VNIVERSIDAD NACIONAL
AVPNOMA DE
MEXICO

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

DISEÑO, IMPLEMENTACIÓN Y EVALUACIÓN DE UN EDITOR DE DOCUMENTOS XML EN JAVA PARA EL SISTEMA DE AUTORÍA COLABORATIVA ELXI

T E S I S

QUE PARA OBTENER EL GRADO DE:

MAESTRO EN INGENIERÍA (COMPUTACIÓN)

P R E S E N T A:

ANIBAL JESÚS AVELAR ROSALES

DIRECTOR DE LA TESIS: DR. MANUEL ROMERO SALCEDO

MÉXICO, D.F. **2004**

Dedicatorias y Agradecimientos

Dedicada a mi esposa Lidia. Sin ti, nunca hubiera podido concluir este trabajo. Gracias Lidy.

Un agradecimiento a toda la comunidad de usuarios de Debian GNU/Linux por sus consejos y buenos trucos. El presente trabajo fue realizado en su totalidad con herramientas de software libre. En la construcción de la aplicación y la edición del presente documento.

Una especial mención a mi asesor y tutor de la maestría en Ingeniería y Ciencias de la Computación, el Dr. Manuel Romero Salcedo.

Índice general

Índice de figuras

Índice de tablas

PREFACIO

El presente trabajo describe el diseño, implementación y evaluación de un editor de documentos XML[1] (lenguaje de marcado extensible) construido con el lenguaje de programación Java.

La aplicación forma parte del sistema de *Edición coLaborativa de documentos XML en Internet* (ELXI). El diseño de ELXI se ha basado principalmente en la característica de permitir que existan varios escritores editando un documento a la vez. El sistema hoy día se está trabajando para convertirlo en un producto final. Este sistema se caracteriza por su funcionalidad debido a que parte de la arquitectura, la administración de los documentos, el espacio de trabajo y la sesión de trabajo han sido implementados exitosamente.

La arquitectura base de ELXI fue construida con la colaboración del Ing. César Murillo, la M. en I. Silvia Ramírez y el que suscribe. Cada uno ha intervenido en el desarrollo del sistema con un tema en específico:

- **Consistencia sobre los documentos XML replicados.**
- **Control de concurrencia en un sistema de autoría colaborativa.**
- **Diseño, implementación y evaluación de un editor de documentos XML (el presente trabajo).**

El sistema ELXI ha despertado el interés de varios grupos de trabajo en la industria. Así mismo, ha sido impulsado por el Consejo Nacional de Ciencia y Tecnología (CONACYT) mediante el proyecto "Ambiente Multipropósito para la Edición Colaborativa en Intranet e Internet" con referencia 4003165-J3204A, dirigido por el Dr. Manuel Romero Salcedo y por el proyecto "Computación Distribuida Inteligente (CDI)" con referencia D.00006 del Instituto Mexicano del Petróleo.

El objetivo en estos proyectos es crear un ambiente de edición colaborativa para la elaboración de documentos científicos y técnicos en el formato estándar universal XML. Se requiere utilizar un formato estándar universal común para todas las plataformas computacionales conocidas (Windows, Linux, Solaris, Mac, etc). En muchas ocasiones los usuarios, venidos de diferentes campos del conocimiento, desean usar su editor de textos predilecto. Sin embargo, al momento de juntar los diferentes documentos, se dan cuenta que se desperdicia un tiempo enorme para conseguir

[1] Del inglés "Extensible Markup Language".

un documento consistente y en un sólo formato. Lo ideal es que cada usuario use o genere un formato común y universal, con ello se evitarían los problemas de compatibilidad de formatos, fusión de versiones, ajustes de presentación, etc. También es de vital importancia que esa información esté representada de tal forma que permita ser reutilizada de diferentes maneras, con el fin de minimizar tiempos y esfuerzos. El lenguaje XML, como se observará en el transcurso de este trabajo de tesis, se adapta muy bien a todos estos requerimientos.

Objetivo

El presente trabajo tiene como objetivo diseñar, implementar y evaluar una aplicación que permita la edición de documentos compartidos y que usa al lenguaje XML como formato interno de los documentos.

El editor será realizado en el lenguaje de programación Java y hará uso del formato XML para controlar y almacenar la información de todos los documentos. Permitirá manipular los documentos, haciendo uso de herramientas que faciliten la edición colaborativa. También permitirá la visualización de los documentos creados y su exportación a otros formatos como el lenguaje de marcas hipertextuales (HTML[2]) y el formato portable de documentos (PDF[3]).

También debe permitir la realización de otros procesos del trabajo colaborativo, como son la creación de grupos de trabajo, la asignación de roles y la asignación de permisos de grupo y documentos.

[2]Del inglés "Hypertext Markup Language"
[3]Del inglés "Portable Document Format".

INTRODUCCION

¿Qué es XML?

XML es un meta-lenguaje que proporciona un formato para describir datos de forma estructurada. Esto facilita declaraciones de contenido y resultados de búsqueda más precisos entre múltiples arquitecturas de cómputo. El lenguaje XML fue desarrollado por el *Consorcio de la Telaraña Mundial* (W3C[4]).

XML ofrece a cualquier programador el poder contar con datos estructurados en una amplia variedad de aplicaciones. XML permite crear documentos con formatos de datos que son exclusivos para aplicaciones específicas; es además un formato ideal para la transferencia de datos entre aplicaciones en servidores remotos de una forma estructurada y coherente. Por tanto, XML es ante todo un metalenguaje que permite diseñar un lenguaje propio de etiquetas para múltiples clases de documentos. Por ese motivo XML es el formato ideal para el editor colaborativo que necesita distribuir la información a través de distintos clientes que se encuentran en distintos lugares sobre la red, manteniendo la consistencia e integridad en los datos.

Antecedentes de colaboración y edición distribuida

Con el surgimiento de nuevas tecnologías en las comunicaciones y en el campo de la computación, se han ido optimizando los sistemas de control de información. Principalmente, se han salvado distancias y minimizado el tiempo para la obtención de datos. Debido a ello, la demanda de los sistemas de cómputo se ha incrementado, así como también las necesidades de adaptación de los usuarios finales para interaccionar con ellos.

Algunas de las aplicaciones que han surgido dentro de estas tendencias, como es el caso del correo electrónico, los cuartos de conversación (chats), la transferencia de archivos, la videoconferencia, etc., han facilitado el desarrollo de algunas actividades llevadas a cabo por personas que trabajan de manera conjunta para resolver un problema en común. Sin embargo, estas aplicaciones no han cubierto del todo las necesidades de los grupos de trabajo porque no proporcionan las suficientes herramientas para apoyar las tareas de comunicación, colaboración y coordinación de

[4]Del inglés "World Wide Web Consortium". Organismo que vela por el desarrollo de la teleraña mundial WWW.

actividades. Por ejemplo, el correo electrónico es un medio que no permite la suficiente coordinación y colaboración porque que es asíncrono y no permite la interacción directa entre usuarios, o los cuartos de conversación que no permiten el intercambio de datos y solo transmiten mensajes de forma síncrona. Para resolver esta problemática, en el área de investigación de *Trabajo Colaborativo Asistido por Computadora* (CSCW[5]), junto con el de *Interacción Humano Computadora* (HCI[6]), se han desarrollado sistemas colaborativos que asisten las actividades y el trabajo en grupo. Sin embargo, debido a que los usuarios frecuentemente se encuentran distribuidos en tiempo o espacio, es importante considerar mecanismos que faciliten el almacenamiento de la información compartida y consistente, la coordinación del trabajo y la comunicación a través de medios eficientes.

Es en la comunicación a través de medios eficientes donde surge la necesidad de una aplicación para el usuario final que permita el trabajo colaborativo y la edición de los documentos. En las siguientes secciones se exponen las arquitecturas y tecnologías que son necesarias para el funcionamiento del editor distribuido. Asimismo, se describe la estrategia de programación que es utilizada para atender las solicitudes del usuario que hace uso del editor y su interacción con otros usuarios que hacen uso del sistema.

Criterios y requerimientos en el diseño de sistemas groupware

El CSCW es una disciplina emergente que motiva y asiste a sistemas para el trabajo en grupo con ayuda de tecnologías de información y comunicación (como son las computadoras), siendo atractiva a aquellos interesados en el diseño del software y el comportamiento social y organizacional, científicos de la computación, investigadores de la comunicación, entre otros.

Para asistir las actividades que lleva a cabo un grupo de personas en la realización de una tarea en común, se deben considerar tres aspectos clave: la comunicación, la colaboración y la coordinación. La comunicación es uno de los aspectos más importantes en un grupo de trabajo, ya que mediante ella se realiza el intercambio de información necesaria para el desarrollo de una actividad. Ésta puede ser espontánea y sin limitantes cuando las personas se encuentran en un mismo lugar. Sin embargo, cuando se encuentran a distancia pueden recurrir a diferentes medios, por ejemplo el teléfono o las aplicaciones creadas para Internet como los cuartos de conversación. Hay aplicaciones, en las que no se permite una interacción entre usuarios, tan sólo realizan el envío de mensajes, como el correo electrónico.

Otro pilar en las actividades de grupo es la colaboración, en la cual se establece la relación entre el trabajo individual y de grupo. Además, para hacerla efectiva es necesaria la compartición de información. Algunos de los grupos de trabajo se auxilian de los sistemas de cómputo como las bases de datos o los servidores de archivos para que múltiples usuarios puedan acceder a un mismo conjunto de datos.

[5]Del inglés "Computed-Supported Cooperative Work".
[6]Del inglés "Human Computer Interaction".

La comunicación y la colaboración pueden ser mejoradas si las actividades del grupo son coordinadas. Por ejemplo, cuando dos usuarios alteran un mismo documento al mismo tiempo, cada quien tendrá una versión del documento, siendo inconsistente la información para ambos. Una posible solución a este problema es la edición del documento por turnos, una vez que un colaborador haya finalizado con las modificaciones pertinentes, el siguiente colaborador podrá trabajar en el documento. Cuando este último haya terminado, el primer usuario podrá disponer de la versión del segundo. Este tipo de coordinación permite que ambos usuarios obtengan las aportaciones que uno y otro ha escrito en el mismo documento.

