuado. Si la funcionalidad está implementada en clases Java, la complejidad es nula, basta crear y devolver una nueva instancia del módulo correspondiente. Si por el contrario el módulo se hubiera implementado en un EJB la mecánica sería diferente. La búsqueda del componente y su creación se haría siguiendo el patrón *Localizador de Servicio* que gestiona las creaciones de Contextos JNDI, la búsqueda de componentes y almacena en caché los componentes Home.

Se pretende aislar la complejidad de las busquedas, ahorrar recursos en la creación de nuevos Contextos así como en la recuperación de los interfaces Home en los EJBs. El patrón *Localizador de Servicio* centraliza estas tareas, proporcionando una caché unificada para la creación de EJB y búsqueda en directorios JNDI así como el almacenamiento de las dierentes variables de entornos.

Si se presentara el caso, tambien ocultaría las particularidades de busquedas de objetos distribuídos en diferentes directorios de nombres. Tambien podemos señalar que este patrón es implementable tanto en el Contenedor de Servlets, en cuyo caso adopta la forma de un *Singleton*, es decir un único objeto sirve a todos los componentes de la aplicación, como para prestar servicio a los EJB, en cuyo caso perdería sus cualidades de caché global y de *Singleton* pero seguiría facilitando el acceso a otros objetos y componentes de negocio.

- Devolver una referencia válida de modulo al cliente. Puede ser un conjunto de clases java o bien un EJB, en cuyo caso GestorImpresion no solamente realiza la búsqueda JNDI, la creación de objetos remotos, etc,por medio de *Localizador de Servicio,* sino que devuelve al cliente una clase *fachada* que oculta al cliente el trabajo contra un EJB.

Al devolver, en el caso de trabajar con componentes de negocio remotos, un objeto mas simple que envuelve al verdadero objeto que implementa la lógica de negocio, estamos empleando elpatrón Proxy, representando un objeto complejo envuelto en otro mas simple se simplica su manejo por parte del cliente.

Empleando los patrones Proxy, Localizador de Servicio y Delegado de Negocio logramos poner adisposición del resto de la aplicación una serie de objetos cuyo manejo puede resultar complejo, ademas de suponer una recarga a las máquinas que lo alojan si no son manejados adecuadamente.

El cliente no tendría por qué conocer en profundidad el funcionamiento de EJBs, el tratamiento de excepciones remotas o incluso su localización, se pretende delegar estas labores al *GestorImpresion,* de acuerdo al patrón *Delegacion de Negocio.GestorImpresion* se sitúa entre los componentes de negocio y sus clientes..

Capa Vista

El módulo *Vista* gestiona la creación de las diferentes salidas de impresión.

La separación del componente Modelo del de Vista obliga la separación de los datos de su Representación lo que facilita añadir múltiples representaciones de los datos o sea diferentes formatos de impresión para el mismo conjunto de datos y tambien añadir nuevos tipos de datos cuando las circunstancias lo requieran.

Los componentes Modelo y Vista pueden modificarse de forma independiente, excepto su interfaz, mejorando el mantenimiento de la aplicación y sus posibilidades de extensibilidad. En cuanto a las ventajas de separar el Controlador de la Vista, es decir, el comportamiento de la aplicación de su representación, nos permite la selección de las salidas en tiempo de ejecución a elección del Controlador como ya se ha mencionado.

La aplicación web del INEM, Contrat@, está preparada para generar impresiones en formatos múltiples FDF, PDF, XML, PostScript y PCL 5 entre otras. Para lograr esta multiplicidad de salidas, y en orden a permitir incluir mas si fuese necesario, se ha implementado una arquitectura de modulos de impresion incrustables, la inclusión de módulos adicionales no modifica el código existente.

El contrato de diseño establecido para los módulos de impresión es muy simple y poco estricto, de cara a facilitar la creación de nuevos módulos de impresión:

- Los módulos de impresión deben implementar el interfaz ModuloImpresion, que contiene los métodos necesarios para gestionar impresiones.
- No hay límite en cuanto al número de clases necesarias para desarrollar la funcionalidad en tanto en cuanto exista un único punto de acceso disponible, la clase que implemente el interfaz ModuloImpresion, para que lo recupere el componente Controlador .
- Para componentes distribuídos, se establecen clausulas adicionales:

La lógica debe implementarse en un EJB de sesión. El uso de EJB de Entidad está descartado, el EJB dirigido por mensajes tampoco es apto al necesitar un comportamiento síncrono. Se descarta por cuestiones de rendimiento emplear EJB con estado.