Se puede apreciar que los grupos de personas se están apoyando en los sistemas de cómputo para llevar a cabo las actividades de comunicación, colaboración y coordinación, sin embargo éstas no han podido ser apoyadas plenamente debido a que la mayoría de estos sistemas proporcionan una interacción entre usuario y sistema, y no la incorporan entre usuarios. Debido a esto, ha surgido la necesidad del diseño de sistemas de cómputo que asistan el trabajo en grupo, mejor conocidos como *groupware* [Ell91]. Dichos sistemas, se han enfocado en la solución de los problemas de la comunicación y de la coordinación de actividades en la realización de una tarea en común. Se propone entonces la siguiente definición de los sistemas groupware: "Sistemas de cómputo que asisten a un grupo de personas involucradas en una tarea o propósito en común y que proveen una interfaz para un espacio compartido" [Ell91]. En esta definición, hay dos aspectos importantes que subrayar, tarea o propósito en común y espacio compartido, por lo que quedan excluidos aquellos sistemas multiusuarios, en donde los usuarios no comparten alguna tarea en común, como podrían ser los sistemas de computo UNIX que permiten sesiones multiusuario y un espacio compartido (el servidor), pero cada usuario realiza tareas distintas y sin tener propósitos en común.

En la actualidad, existen sistemas groupware con diferentes funcionalidades que sirven de apoyo al trabajo colaborativo. Algunos proponen aplicaciones para generar ideas, estructurarlas y finalmente evaluarlas. Otras herramientas, como los calendarios compartidos, asisten al manejo de citas, tareas, programación de juntas, existen otras que permiten trabajar sobre un recurso o documento como los editores compartidos, las hojas de cálculo compartidas o el dibujo compartido. La meta de CSCW es descubrir formas de usar la tecnología para los grupos de trabajo a través del tiempo y el espacio. Por ello, las aplicaciones groupware no tratan de remplazar a la gente en una situación interactiva, en lugar de ello, es una herramienta mejorada para los procesos colaborativos.

En el diseño de los sistemas groupware, existen varios criterios [JaLo92, Vol93]. Estos criterios pueden ser utilizados como requerimientos para la definición de un sistema groupware. A continuación se describen.

- **Espacio de trabajo.** Es una simulación de un espacio físico mediante computadoras, en donde las personas pueden ver y manipular los objetos relacionados con las actividades que realizan. Uno de los beneficios en el desarrollo del trabajo colaborativo es que cada

usuario utiliza lo que hay en su espacio de trabajo individual, sin que sea modificado por otros. Sin embargo, algunas veces es deseable que las modificaciones que se realizan en el sistema sean visibles para todos los usuarios involucrados.

- **Conciencia de grupo.** Para lograr una colaboración consciente entre usuarios, se requiere mantener la compartición de la información y el conocimiento de las actividades grupales e individuales que se lleven a cabo, es decir, mantener una conciencia de grupo en los espacios de trabajo compartido.

- **Interacción.** En comparación con las aplicaciones monousuario, existen nuevos requerimientos para la interacción en un ambiente groupware. La interacción puede ser síncrona o asíncrona dependiendo del tiempo de respuesta que se requiere. Por ejemplo, la edición conjunta de varios autores en la misma posición del texto se relaciona con el trabajo síncrono, mientras que enviar un correo electrónico es una tarea asíncrona.

- **Coordinación.** Para poder comunicarse y colaborar con el resto de la organización es necesario un medio que permita la integración y la adaptación armoniosa de todos los elementos que participan en un proceso. En la interacción, la coordinación depende del tamaño del grupo y de la forma de trabajo individual de cada uno de los miembros del grupo con respecto al modo en que prefieren interactuar.

- **Memoria organizacional.** Se refiere al registro del proceso de la interacción del grupo, incluyendo la comunicación, las tareas ejecutadas, los productos obtenidos y su historial. Este tipo de memoria proporciona a los colaboradores una mejor comunicación y coordinación en sus actividades.

- **Distribución.** Gracias a las redes de computadoras, los usuarios, situados en diferentes lugares geográficos, pueden interactuar remotamente. De acuerdo a la forma en que los procesos están ubicados en una red, los sistemas groupware se pueden dividir dependiendo de su arquitectura en centralizados, distribuidos o híbridos.

- **Flexibilidad.** Se refiere a la habilidad de poder cambiar dinámicamente de estados o modos en un sistema groupware. Por ejemplo, en un sistema que tiene definido su espacio de trabajo pueden existir diferentes herramientas abiertas y el usuario puede estar interactuando con una y con otra sin que se vea alterada la ejecución de sus procesos.

- **Visualización.** En un espacio compartido es necesario reflejar un estado coherente del mismo. Por ejemplo, cuando se manipulan objetos remotos, los usuarios deben percibirlos como si fueran objetos locales para cada uno. Para la coherencia de la interfaz, se ha definido el concepto de vistas de espacio, las cuales se clasifican en cuatro tipos:

    - WYSIWIS (WHAT YOU SEE IS WHAT I SEE)[Ste87], en donde la vista del contenido del espacio es la misma para cada usuario.

- WYSIWYG (WHAT YOU SEE IS WHAT YOU GET)[App87], en donde la vista representa exactamente cada objeto, tal y como existe dentro del espacio.
- WYSIWIMS (WHAT YOU SEE IS WHAT I MAY SEE)[NePe91], permite perspectivas de vistas independientes y espacios de trabajo privados.
- WYSIMID (WHAT YOU SEE IS WHAT I DO)[Gut96], en donde cada usuario visualiza, en forma síncrona o asíncrona, las acciones realizadas por su demás colegas.

La visualización es uno de los principales criterios que se tomaron en cuenta para el diseño del editor colaborativo y que junto con los criterios de espacio de trabajo, la conciencia de grupo, la coordinación y la distribución permiten describir el funcionamiento y comportamiento de la aplicación descrita en éste trabajo de tesis. El tipo de interfaz que se diseño en este trabajo se podría definir como de WYSIWIS porque todos los usuarios muchas veces podrán ver lo mismo. También, como permite visualizar por cada uno de los usuarios en sesión cada cambio en los documentos de forma síncrona (o asíncrona si el usuario no esta en sesión) se define también como de WYSIMID. Además, como permite editar diferentes documentos y estos pueden ser privados y sólo para uso personal es también WYSIWIMS. Sin embargo, como la parte de edición se encuentra separada de la parte de visualización por motivos de diseño y del tipo de editor que se planteó al inicio del proyecto. La idea inicial era de dar funcionalidad colaborativa y de visualización por medio de transformación y no de ser un editor WYSIWYG que representa cada objeto tal como es, en edición y visualización.

SISTEMA DE AUTORÍA COLABORATIVA ELXI

Actualmente, la arquitectura del sistema ELXI y algunos procesos como la administración y elaboración de documentos, creación de sesiones de trabajo y envíos de mensajes, han sido implementados. Sin embargo, el proyecto ELXI continua en la fase de desarrollo. Uno de los requerimientos que se demanda es una interfaz para el usuario que permita la edición de documentos. Con el fin de proponer este mecanismo, primero se describirá el estado inicial del que se parte, es decir cómo está integrado y cómo trabaja ELXI y de tal forma conocer en qué parte de la integración de ELXI forma parte el editor. Para esto se necesita conocer cuales son los componentes que constituyen al sistema ELXI y así visualizar en que parte esta localizado el editor colaborativo que se nombrará a partir de aquí como *Editor XML de Transformación* (XTE[7]). En la figura 1 se visualizan los componentes y tecnologías involucradas en el editor colaborativo y su relación con el sistema ELXI.

[7]Del inglés "XML Transformation Editor".

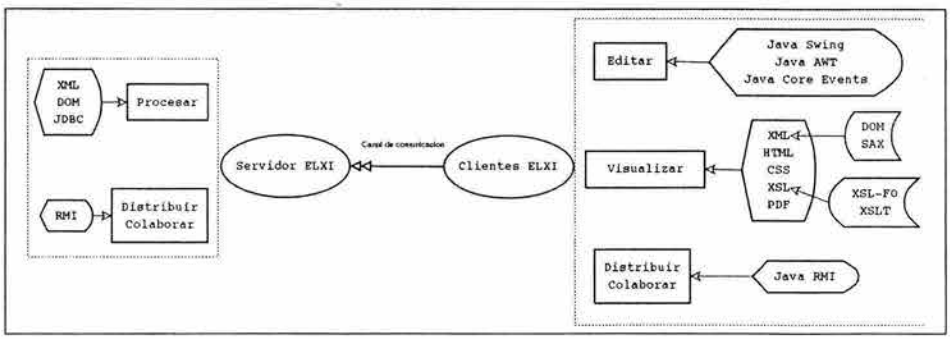

Figura 1: Componentes ELXI

Organización

La presente tesis consta de 4 capítulos cada uno de los cuales trata de un problema distinto a resolver para constuir el editor colaborativo. En la figura 2 se ven cada una de las tecnologías involucradas en el editor y como se relacionan con cada uno de los capítulos del presente trabajo.

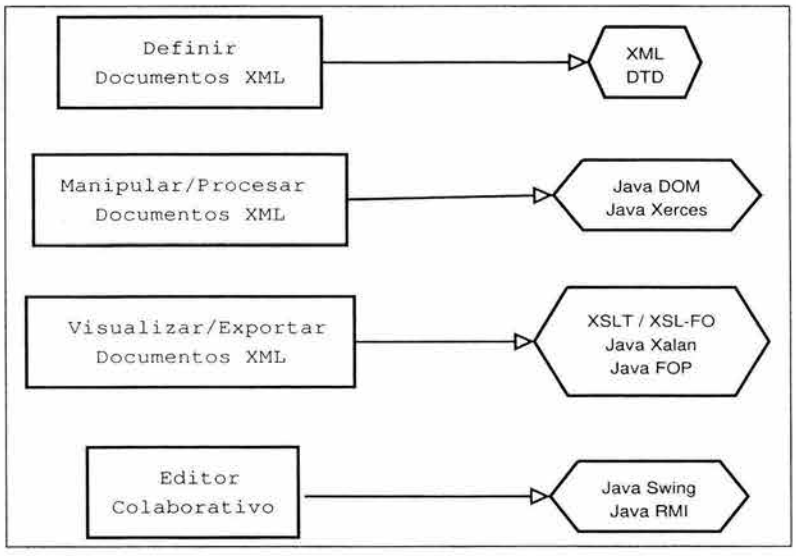

Figura 2: Componentes del editor colaborativo

En el Capítulo 1 se verán conceptos básicos de XML como son su definición, ímportancia y uso. Cómo construir y procesar los documentos XML. Que es una DTD y cómo se define. Porqué es importante analizar la estructura de un documento XML. Un breve análisis del modelo de objeto de documento (DOM[8]). Se describirán distintas herramientas que existen para analizar XML.

En el Capítulo 2 se define el modelo de objeto de documento (DOM) . Cómo hacer uso de la interfaz de programación Java DOM llamada Xerces[9]. Cómo hacer uso de las bibliotecas de Java Xerces para realizar las distintas operaciones de crear, borrar, añadir y manipular los nodos del árbol DOM.

[8]Del inglés "Document Object Model".

[9]Desarrollado por la fundación Apache. Esta en el lenguaje de programación Java y es de código abierto.

En el Capítulo 3 se describe cómo dar formato a los documentos XML para su posterior visualización. Cómo se define el formato de hojas de estilo extensible (XSL[10]) y conocer sus dos principales especificaciones los objetos de formateo del lenguaje de hojas de estilo extensible (XSL-FO[11]) y el lenguaje extensible de hojas de estilo de transformación (XSLT[12]). Cómo usar XSLT para transformar un documento XML al formato HTML. Cómo crear plantillas de diseño XSL-FO para transformar al formato PDF. También se describe como hacer uso de la interfaz de programación Java Xalan para procesar las plantillas XSLT y la interfaz de programación Java FOP para procesar las plantillas XSL-FO.

En el Capítulo 4 se describe el diseño, implementación y evaluación del editor colaborativo de documentos realizado en Java. Se hace uso de la interfaz de programación Java Swing que permite crear aplicaciones gráficas de una gran funcionalidad. También se hace uso de las bibliotecas involucradas en el control de los eventos eventos de los componentes (del mouse, del teclado, de las ventanas, etc.). En este capítulo se hace un análisis de cómo fue construido y cuales son los componentes más importantes del editor XTE. Se describen los procesos mas importantes del editor: el procesamiento de los documentos XML, la funcionalidad colaborativa y cómo realiza los procesos de transformación XSLT/XSL-FO para poder visualizar y exportar los documentos XML a otros formatos.

En las conclusiones y perpectivas se describen los objetivos alcanzados en este trabajo de tesis. Cuál fue el alcance y lo qué falta por hacer.

En la evaluación, resultados y discusión se narran los resultados obtenidos de la evaluación realizada a miembros del Instituto Mexicano del Petróleo.

[10]Del inglés "Extensible Style Language".
[11]Del inglés "Formatting Objects Extensible Style Language".
[12]Del inglés "Extensible Style Language Transformation".

Capítulo 1

DEFINICIÓN Y PROCESAMIENTO DE DOCUMENTOS XML

En este capítulo se verá la definición de documentos XML. La estructura y modelado de datos con XML. ¿Que es D*efinición del Tipo de Documento* (DTD[1])?. Las DTD a profundidad. Cómo se procesa un documento XML. Porque es conveniente analizar sintácticamente al documento XML y algunos fundamentos del procesamiento XML. ¿Que es un documento XML bien-formado? y la diferencia con un documento válido. El árbol de análisis como medio para hacer la validación del documento XML. Analizar sintácticamente por medio del lenguaje de programación Java. Breve descripción de los analizadores sintácticos XML en Java que hay en el mercado. Los analizadores sintácticos XML simple y de objeto de documento. El analizador sintáctico Java Xerces.

1.1. Introducción

HTML se convirtió en un lenguaje de marcas de inmensa popularidad durante estos últimos años. Sin embargo, es un lenguaje que tiene limitaciones como son: no se puede separar el formato del contenido de una manera clara, los datos no mantienen una estructura definida, los documentos son estaticos y el lenguaje tiene distintas interpretaciones en distintos clientes. Se ha querido subsanar las limitaciones del lenguaje HTML por medio del lenguaje de marcas HTML dinámico (DHTML[2]) y las hojas de estilo en cascada (CSS[3]). Todo esto ha sido insuficiente para crear una arquitectura abierta de tipo cliente/servidor, por lo que el W3C, se propuso crear un nuevo estándar llamado XML, el cual parte de las bases del lenguaje generalizado estandar de marcas (SGML[4]). SGML fue diseñado para permitir el intercambio de información entre distintas plataformas, soportes físicos, lógicos y diferentes sistemas de almacenamiento y presentación (bases de

[1] Del inglés "Document Type Definition".
[2] Del inglés "Dynamic HTML".
[3] Del inglés "Cascading Style Sheet".
[4] Del inglés "Standard Generalized Markup Language".

datos, edición electrónica, etc.), con independencia de su grado de complejidad. El XML es un subconjunto del SGML.

XML se ha desarrollado por el Grupo de Trabajo XML desde 1996, que en esos primeros años era llamado *Comité Examinador Editorial*[5] SGML. Para el 10 de febrero de 1998 el W3C ratificó la especificación XML en su versión 1.0. Actualmente la versión 1.1 es ya una recomendación oficial desde el 4 de febrero de 2004. Esta recomendación establece entre otras cosas que: XML es un metalenguaje que ordena, estructura y describe los documentos de las páginas Web[6]. XML no sustituye a HTML pues sirven para cosas distintas: HTML para presentar información en páginas Web y XML para representar e intercambiar datos, independientemente de su presentación. XML y HTML son complementarios. Algunos de los objetivos planteados por el Grupo de Trabajo XML y el W3C son:

- XML debe ser directamente utilizable sobre Internet.
- XML debe soportar una amplia variedad de aplicaciones.
- XML debe ser compatible con SGML.
- Debe ser fácil la escritura de programas que procesen documentos XML.
- Los documentos XML deben ser legibles por los usuarios de este lenguaje y razonablemente claros.
- El diseño de XML debe ser formal, conciso y preparado rápidamente.
- Los documentos XML deben ser fácilmente creables.

1.2. Definición de documentos XML

Como se mencionó anteriormente, XML es un meta-lenguaje que proporciona un formato para describir datos de forma estructurada y coherente. Esto facilita declaraciones de contenido más precisas y resultados de búsqueda más precisos entre múltiples arquitecturas de cómputo.

XML ofrece a cualquier programador el poder de contar con datos estructurados en una amplia variedad de aplicaciones. XML permite crear documentos con formatos de datos que son exclusivos para aplicaciones específicas; es además un formato ideal para la transferencia de datos entre aplicaciones en servidores locales remotos de una forma estructurada y coherente. Por tanto, XML es ante todo un metalenguaje que permite diseñar un lenguaje propio de etiquetas para múltiples clases de documentos.

[5] Del inglés "Editorial Review Board".
[6] Abreviatura de "World Wide Web" o WWW.

Existen muchos beneficios al utilizar XML tanto en la Web como en otros contextos:

- **Proporciona datos de forma local.** Los datos proporcionados están disponibles de forma local. Los datos pueden ser leídos por el analizador sintáctico de XML, y a continuación enviados a una aplicación local, como un programa cliente de Web, para una posterior visualización o procesamiento. También los datos pueden ser manipulados mediante algún lenguaje de programación utilizando el modelo de objetos XML.
- **Ofrece a los usuarios una vista apropiada de datos estructurados.** Los datos distribuidos pueden ser presentados de diferentes formas. Un conjunto de datos local puede ser presentado en la vista más adecuada al usuario, dinámicamente, basada en factores como la preferencia y configuración del usuario.
- **Permite la integración de datos estructurados desde diversas fuentes en vistas lógicas comunes.** Se utilizan diversas fuentes que integran datos que estén disponibles para su distribución en el escritorio de trabajo y otros servidores para una posterior agregación, procesamiento y distribución de los mismos.
- **Mejora el rendimiento mediante actualizaciones granulares.** XML permite la actualización granular. Los desarrolladores no tienen que enviar los datos estructurados completos cada vez que se produce un cambio. Con la actualización granular, únicamente el elemento modificado debe ser enviado desde el servidor al cliente. Los datos modificados pueden ser presentados sin la necesidad de refrescar la página completa o la tabla.

1.2.1. Estructura del XML

El XML consta de cuatro especificaciones (el propio XML sienta las bases sintácticas y el alcance de su implementación):

- **DTD** - Definición del Tipo de Documento. Es, en general, un archivo que encierra una definición formal de un tipo de documento y, a la vez, especifica la estructura lógica de cada documento. Define tanto los elementos de una página como sus atributos. La DTD del XML es opcional. En tareas sencillas no es necesario construir una DTD, entonces solo se trataría de un documento bien formado y si lleva una DTD será un documento válido.
- **XSL** - Lenguaje de hojas de estilo extendido. Define o implementa el lenguaje de estilo de los documentos escritos para XML. Permite modificar el aspecto de un documento. Se pueden lograr diversas columnas, texto girado, orden de visualización de los datos de una tabla, múltiples tipos de letra con amplia variedad en los tamaños, etc. Este estándar está basado en el lenguaje de semántica y especificación de estilo de documento (DSSSL[7]) y,

[7]Del inglés "Document Style Semantics and Specification Language".

por otro lado, se considera más potente que las hojas de estilo en cascada (CSS), usado en un principio con el lenguaje HTML dinámico (DHTML[8]).