Tanto el interfaz remoto, EJBObject, como su correspondiente EJB incluirán el interfaz ModuloImpresion aunque obligatoriamente solo deban implementar un método, *public Object print (DatosImpresion datos)* que desencadena el proceso de generación de Impresiones.

Los métodos restantes incluídos en el interfaz no deben implementarse, no tienen sentido en un EJB de Sesion sin estado, lanzando una excepción genérica ImpresionException si se invocan.

El comportamiento de este capa es el siguiente

El componente Controlador, desde GestorImpresion, debe localizar el módulo de impresióncorrecto. La función de localización de módulos está implementada siguiendo el patrón Localizador deServicio, ya explicado en el apartado anterior

Si la lógica de impresión se ha desarrollado en Java estándar, el *GestorImpresion* devolverá al cliente una referencia de tipo *ModuloImpresion* del componente seleccionado.

Ante componentes distribuídos, el cliente obtiene un nuevo objeto siguiendo los patrones de diseño Delegación de Negocio y Proxy. *GestorImpresion* recupera el componente remoto indicado y lo envuelve en una clase que oculta la naturaleza remota de sus métodos. El cliente obtiene una referencia al Proxy, que tambien es de tipo ModuloImpresion.

El retorno de los métodos de impresión es un *java.lang.Object* genérico, es responsabilidad del cliente del sistema realizar los castings necesarios.

MÓDULOS DE IMPRESIÓN

Los módulos de impresión son piezas de software que aportan funcionalidades específicas de impresión.

Hasta el momento se han desarrollado tres módulos diferentes, generación de FDFs, generación de XML y el tercero, al que se dedica un apartado del documento, generacion de impresiones múltiples basado en el estandar XSL/XSL-FO.

Módulo de Impresión FDF.

La implementación de este módulo va dirigida a la generación de FDF, se utiliza en CONTRAT@. FDF es un formato de archivos definido por Adobe para trabajar con Acrobat Forms, formularios PDF o simplemente formularios. Son todos los datos que envían o reciben estos formularios y que permiten su actualización on line.

FDF permite trabajar sobre un conjunto de datos que pueden ser asignados a cualquier plantilla FDF, el resultado final que observa el cliente es un documento PDF que puede llevar botones de formulario, con los datos contenidos en el archivo FDF. Suelen tener un tamaño muy reducido, la mayoría estan entre los 10KB y 20KB, al contener solamente datos. La plantilla PDF que aloja estos datos es otro de los campos del FDF, donde se indica la URL de la misma. Esta implementación usa la libreria FDFToolkit de Adobe, de libre distribución.

El módulo devuelve como resultado de la impresión un archivo FDF en un objeto java.io.ByteArrayOutputStream, este stream puede ser serializado en un archivo por el cliente o bien servirse directamente a un javax.servlet.ServletResponse (enviarse como respuesta de un servlet).

Es un módulo de propósito muy especializado, orientado fundamentalmente a otorgar funcionalidades de navegacion a documentos legales en PDF.

Frente a otros modos de generacion de PDFs, este módulo permite generar y trabajar con botones de formulario, lo que permite insertar formularios FDF en la navegación de la aplicación.

Módulo de Impresión XML

Genera documentos XML de los diversos tipos de datos de la aplicación a partir de los diferentes esquemas xml o dtds que cada tipo de dato tiene asociado.

Su propósito fundamental es unificar la generación de documentos a partir de los diversos tipos de datos que maneja la aplicación, utilizando para ello el estándar por excelencia para intercambio de datos, el XML. Una vez generados estos xml, pueden enviarse a otros procesos o bien al tercer Modulo de Impresión, el motor de transformaciones e impresion que se describe en el apartado siguiente.

La impresión devuelve un objeto *org.w3c.dom.Document*, que encapsula el XML generado. Este objeto puede serializarse a un archivo o enviarse al Motor de Transformaciones e Impresión XSL.

Para facilitar la creación de esquemas XML o de DTDs en formato XML, se pueden utilizar herramientas que se descargan gratuitamentepara 30 días, como puede ser el XMLSPY.