- **XLL**[9] - Lenguaje de enlaces extendido: Define el modo de enlace entre diferentes enlaces. Esta especificación a su vez, esta basada en la especificación *XPath*[10]. Este lenguaje de enlaces tiene dos componentes:

  1. **XLink**[11]: Define la forma en la que los documentos deben enlazarse y es un enlace bidireccional que nos permite navegar entre páginas. Un enlace puede tener múltiples enlaces destino. Estos enlaces nos llevan al documento completo.
  2. **XPointer**[12]: Describe como se debe apuntar a un lugar específico en un determinado documento XML (utiliza *XPath*). Son enlaces bidireccionales que nos llevan no al documento completo, sino a una parte concreta dentro de éste.

  El XLL va más allá de los enlaces simples que sólo soporta el HTML. Se establecen los siguientes mecanismos hipertextuales que soporta esta especificación:

  1. Denominación independiente de la ubicación.
  2. Enlaces que pueden ser también bidireccionales.
  3. Enlaces que pueden especificarse y gestionarse desde fuera del documento a los que se apliquen.
  4. Hiperenlaces múltiples (anillos, múltiples ventanas, etc.).
  5. Enlaces agrupados (múltiples orígenes).
  6. Transclusión (el documento destino al que apunta el enlace aparece como parte integrante del documento origen del enlace).
  7. Se pueden aplicar atributos a los enlaces (tipos de enlaces).

- **XUA**[13] - Agentes de usuario para XML. Estandarización de programas cliente de visualización de datos XML. Todavía está en proceso de creación de borradores de trabajo. Se aplicará a todos los programas cliente para que compartan todos las especificaciones XML.

Implementaciones de XML

Algunas de las implementaciones/aplicaciones que se han desarrollado con XML son:

[8] Del inglés "Dynamic HTML".
[9] Del inglés "Extensible Linking Language".
[10] XPath es una especificación que indica como navegar por los nodos que forman el árbol de un documento XML.
[11] XLink desde el Junio 27 de 2001 es una recomendación y la fecha de este trabajo la actual versión es la 1.0.
[12] Xpointer desde el 25 Marzo de 2003 es una recomendación y a la fecha de este trabajo la actual versión es la 1.0
[13] Del inglés "XML User Agent".

- **RDF**[14] - Esquema de descripción de recursos. Es una de las aplicaciones más importantes y permite describir los datos de cada documento y definir las relaciones que hay entre los datos XML. Podremos encontrar las siguientes virtudes del RDF:

  1. Mejores motores de búsqueda. Se han adherido a esta especificación los motores de búsqueda de Yahoo!, Altavista, WebCrawler, Google, etc.
  2. La capacidad de describir los contenidos y sus relaciones en una biblioteca digital o sede Web.
  3. Permite el acceso a una parte concreta del documento y facilita el intercambio de los datos.
  4. Establece los derechos de propiedad intelectual en las propias páginas Web.

- **CML**[15] - Lenguaje de marcas para química. Describe, entre otras formulas, las estructuras moleculares y cristalinas, los análisis de espectros y otros objetos de interés para los químicos.

- **MathML**[16] - Lenguaje de marcas para matemáticas. Apto para codificar signos matemáticos, símbolos científicos, etc.

- **EDI**[17] - Intercambio electrónico de datos.

Desarrollo de aplicaciones con XML

Se establecen cuatro tipos de aplicaciones que impulsarán el desarrollo del XML:

- Aplicaciones que exijan que el cliente Web medie entre dos o más bases de datos. Se hará posible la integración de bases de datos distribuidas en los programa cliente que admitan XML, pudiéndose modificar el contenido y la estructura de ésta. Se irá pareciendo cada vez más a una arquitectura cliente-servidor.

- Aplicaciones que intentan transferir una parte significativa de la carga del proceso del servidor al cliente Web. Funcionará con un subprograma en algún lenguaje de programación que se insertará en la computadora del cliente. Esta carga hará que muchas de las funciones de modificación puedan desarrollarse desde el mismo programa cliente Web.

- Aplicaciones que precisen que el cliente Web presente diferentes versiones de los mismos datos a diferentes usuarios.

[14] Del inglés "Resource Description Framework".
[15] Del inglés "Chemical Markup Language".
[16] Del inglés "Mathematical Markup Language".
[17] Del inglés "Electronic Document Interchange".

- Aplicaciones en las que agentes Web inteligentes intentan adaptar la búsqueda de información a las necesidades de los usuarios. Habrá una interacción entre la información requerida y las preferencias del usuario de la aplicación. Con el XML vendrá una segunda generación de robots que permitirá una mayor precisión en la búsqueda requerida [AlaGa98].

Características del lenguaje XML

Algunas de estas características ya fueron mencionadas anteriormente. Además, se tienen las siguientes:

- Es un lenguaje de marcas abierto y extensible. Los identificadores pueden crearse de manera simple y ser adaptados por medio de un validador de documentos o analizador sintáctico.
- Integración de los datos desde fuentes muy distintas. Se podrá hacer el intercambio de documentos entre las aplicaciones tanto en la misma computadora como en una red local o red extensa.
- Datos compuestos de múltiples aplicaciones. La extensibilidad y flexibilidad de este lenguaje permitirá agrupar una variedad amplia de aplicaciones.
- El concepto de hipertexto permite denominación independiente de la ubicación, enlaces bidireccionales, enlaces que pueden especificarse desde fuera del documento, hiperenlaces múltiples, enlaces agrupados, atributos para los enlaces, etc. Creado a través del lenguaje de enlaces extensible (XLL).

En la tabla 1.1 se observa un resumen con algunas de las diferencias significativas del XML con respecto a los otros lenguajes que se han mencionado:

	HTML/DHTML	XML	SGML
Gramática	Fija y no ampliable	Extensible	Extensible
Estructura	Monolítica	Jerárquica	Jerárquica
N° de marcas	Fijas	Sin límite	Sin límite
Complejidad	Baja	Mediana	Alta
Diseño de páginas para navegador	Fijado por tags. Etiquetas con atributos CSS en DHTML	CSS o DSSSL XSL(XSLT)	DSSSL
Enlaces	Simples enlaces	Poderosos enlaces (XLL)	HyTime
Exportabilidad (formatos/aplicaciones)	No	Sí	Sí
Validación	Sin validación	Pueden validarse	Obligatorio DTD
Indización/Catalogación de páginas Web	Sólo lo permite los atributos de la etiqueta <META>, e implementaciones como DC.	Una descripción abierta y personalizable con el RDF.	Algún proyecto como TEI, DLI, etc.

Tabla 1.1: Resumen de diferencias entre diferentes lenguajes de marcado de etiquetas [XmlQue]

XML se está actualmente usando en aplicaciones fuera del mundo Internet/Intranet, haciendo mucho más sencillo y comodo el intercambio de datos.

1.2.2. Modelado de datos con XML

Fundamentos de la Definición de Tipo de Documento (DTD)

Una DTD es una especificación y estructuración necesaria que permite validar el contenido estructural y formal de un documento de SGML y por lo tanto también de XML. Las DTD se pueden usar para la definición de modelos de contenido, es decir, en qué orden y qué elementos pertenecen a un elemento de orden superior en la jerarquía del documento; además permiten (aunque de modo muy limitado) imponer ciertas restricciones sobre el tipo de los elementos. Una DTD indica sólo cuáles elementos, atributos, etc, tiene un documento y cómo se anidan. No dice nada acerca de tipos de dato. El único tipo de dato que reconoce es CDATA[18] (texto plano). Para el editor colaborativo XTE, éste es el tipo de validador de documento que se utiliza, sobre todo porque al tipo de documento que se definió para el editor no necesita mayor rigidez. Sin embargo, cuando se necesita algo más rígido se debe usar XML *Schema*.

Fundamentos de XML SCHEMA

Al igual que las DTD, los esquemas describen la estructura de la información. El motivo de la creación de este nuevo estándar para realizar la labor de las DTD es, básicamente, la utilidad. El descubrimiento de nuevas aplicaciones de XML forzó la creación de una nueva solución que proporciona un nuevo mecanismo para modelado de datos, esa solución son los esquemas o *Schemas*. Los esquemas indican los tipos de dato, número mínimo y máximo de ocurrencias y otras características más específicas. Un XML *Schema* es algo similar a una DTD, es decir, que define qué elementos puede contener un documento XML, como están organizados, y que atributos y de qué tipo pueden tener sus elementos.

DTD vs XML Schema

Las limitaciones principales que se han encontrado en las DTD y que hizo necesario la aparición de XML *Schema* son:

- Posee un lenguaje propio de escritura, lo cual deriva en problemas a la hora de:

  1. El aprendizaje: no sólo hay que aprender XML además hay que aprender el lenguaje de las DTD.

[18] Del inglés "character data".

2. Procesado del documento: las herramientas y analizadores sintácticos que se empleen para tratar los documentos de XML deben ser capaces de procesar las DTD.

- No permite el uso de espacios de nombre (***NameSpaces***[19]), siempre y cuando se anteponga un prefijo al nombre del elemento.
- Tiene una definición de tipo de datos del documento extremadamente limitado. No permite definir el que un elemento pueda ser de un tipo numeral o de un tipo de fecha, sólo presenta variaciones limitadas sobre cadenas de texto.
- El mecanismo de extensión es complejo y frágil, además está basado en sustituciones sobre cadenas. Lo peor de dichas extensiones es que realmente no hace explicitas las relaciones en ningún momento, es decir, dos elementos que tienen definido el mismo tipo de contenido no presentan ninguna relación.