Módulo de Impresión XSL

La arquitectura abierta y modular de la aplicación CONTRAT@ ha permitido incluir entre los módulos de impresión el módulo que implementa el estandar XSL/XSL-FO, para generar, a partir de documentos XML asociados a plantillas XSL, diferentes salidas de impresión, PDF, PostScript, XML, PCL, RTF, etc.

Este módulo, a diferencia de los anteriores, no es de desarrollo propio, sino que se ha utilizado el motor FOP de la fundación Apache.

Se presenta como un módulo muy versatil, a partir de un único XML de datos y una única hoja de estilo en XSL se pueden generar diferentes impresiones o visualizaciones de los mismos datos. Otra ventaja que proporciona es su facilidad de mantenimiento que permite, creando nuevas hojas de estilo XSL, salidas distintas sin necesidad de reprogramar el módulo.

El módulo es 100% Java, lo que permite su portabilidad inmediata a otras arquitecturas que no sean servidores de aplicaciones, basta disponer de una Java Virtual Machine 1.3 o superior

para que funcione. Esta posibilidad resulta muy útil y proporciona grandes prestaciones al permitir disponer de funcionalidades de impresión avanzadas sin tener que usar servidores de aplicaciones.

A modo de aclaración

El empleo de los términos XSL, XSL-FO y XSLT ofrece una cierta confusión.

XSL (eXtensible Stylesheet Language), es un estándar que especifica un lenguaje para la presentación y transformacóon de documentos XML.

Como estas dos funcionalidades son completamente diferentes, se propuso XSLT (eXtensible Stylesheet Language Transformations) como un estándar que solo contemplara la parte de transformación de documentos XML, dejando la especificación XSL para el resto de las funcionalidades descritas.

Actualmente, XSL se suele identificar con XSL-FO (eXtensible Stylesheet Language - Formatting Objects) que describe un lenguaje de generación de impresión de documentos XML.

MOTOR DE TRANSFORMACIÓN XSL-FO

El Motor de Transformación XSL-FO (eXtensible Stylesheet Language - Formatting Objects) es un motor de generación de impresiones de documentos XML a partir de hojas de estilo XSL (o XSL-FO con instrucciones XSLT).

Descripción general

Este sistema presenta una serie de ventajas:

Entorno de ejecución, puede funcionar en cualquier Sistema Operativo que tenga una Maquina Virtual de Java 1.3 o superior al estar desarrollado el 100% en Java. El mismo desarrollo puede funcionar, indistintamente en AIX, Linux, Unix en general, Windows o incluso sobre os390.

Es gratuito y su código fuente es público. El motor empleado en el INEM es el Apache-FOP (direccion: http://xml.apache.org/fop/index.html), software de open source desarrollado por la Fundación Apache, creadores del Servidor HTTP Apache, el servidor mas usado en internet durante los 8 últimos años. A pesar de ser gratuito, es un software de calidad profesional con el soporte de toda la comunidad de desarrolladores de software libre.

Empleo de estandares, frente a otro tipo de soluciones propietarias, trabaja con documentos XML y hojas de estilo XSL, ambas especificaciones universalmente aceptadas.

Generación de múltiples salidas de impresión, el mismo conjunto de datos puede generar múltiples salidas, PDF, XML, PCL, SVG, Postscript, etc a partir de una única hoja XSL.

Ausencia de programación, facilidad de mantenimiento. La implementación de este tipo requiere de una sencilla capa desarrollada en Java que se limite a hacer de front end. El mantenimiento del software es mínimo. Por otra parte, llevar el desarrollo de un servidor de aplicaciones a una instalacion stand-alone (sin servidor de aplicaciones, solamente con el Java Runtime, invocable por linea de comandos) es inmediato.

Facilidad para cambiar o añadir formatos: Al asociarse cada documento XML en tiempo de ejecución con su correspondiente hoja XSL, basta con cambiar la hoja XSL o crear otra a la hora de añadir o cambiar formatos de impresión

Funcionamiento y estructura de los documentos XSL

Los documentos u hojas XSL son documentos XML con información para producir impresiones´, contienen información sobre el formato de la salida y su contenido. Ya se ha mencionado que es muy frecuente que contengan ademas comandos XSLT para dotar de una mayor versatilidad a los outputs generados.