Estas reglas son superadas en la especificación de XML *Schema,* permitiendo un lenguaje mucho más expresivo. Aparte de la expresividad, la presencia de los esquemas permite un intercambio de información mucho más robusto. Además de resolver los problemas antes expuestos, XML *Schema* permite una serie de ventajas adicionales que se consideraron importantes:

- Usan la misma sintaxis de XML, al contrario que los DTD. Los esquemas están realizados también con lenguaje XML lo cual es mucho mas flexible y no requiere aprender otro lenguaje. Mientras que las DTD están realizadas con su propia sintaxis lo cual requiere aprender todo un nuevo lenguaje.
- Permite tipos definidos por el usuario dándoles un nombre y que se pueden emplear en distintas partes dentro del *Schema.*
- Es posible agrupar atributos, haciendo más comprensible el uso de un grupo de aspectos de varios elementos distintos, pero con denominador común, que deben ir juntos en cada uno de estos elementos.
- Sin embargo, la característica que más resalta la utilidad de XML *Schema* es la posibilidad de extender tipos definidos por el usuario de un modo específico. Es decir, permite definir nuevos tipos de datos de otros ya definidos. Con esto, se tendrán datos muchos mas especializados y específicos a la aplicación.
- Permiten especificar los tipos de datos. Permite definir los tipos de datos de una forma mas detallada. Además, maneja mucho más tipos de datos predefinidos. Las DTD a todos los datos los maneja como de tipo CDATA o de texto plano. Un esquema o *Schema* permite definir el tipo del contenido de un elemento o de un atributo, y especificar si debe ser un número entero, o una cadena de texto, o una fecha, etc. Se definen los tipos base que se

[19]Los **NameSpaces** permiten definir elementos con igual nombre dentro del mismo contexto.

pueden emplear dentro de esquema de XML, por ejemplo: *integer, boolean, string, date,* etc. Los DTD no permiten hacer estas definiciones.

1.2.3. Las DTD a profundidad

La DTD define los tipos de elementos, atributos y entidades permitidas y puede expresar algunas limitaciones para combinarlos. La figura 1.1 permite observar para que sirve una DTD en un documento XML.

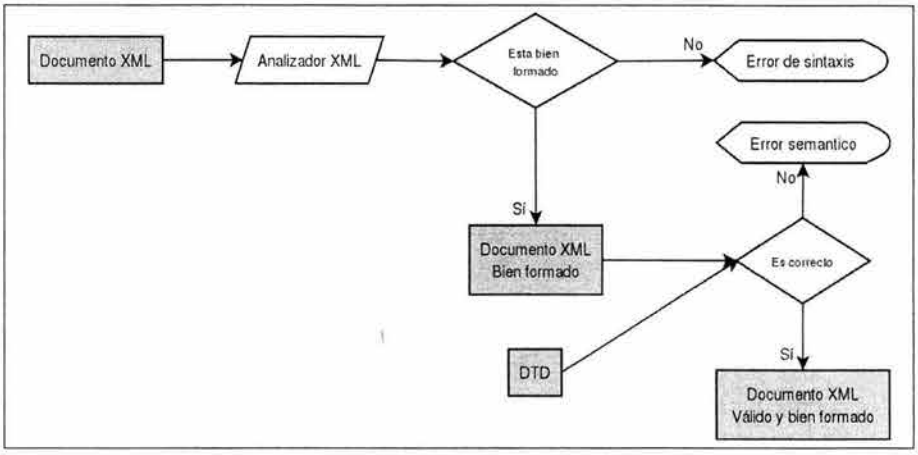

Figura 1.1: Proceso de validación de un documento XML

Los documentos XML que se ajustan a su DTD, se denominan "válidos" o "correctos". Un documento "bien-formado" simplemente respeta la estructura y sintaxis definidas por la especificación de XML. Un documento "bien-formado" puede además ser "válido" o "correcto" si cumple las reglas de una DTD determinada. También existen documentos XML sin una DTD asociada, en ese caso no son "válidos", son simplemente "bien-formados".

La DTD puede residir en un archivo externo y quizá compartido por varios (puede que miles) de documentos. Por ejemplo, en la figura 1.2 se tiene una DTD para libretas de direcciones llamada AddressBook.dtd:

```
<?xml encoding="UTF-8"?>
<!ELEMENT addressbook (contacto)+>
<!ELEMENT contacto (nombre, calle, ciudad, pais, codigo)>
<!ELEMENT nombre (#PCDATA)>
<!ELEMENT calle (#PCDATA)>
<!ELEMENT ciudad (#PCDATA)>
<!ELEMENT pais (#PCDATA)>
```

Figura 1.2: DTD de libreta de direcciones AddressBook.dtd

La DTD puede también estar contenida en el propio documento XML como parte de su declaración de tipo de documento.

En la figura 1.3 se presenta un ejemplo de una DTD:

```
<!DOCTYPE contacto[ <!ELEMENT contacto (nombre, calle, ciudad, pais)>
<!ELEMENT nombre (#PCDATA)>
<!ELEMENT calle (#PCDATA)>
<!ELEMENT ciudad (#PCDATA)>
<!ELEMENT pais (#PCDATA)>
<contacto>
    <nombre>Anibal J. Avelar </nombre>
    <calle>Taxqueña, 27</calle>
    <ciudad>Distrito Federal</ciudad>
    <pais>México</pais>
</contacto>
```

Figura 1.3: DTD de libreta de direcciones incrustada en el documento XML

La declaración del tipo de documento comienza en la primera línea y termina con "]>". Las declaraciones DTD son las líneas que empiezan con "<!ELEMENT" y se denominan declaraciones de tipo elemento. También se pueden declarar atributos, entidades y anotaciones para una DTD.

Un ejemplo de una documento XML que utiliza una DTD externa se observa en la figura 1.4.

```
<?xml version="1.0"?>
<!DOCTYPE coche SYSTEM "http://www.sitio.com/dtd/coche.dtd">
<coche>
    <modelo>...</modelo>
    ...
</coche>
```

Figura 1.4: Documento XML que hace una referencia a una DTD externa

Declaraciones tipo elemento

Los elementos son la base de las marcas XML y deben ajustarse a un tipo de documento declarado en una DTD para que el documento XML sea considerado válido. Las declaraciones de tipo de elemento deben empezar con "<!ELEMENT" seguidas por el identificador genérico del elemento que se declara. A continuación tienen una especificación de contenido. Por ejemplo:

```
<!ELEMENT receta (titulo, ingredientes, procedimiento)>
```

En este ejemplo, el elemento **<receta>** puede contener dentro elementos **<titulo>**, **<ingredientes>** y **<procedimiento>**, que, a su vez, estarán definidos también en la DTD y podrán contener más elementos. Siguiendo la definición de elemento anterior, el ejemplo de la figura 1.5 sería un documento XML válido:

```
<receta>
    <titulo>...</titulo>
    <ingredientes>...</ingredientes>
    <procedimiento>...</procedimiento>
</receta>
```

Figura 1.5: Documento XML válido

Sin embargo, un ejemplo de un documento no válido se puede observar en la figura 1.6, el documento no es válido porque define un elemento **<parrafo>** no definido en la DTD del documento:

```
<receta>
    <parrafo>Esto es un párrafo</parrafo>
    <titulo>...</titulo>
    <ingredientes>...</ingredientes>
    <procedimiento>...</procedimiento>
</receta>
```

Figura 1.6: Documento XML no válido según su DTD

La especificación de contenido puede ser de cuatro tipos:

- **Empty:** Puede no tener contenido. Suele usarse para los atributos.

```
<!ELEMENT salto-de-pagina EMPTY>
```

- **Any:** Puede tener cualquier contenido. Puede contener datos de carácter o sub-elementos.

```
<!ELEMENT batiburrillo ANY>
```

- **Mixed:** Puede tener caracteres de tipo datos o una mezcla de caracteres y sub-elementos especificados en la especificación de contenido mixto. En el ejemplo, el primer elemento definido (**<enfasis>**) puede contener datos de carácter (#PCDATA). Y el segundo (**<parrafo>**) puede contener tanto datos de carácter (#PCDATA) como sub-elementos de tipo **<enfasis>**.

```
<!ELEMENT enfasis (#PCDATA)>
<!ELEMENT parrafo (#PCDATA|enfasis)*>
```

- **Element:** Sólo puede contener sub-elementos especificados en la especificación de contenido.

```
<!ELEMENT mensaje (remite, destinatario, texto)>
```

Para declarar que un tipo de elemento tenga contenido de elementos se especifica un modelo de contenido en lugar de una especificación de contenido mixto o una de las claves ya descritas.

Modelos de contenido

Un modelo de contenido es un patrón que establece los sub-elementos aceptados y el orden en que se acepta. Un modelo sencillo puede tener un solo tipo de sub-elemento:

```
<!ELEMENT aviso (parrafo)>
```

Esto indica que **<aviso>** sólo puede contener un solo **<parrafo>**.

```
<!ELEMENT aviso (titulo, parrafo)>
```

La coma, en este caso, denota una secuencia. Es decir, el elemento **<aviso>** debe contener un **<titulo>** seguido de un **<parrafo>**.

```
<!ELEMENT aviso (parrafo | grafico)>
```

La barra vertical "|" indica una opción. Es decir, **<aviso>** puede contener o bien un **<parrafo>** o bien un **<grafico>**. El número de opciones no está limitado a dos, y se pueden agrupar usando paréntesis.

```
<!ELEMENT aviso (titulo, (parrafo | grafico))>
```

En este último caso, el **<aviso>** debe contener un **<titulo>** seguido de un **<parrafo>** o de un **<grafico>**. Además, cada partícula de contenido puede llevar un indicador de frecuencia, que siguen directamente a un identificador general, una secuencia o una opción, y no pueden ir precedidos por espacios en blanco. La tabla 1.2 muestra los indicadores de frecuencia de los identificadores.

Indicadores de frecuencia	
?	Opcional (0 o 1 vez)
*	Opcional y repetible (0 o más veces)
+	Necesario y repetible (1 o más veces)

Tabla 1.2: Indicadores de frecuencia de identificadores

Un ejemplo de los indicadores de frecuencia podría ser :

```
<!ELEMENT aviso (titulo?, (parrafo+, grafico)*)>
```

En este caso, **<aviso>** puede tener **<titulo>**, o no (pero sólo uno), y puede tener cero o más conjuntos **<parrafo><grafico>**, **<parrafo><parrafo><grafico>**, etc.

Declaraciones de lista de atributos

Los atributos permiten añadir información adicional a los elementos de un documento. Una diferencia entre los elementos y los atributos es que los atributos no pueden contener sub-atributos. Se usan para añadir información corta, sencilla y no estructurada. En la figura 1.7 se puede observar el uso de estos:

```
<mensaje prioridad="urgente">
    <de>Anibal Jesús </de>
    <a>Lidia Mendez</a>
    <texto idioma="ingles">Who are you? ... </texto>
</mensaje>
```

Figura 1.7: Atributos en un documento XML

Otra diferencia entre los atributos y los elementos, es que cada uno de los atributos sólo se puede especificar una vez y en cualquier orden.

Del ejemplo anterior, para declarar la lista de atributos de los elementos **<mensaje>** y **<texto>** en la DTD se haría lo que se observa en la figura 1.8:

```
<!ELEMENT mensaje (de, a, texto)>
<!ATTLIST mensaje prioridad (normal | urgente) normal>
<!ELEMENT texto(#PCDATA)>
<!ATTLIST texto idioma CDATA #REQUIRED>
```

Figura 1.8: Definición de atributos

Las declaraciones de los atributos empiezan con "<!ATTLIST" y a continuación del espacio en blanco viene el identificador del elemento al que se aplica el atributo. Después viene el nombre del atributo, su tipo y su valor por defecto. En el ejemplo anterior, el atributo "prioridad" puede estar en el elemento **<mensaje>** y puede tener el valor "normal" o "urgente", siendo "normal" el valor por defecto si no especificamos el atributo. El atributo "idioma" pertenece al elemento texto y puede contener datos de carácter (CDATA). La palabra "#REQUIRED" significa que no tiene valor por defecto, ya que es obligatorio especificar este atributo.

A menudo interesa que se pueda omitir un atributo, sin que se adopte automáticamente un valor por defecto. Para esto se usa la condición "#IMPLIED". Por ejemplo, en una supuesta DTD que defina la etiqueta **<IMG>** de HTML:

```
<!ATTLIST IMG URL CDATA #REQUIRED
              ALT CDATA #IMPLIED>
```

Es decir, el atributo "URL" es obligatorio, mientras que el "ALT" es opcional y si se omite, no toma ningún valor por defecto.

Atributos de datos de cáracter y de nombre de símbolo

Los atributos de datos de cáracter CDATA son los más sencillos y pueden contener diferentes tipos de datos. Los atributos nombre de símbolo (NMTOKEN[20]) son parecidos, pero sólo aceptan los caracteres válidos para nombrar cosas (letras, números, puntos, guiones, subrayados y los dos puntos).

```
<!ATTLIST mensaje fecha CDATA #REQUIRED>
<mensaje fecha="15 de Julio de 2003">
<!ATTLIST mensaje fecha NMTOKEN #REQUIRED>
<mensaje fecha="15-7-2003">
```

Atributos enumerados y notaciones

Los atributos enumerados son aquellos que sólo pueden contener un valor de entre un número reducido de opciones.

```
<!ATTLIST mensaje prioridad (normal | urgente) normal>
```

[20]Del inglés "name token".

En el ejemplo, el elemento **<mensaje>** tiene un atributo "prioridad" que puede tener los valores de "normal" o "urgente" y tiene un valor por defecto de "normal".

Un ejemplo para resumir todo lo visto hasta ahora podría ser una DTD que defina una base de datos de personas con direcciones *e-mail*. La DTD se puede observar en la figura 1.9:

```
<?xml encoding="UTF-8"?>
<!ELEMENT lista (persona)+>
<!ELEMENT persona (nombre, email*, relacion?)>
<!ATTLIST persona id ID #REQUIRED>
<!ATTLIST persona sexo (hombre | mujer) #IMPLIED>
<!ELEMENT nombre (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT relacion EMPTY>
```

Figura 1.9: Ejemplo de una DTD

Basándonos en este DTD, se puede describir un documento XML como en la figura 1.10:

```
<?xml version="1.0"?>
<!DOCTYPE lista SYSTEM "Lista.dtd">
<lista>
    <persona sexo="hombre" id="fixxxer">
        <nombre>Anibal Jesus Avelar Rosales</nombre>
        <email>afixxxer@yahoo.com.mx</email>
        <relacion amigo-de="lidy"/>
    </persona>
    <persona sexo="mujer" id="lidy">
        <nombre>Lidia Mendez Lecona</nombre>
        <email>lidyml@yahoo.com.mx</email>
    </persona>
</lista>
```

Figura 1.10: Documento XML que usa el DTD de la figura 1.9

Crear una DTD es como crear nuestro propio lenguaje de marcado para una aplicación específica. Para el editor XTE se debe definir una DTD para poder validar los documentos utilizados por el sistema. Se deben validar los documentos XML para garantizar que siguen siendo consistentes y coherentes. El editor XTE es parte de una aplicación colaborativa y un aspecto de la colaboración es la distribucion de los datos, por lo que se presenta el problema de la concurrencia sobre los documentos, este es un problema que puede presentarse con bastante frecuencia y puede generar inestabilidad, pérdida de consistencia en la información y que los documentos puedan perder el formato original. Por ese motivo, los documentos que utiliza el editor XTE siempre deben seguir siendo válidos y el uso de una DTD es indispensable. Más adelante se definirá el tipo de DTD que usará el editor.

1.3. Procesamiento de documentos con XML

1.3.1. Fundamentos del procesamiento XML

A un documento sólo se le puede llamar documento XML si se adhiere a las reglas de la especificación XML. Los creadores de esta especificación XML hicieron que la conformidad tuviera una

prioridad muy alta, porque no querían que los problemas que habían afectado a HTML también afectaran a XML. Una de las reglas más básicas que establecieron fue que si un documento no estaba bien construido, un programa cliente no debería seguir analizando sintácticamente ese documento de una manera normal.

Procesar un documento XML

Analizar sintácticamente un documento XML es el primer paso a la hora de procesarlo. Los analizadores XML son de dos tipos:

- El analizador estándar; que comprueba la buena construcción.
- El analizador de validación; que asegura que la estructura del documento es compatible con su DTD, si es que la hay.

Estos dos tipos de analizadores XML definen dos tipos principales de documentos XML:

- **Documentos válidos**: un documento XML válido está definido por una DTD que es la gramática del documento que define qué tipos de elementos, atributos y entidades podría haber en el documento. El DTD define el orden y también la ocurrencia de elementos. Si cumple con la DTD es un documento válido.
- **Documentos bien-formados**: Se comprueba su buena construcción y no tiene que adherirse a una DTD. Pero debe seguir dos reglas:

  1. debe haber un elemento raíz.
  2. todo elemento debe tener una etiqueta de apertura y otra de cierre.

  Todos los analizadores XML deben comprobar que la construcción sea correcta y si es así el documento será válido. Si un documento no está bien formado, el analizador no deberá seguir procesando el documento de una manera normal. Si hay una DTD y el analizador es de validación, éste también comprobará el contenido del documento frente a la DTD.

¿Por qué analizar sintácticamente XML?

Dado que un documento XML está diseñado para ser un documento autónomo, deberá contener todos los recursos que permitan a un procesador de XML procesarlo con éxito. Estos recursos son los elementos que hacen que un documento XML esté bien formado.

Se necesita analizar sintácticamente un documento XML (por medio de un procesador) para asegurarse de que está bien construido y si no lo está, no se continue con el resto del proceso.

¿Por qué validar un documento XML?

Un analizador de validación comprueba si un documento XML hace referencia a una DTD. Se necesita validar un documento para asegurarse cumple con una DTD y por lo tanto es un documento válido.

Clases de documentos

Si un documento es compatible con una determinada DTD, ese documento pertenecerá a una cierta clase de documentos definida por la propia DTD. Es posible usar la DTD y un analizador de validación para verificar que un determinado documento XML pertenece a la clase de documento a la que dice pertenecer. Cuando se confirma que el documento pertenece a una clase, se puede presuponer que tiene ciertas propiedades y un gran número de operaciones que se pueden realizar en él. Por ejemplo, la DTD puede utilizarse para llevar a cabo una validación rudimentaria del contenido, mientras que el documento puede utilizarse como paquete de contenido de transacciones electrónicas, mientras que el modelo de objeto de documento o DOM se puede usar para ordenar, cotejar y manipular el documento. EL DOM es una estructura de datos que representa la estructura lógica en forma de árbol de texto XML.

Utilizar una DTD con el DOM

Muchas de las implementaciones más interesantes de XML requieren la ordenación y la búsqueda de un documento XML para poder presentar ciertas vistas. Esto se hace con un lenguaje de programación para atravesar el árbol de documentos. Si el programa cliente puede estar seguro de la estructura del documento, su tarea será mucho más sencilla.

El analizador XML

Los analizadores son de dos tipos: los que forman un árbol de análisis y los que analizan sintácticamente un documento como estructura plana. El primer tipo es, con mucho, el más habitual. Aunque no sea tan rápido como el segundo tipo, es mucho más versátil y es extensible.

Árboles de análisis

En el dialecto de la programación orientada a objetos, todo documento XML puede considerarse una serie de objetos con ciertas propiedades. El documento es un objeto, todos los elementos individuales son objetos, los comentarios son objetos y todas las cadenas de texto también son objetos. Un árbol de análisis es una estructura que no sólo actúa como marco para estos objetos de documento, sino que también muestra como están relacionados entre sí.

Sintaxis de la relación

Para definir las relaciones que hay entre los objetos se utiliza una sintaxis conocida. Cada objeto (a excepción del objeto de documento) posee un objeto primario, y muchos objetos poseen objetos secundarios. Los objetos que comparten los mismos primarios se denominan iguales.

Sintaxis de los nodos

Mientras que una sintaxis basada en una familia es susceptible de describir los aspectos relacionales de un árbol de análisis, se utiliza el lenguaje de la biología para describir el propio árbol: Un elemento raíz, que es el elemento desde el que surge del resto de elementos. Cada uno de los objetos del árbol como nodo, que es el lugar donde se ramifica una planta. Los objetos finales del árbol (estas partes no tienen secundarios), que son las hojas. Luego se mezclan las dos sintaxis con lo que se tiene los nodos iguales y nodos secundarios y primarios.