Los motores de transformación, como el Apache FOP aquí descrito, generan impresiones a partir de documentos XML que contienen los datos y de las hojas XSL que aportan la información necesaria para generar la salida de los datos. Ademas del formato de los datos, el motor puede redirigir esa salida a multiples contextos, es aquí cuando hablamos de generación de múltiples salidas de impresión.

Una misma pareja <documento xml de datos >--- <documentos xsl> puede utilizarse para diferentes salidas: PDF, PCL, PostScript, SGA, texto ascii, etc. El motor intentará, dentro de las limitaciones que impone cada contexto de impresión, generar una salida de los datos lo mas fiel al documento XSL, por ejemplo, una salida que incluya imagenes no tendrá las imágenes si la salida es texto plano, o tambien el resultado de la impresión en PCL será mas pobre que en PostScript o PDF.

Usando el Motor de Impresión XSL-FO

En el caso de implementar el motor como una aplicacion stand alone, invocable por línea de comandos, bastaría realizar un pequeño programa Java que genere los dos archivos (el XML de datos y la hoja de estilo XSL) , seleccione el tipo de impresión a generar para poder emplear FOP, trate las excepciones, mensajes al cliente, etc. De manera similar, la inclusión del motor en una aplicación J2EE tampoco supondría dificultad alguna, y podría envolverse o bien en una clase Java o en un EJB.

Suponiendo que ya se ha desarrollado la clase que se encargara de interactuar con el usuario de impresión, se parte de un fichero XML que contiene un conjunto de datos para los que se desea generar un tipo de impresión. Se debe disponer tambien de un documento XSL-FO, donde se detallan las operaciones de transformación e impresión a realizar sobre el documento XML.

Para conseguir las impresiones, se envían al motor estos dos ficheros, mas el tipo de salida a generar (PDF, PCL, PostScript, RTF...). Hay que señalar que un único XSL-FO basta para generar impresiones de diferentes tipos, siendo responsabilidad del motor el interpretar las instrucciones de impresión de manera distinta segun el formato de salida elegido.

Una vez se le han suministrado al motor los ficheros de datos a imprimir, el fichero XML mas el formato que deben tener esos datos en el documento XSL además del tipo de impresión deseado, el motor comienza el proceso de generación del documento de impresión que es transparente para el cliente. A nivel interno, el proceso transcurre de la manera siguiente:

> En primer lugar, creará un documento temporal en memoria interpretando las instrucciones de transformacion de datos (XSLT) si las hubiera. Internamente crea una estructura de árbol representativa del documento XML sobre la que aplica las transformaciones oportunas hasta llegar a tener otra estructura diferente resultado del proceso de transformación. El motor FOP de Apache emplea para este propósito el motor XALAN.

A continuación, a este documento temporal se le aplican las instrucciones de impresión. De nuevo se examina la estructura del documento y se le van aplicando las sentencias de impre-

sión.En último lugar, el motor crea el tipo de impresión seleccionada para ese documento XML transformado y formateado de acuerdo a las reglas del fichero XSL-FO. Destacar que hasta este último punto el proceso es idéntico para cualquier tipo de impresion elegida, es en el momento de generar una impresión concreta cuando el motor debe particularizar su comportamiento segun el tipo de salida elegido. FOP genera salidas directamente imprimibles como PCL o PostScript asi como documentos imprimibles de tipo PDF o RTF.

CONCLUSIÓN

A modo de resumen se quiere destacar los puntos siguientes:

Con este sistema se unifica la definición de datos en toda la aplicación utilizando el estándar XML permitiendo la impresión nativa de los XML definidos.

Es un sistema potente, fácil de implementar y de poco mantenimiento, tengamos en cuenta que admite múltiples salidas PDF, PCL, PostScript, RTF, XML, FDF, se puede instrumentar en servidor de aplicaciones web o en cualquier otro servidor que tenga instalada una Máquina Virtual Java (MVJ) 1.3. El INEM lo utiliza en servidores Unix.

Utiliza software libre, descargado de la red.

El Sistema de Impresión que se ha pretendido exponer brevemente en este documento está desarrollado y programado para una MVJ 1.3, en el INEM y se pone a disposición de cualquier organismo de la Administración Pública Central, Autonómica o Local que desee utilizarlo.

En este documento, por razones de volumen, no se incluye la programación ni los modelos UML de los diferentes componentes.