Una lista de propiedades permite que el analizador reconstruya la totalidad del documento. La formación de esta lista es un preludio para la comprobación de que el documento está bien construido. En la tabla 1.3 se tiene la lista de nodos de árbol y sus propiedades.

Nodos de árbol	Propiedades
Node type	Comentario, declaración de versión de elemento, etc.
Node name	En el caso de que fuera un elemento, sería el nombre del elemento
Node value	En caso de que fuera texto o nodo de comentario, sería el contenido.
Parent node	El nodo primario
HasChildNodes	Una propiedad booleana que sirve para averiguar si un nodo tiene secundarios.
Child list	Una lista separada, probablemente otra matriz, de secundarios de un elemento.

Tabla 1.3: Nodos de árbol y propiedades

El analizador de validación

El primer paso a la hora de validar un documento consiste en construir un árbol de análisis y comprobar que está bien formado. Enseguida, cada elemento del árbol debe compararse con la DTD. Un analizador puede seguir dos estrategias cuando analiza sintácticamente un documento frente a una DTD:

- Construir una estructura lógica a partir de la DTD y luego hacer que coincida el documento frente a esta estructura lógica.
- Recorrer nuevamente el documento XML y hacer que coincida cada uno de los elementos individuales frente a la DTD.

La mayor parte de analizadores adopta la primera estrategia por ser la solución mas óptima y que ocupa menos tiempo de procesamiento. La segunda opción evidentemente es mas lenta sobre todo si la DTD a comparar es muy extensa.

Los analizadores como objetos

Hay dos formas de utilizar un analizador en una aplicación:

1. Para ver si un documento está bien formado.
2. Como parte de un procesador.

Si se está en el segundo caso (donde se está normalmente), el analizador podrá ser tratado como objeto. Con los analizadores basados en Java, esto significa que el analizador se incorporaría como clase en la aplicación Java. Con los analizadores basados en C o se puede incorporar el código fuente nativamente y acceder a las otras variables directamente o se puede tratar el analizador como una clase que tiene sus propios métodos y propiedades.

1.3.2. Analizar sintácticamente XML con Java

Desde que surge XML, los programadores de Java han estado en el centro de la comunidad de desarrolladores XML. Las dos tecnologías se complementan muy bien: XML por la transportabilidad de los datos y Java por la transportabilidad del software. Es más, hay una serie de razones por las que Java es especialmente recomendable a la hora de trabajar con XML:

- Los desarrolladores logran una gran productividad con Java. A menudo, es posible obtener implementaciones robustas de las nuevas tecnologías para que funcionen en Java dos veces más rápido de lo que se puede en C.
- Java siempre ha estado muy orientado a la Web, tanto en clientes como en servidores, y XML se dirige a la próxima generación de tecnología Web.
- El fuerte soporte que tiene Java en Unicode[21] cumple todos los requisitos del conjunto de caracteres en XML, por lo que los desarrolladores que no trabajen en inglés no tendrán ninguna desventaja.

En Java hay más herramientas XML gratuitas que en cualquier otro lenguaje de programación y suelen ser de una gran calidad. [JaXm]

Anatomía de las bibliotecas de analizadores XML en Java

Se habla de los analizadores XML como si se tratara de una sola cosa, pero no es así. En la práctica, los analizadores tienen muchas interfaces de programación de aplicaciones (API[22]) que la mayoría de las cuales hacen muchas mas funciones que analizar sintácticamente.

[21] Proyecto cuya finalidad es proporcionar una codificación única a cada caracter sin importar el sistema operativo o plataforma.

[22] Del inglés "Application Programming Interface".

En el centro de este tipo de librerías hay algo que la especificación XML llama un procesador XML y en la parte superior hay otros niveles de API. Estos niveles pueden soportar características como la validación de esquemas XML, motores de transformación XSLT, motores de consulta y de vinculación. Este conjunto de API principales es parte de la funcionalidad que Sun Microsystems (creador de Java) ha puesto en la extensión estándar de procesamiento XML de Java (JAXP[23]).

Funcionalidad Principal

La principal funcionalidad de todas las API para XML es que deben permitir escribir componentes de procesamiento que analicen sintácticamente el texto XML, procesen los datos en la memoria y escriban texto XML. Existen dos conjuntos de API entre los analizadores XML: SAX y DOM.

SAX y DOM son los modelos de procesamiento más comunes. Usar SAX para procesar un documento XML significa tener que codificar métodos para manejar eventos lanzados por el analizador según encuentra diferentes símbolos del lenguaje de marcas. Con DOM, se tiene que escribir código para pasar a través de una estructura de datos tipo árbol creada por el analizador desde el documento fuente. En la tabla 1.4 se observan algunas características de SAX y DOM.

SAX	DOM
Modelo basado en eventos	Estructura de datos tipo árbol
Acceso serie (flujo de eventos)	Acceso aleatorio (estructura de datos en memoria
Bajo uso de memoria (sólo se generan eventos)	Alto uso de memoria (todo el documento se carga en memoria)
Para procesar partes del documento (capturar eventos importantes)	Para editar el documento (procesar la estructura de datos en memoria)
Para procesar el documento sólo una vez (flujo de eventos temporal).	Para procesar el documento múltiples veces (documento cargado en memoria).

Tabla 1.4: Características SAX y DOM

API simple de XML (SAX[24])

Está disponible como una forma simple, rápida y sencilla de analizar sintácticamente texto XML. Es una API de analizador conducido por eventos que se ha desarrollado en un proceso abierto único antes de que la mayoría de fabricantes se interesara por las API Java para XML. Soporta una estructura muy pequeña, el procesamiento basado en eventos, y el DOM que representa el resultado de analizar en una estructura de datos de árbol que se puede usar con XML. La ventaja principal de SAX es que es ligero y rápido. Esto es principalmente porque es un API basado en eventos, lo que significa que reporta eventos de análisis (como el comienzo y el final de los elementos) directamente a la aplicación usando servicios repetidos, según lo mostrado en la Figura 1.11. Por lo tanto, la aplicación implementa manejadores para ocuparse de los diversos eventos, como el manejo de eventos en una interfaz gráfica de usuario.

[23] Del inglés "Java API for XML Processing".
[24] Del inglés "Simple API to XML".

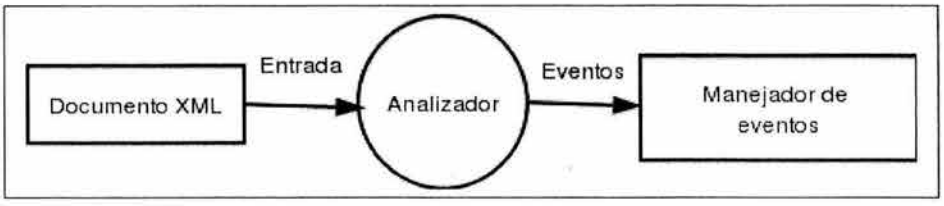

Figura 1.11: SAX usa retro-llamadas para notificar a los manejadores las cosas de interés

SAX define una interfaz *(org.xml.sax.Parser)* a la que obedecen todos los analizadores SAX. También define una clase con métodos estáticos *(org.xml.sax.helpers.Parser-Factory)* que permite que su aplicación cambie de analizadores muy fácilmente. De acuerdo con el modelo SAX, los analizadores deben ser fácilmente sustituibles. Algunas aplicaciones utilizan analizadores de validación y no validación en momentos diferentes. En la tabla 1.5 se tienen algunos nombres de clases de distintos API que implementan el analizador SAX.

Paquete del analizador	Nombre(s) de clase del analizador SAX
Java Project X	com.sun.xml.parser.Parser
	com.sun.xml.parser.ValidatingParser
IBM XML4J	com.ibm.xml.parsers.SAXParser
	com.ibm.xml.parsers.ValidatingSAXParser
ORACLE	oracle.xml.parser.v2.SAXParser
DataChannel	com.datachannel.xml.sax.SAXDriver
AElfred	com.microstar.xml.SAXDriver
Apache Xerces	org.xml.sax.Parser
XP	com.jclark.xml.sax.Driver

Tabla 1.5: Nombres de clase de los analizadores SAX

Modelo de objeto de documento (DOM[25])

DOM está disponible como forma opcional de representar texto XML en la memoria. Se trata de una estructura de datos que representa la estructura lógica en forma de árbol de texto XML. Esta API fue desarrollada a partir de la especificación propuesta por el W3C. DOM es una interfaz neutral a la plataforma y al lenguaje, para acceder y actualizar documentos. Sin embargo, al contrario que SAX, DOM accede a un documento XML a través de una estructura arborescente, compuesta por nodos elemento y nodos de texto (ver figura 1.12).

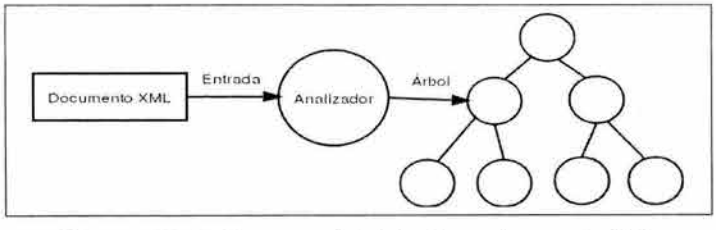

Figura 1.12: DOM crea un árbol desde un documento XML

[25]Del inglés "Document Object Model".

Existen dos situaciones en las que las aplicaciones necesitan obtener objetos *Document* DOM:

- Al trabajar con documentos existentes, las aplicaciones necesitan obtener una instancia *Document* que contenga la salida de un analizador XML: los nodos de los elementos, el texto y otros datos que se encuentran en ese documento XML.
- Al crear nuevos documentos desde cero, las aplicaciones necesitan obtener un documento vacío que se ira creando conforme se vaya necesitando.

1.3.3. Algunos analizadores disponibles para XML

Aunque hay muchos paquetes de análisis XML donde elegir, es posible que se encuentre útil investigar varias colecciones de recursos XML para obtener la última información. Existen varios sitios en Internet donde obtener recursos XML [xml, wdvl]. La presentación en estos sitios se centra en paquetes llenos de características, que incluyen soporte para analizadores DOM y SAX.

El paquete Java Project X de Sun MicroSystems (Jaxp)[26]

El paquete de Sun proporciona implementaciones de alta calidad de las herramientas XML: las API DOM, SAX y XML *Namespace*. Ofrece analizadores de validación y no validación y se sitúa entre los mejores en términos de conformidad, tal y como lo puntualizan pruebas independientes contra las especificaciones SAX, DOM y XML. JAXP no es un analizador, es un grupo de clases que se integran a cualquier otro analizador. Así no importará que analizador se use si este tiene soporte de JAXP.

El paquete XML4J V2 de IBM alphaWorks[27]

Tiene una serie de paquetes de software basado en XML. El paquete XML4J es uno de estos paquetes XML disponibles a través de alphaWorks. Ofrece analizadores de validación y de no validación, con soporte DOM y una serie de API originales para el acceso a las DTD y a otras funcionalidades.

El paquete XERCES de APACHE XML PROJECT[28]

Es desarrollado por la fundación Apache. Xerces es un analizador de validación completa. Está disponible en los lenguajes Java y C. Incluye soporte de analizadores SAX y DOM. Así como un conjunto de herramientas que permiten manipular los documentos XML.

---

[26] **http://java.sun.com/sml.** Paquete Java Project X, incluido el código fuente, ejemplos y la documentación.

[27] Proyecto de IBM de recursos XML. **http://www.alphaworks.ibm.com**. Paquete XML4J, incluido el código fuente y una cantidad relativamente amplia de información.

[28] **http://xml.apache.org/xerces2-j/index.html.** Paquete que incluye el código y los ejemplos van junto al paquete.

1.3.4. Implementaciones del DOM

El paquete Java Project X de Sun Microsystems (Jaxp)

La clase *com.sun.xml.tree.XmlDocument* implementa la interfaz *Document* del DOM y se puede invocar directamente a un constructor.

Análisis sintáctico

Existen varias formas de que este paquete cree un árbol *Document* DOM. Sólo se debe llamar a uno de los distintos métodos estáticos, así como opciones de configuración necesarias para realizar el análisis sobre el documento XML. Un análisis básico de no validación se puede observar en la figura 1.13:

```
import org.w3.dom.*;
import com.sun.xml.tree.XmlDocument;
. . .
try{
    Document doc;
// paso único: ¡obtener el documento!
    doc = XmlDocument.createXmlDocument (documentURI, false);
// operar en la estructura de datos del DOM
}catch (Exception e){
    e.printStrackTrace();
}
```

Figura 1.13: Obtener un objeto *Document* DOM con Jaxp

El parámetro "false" del método significa "no validar". Si se hubiera querido validar ese documento, tendría que haberse pasado "true". También se puede proporcionar un *InputSource* SAX en vez de un URI.

El paquete XML4J v2 de IBM

La clase *com.ibm.xml.dom.DocumentImpl* implementa la interfaz *Document* del DOM, y es posible invocar a un constructor directamente sobre ella. Los nodos de este DOM se pueden serializar. La salida serializada es mucho más grande que el texto XML equivalente, ya que incluye las estructuras de datos binarios que se usan para representar todo el documento DOM en la memoria.

ANÁLISIS SINTACTICO

El paquete de IBM tiene dos analizadores DOM, uno que valida *(com.ibm.xml.parsers.DomParser)* y otro que no *(com.ibm.xml.parsers.NonValidingDOMParser)*.

En la figura 1.14 se puede observar como se obtiene un árbol DOM de un documento XML sin validarlo:

```
import com.ibm.xml.parsers.NonValidatingDOMParser;
import org.w3.dom.Document;
...
try {
    NonValidatingDOMParser parser;
    Document doc;
// Configurar. En este caso, no se establecen propiedades JavaBeans
    Parser = new NonValidatingDOMParser ();
// Listo. Analizar sintacticamente y recoger el documento DOM resultante
    parser.parse (documentURI);
    doc = parser.getDocument ();
// Operar sobre la estructura de datos del DOM
} catch(Exception e){
    e.printStrackTrace();
}
```

Figura 1.14: Obtener un objeto *Document* DOM con XML4J

El paquete Xerces de APACHE XML PROJECT

La clase *javax.xml.parsers.DocumentBuilder* implementa la interfaz *Document* del DOM. Se utiliza una fabrica *DocumentBuilderFactory* que permite construir estáticamente varias instancias de esa clase.

Análisis sintáctico

Se crea un fábrica de objetos tipo *Document*. Todo el documento se guarda en memoria y es posible modificar el árbol de nodos DOM. Si existe un error de validación, se produce una excepción de *ParserConfigurationException*. Este analizador es de validación, pero puede eligirse entre validar o no lo haga, lo cual lo hace muy flexíble. En la figura 1.15 se muestra como obtenerlo:

```
import javax.xml.parsers.*;
public static DocumentBuilder builder;
private static DocumentBuilderFactory docFactory;
static {
    try {
        //Se crea la fabrica de Documents
        docFactory = DocumentBuilderFactory.newInstance();
        //Se obtiene un constructor 'Document' de la fábrica
        builder = docFactory.newDocumentBuilder();
    }
    catch(ParserConfigurationException pce){
        pce.printStackTrace();
    }
}
...
//Tenemos listo nuestro documento
doc = builder.parse(source);
...
//Aquí se procesa el documento
```

Figura 1.15: Obtener un objeto *Document* DOM con Xerces

1.4. Resumen

Se vio cómo es la estructura de los documentos XM y cómo se modelan los datos para poder escribir documentos XML que estén bien construidos y sean válidos. Se observo que las DTD ayudan a validar documentos DTD y cómo escribir documentos XML que cumplan con una DTD correspondiente, así como los tipos de datos que puede reconocer una DTD. Se mencionó qué significa procesar un documento XML y cómo procesarlo por medio de un analizador sintáctico. Se describió la diferencia entre un analizador SAX y un analizador DOM. Se listaron varios API de analizadores sintácticos disponibles que se pueden usar para manipular los documentos XML, varios de ellos soportan analizadores SAX y DOM, algunos son de validación (validan con respecto a una DTD definida) y otros no lo son (sólo validan que este bien construido el documento).

Todas estas herramientas son utilizadas dentro del editor XTE para validar los documentos XML. El formato XML es ideal para el editor XTE, porque su estructura granular (representado por una raíz y varios nodos) permite representar al documento dividido en varios fragmentos. Donde cada fragmento del documento puede ser representado en una relación uno a uno por un nodo en el formato XML. Esto es perfecto para el comportamiento colaborativo del editor, ya que para lograr la edición síncrona y concurrente de los documentos se necesita que los usuarios puedan editar el mismo documento en el mismo intervalo de tiempo pero desde lugares distintos. De esta manera, la capacidad granular del formato XML se aprovecha para representar los documentos fragmentados que hace uso el editor XTE del proyecto ELXI.

Otra ascpecto que necesita el editor XTE es poder validar los documentos con una DTD que represente al tipo de documento que se usará en el editor colaborativo. Esta DTD permitirá validar los documentos y de esta manera garantizar que los documentos son del mismo tipo y mantienen su integridad. Ya que los documentos tendrán que ser compartidos y ser distribuidos a través de la red, es necesario mantengan su integridad y consistencia.

Capítulo 2

EL MODELO DE OBJETO DE DOCUMENTO

Este capítulo trata sobre el Modelo de Objeto de Documento (DOM). También de como leer, recorrer y generar un documento XML por medio del lenguaje de programación Java. También describe cómo manipular los elementos de un árbol XML por medio del DOM de Java Xerces.

2.1. El modelo de objeto de documento

El DOM es el medio por el que se puede acceder y manipular un documento XML. Sin el DOM, XML no sería más que un sistema de almacenamiento para datos a los que se accede por distintos métodos. Con el DOM, XML es un lenguaje de programación universal e independiente de la plataforma y la aplicación.

Un problema con todos los estándares (como XML) es que las aplicaciones que implementan los estándares tienden a producir sus propios métodos para manipularlos. Se vio esto en HTML, donde los navegadores de Web utilizaban sintaxis y métodos distintos para manipular dinámicamente un documento HTML. En efecto, esto significaba que los autores que querían usar el DOM tuvieron que escribir páginas y programas separados de los distintos programas cliente o utilizar una mínima implementación del DOM.

Un DOM bien implementado debe permitir la reconstrucción del documento en su totalidad desde el propio modelo y debe permitir el acceso a cualquiera de las partes del documento. Como añadido, el DOM debe tener una serie de métodos que permitan hacer manipulaciones, adiciones y eliminaciones en el documento original. Afortunadamente el W3C ha creado la recomendación de DOM que permite hacer todo esto.

EL W3C Y EL DOM XML

El W3C tiene un grupo de trabajo que se preocupa de la construcción del DOM para HTML y XML. Aunque su versión actual no está terminada en algunos puntos, el DOM permite acceder a todas las partes del cuerpo de un documento XML.

El DOM se ha ido formulando progresivamente en niveles de implementación y se ha dividido en módulos a fin de concretar objetivos. Actualmente las especificaciones disponibles son las siguientes[1]:

- Nivel 1
  - Núcleo
  - HTML
- Nivel 1 (Segunda Edición)
  - Nucleó (Borrador)
  - HTML (Borrador)
- Nivel 2
  - Núcleo
  - Vistas
  - Eventos
  - Estilo
  - Navegación y Rango
  - HTML
- Nivel 3
  - Núcleo
  - Cargar y guardar
  - Validación
  - Vistas y formateo (nota para recomendación)
  - Abstract Schemas (nota para recomendación)
  - Eventos (nota para recomendación)
  - XPath (nota para recomendación)

[1] Estos datos fueron recogidos del sitio oficial del W3C al mes de Agosto de 2004.

Los modulos que aún permanecen como nota para recomendación por parte del W3C, significa que están en proceso de ser oficialmente una recomendación y con esto adquirir un reconocimiento de especificación estandar. Sólo falta terminar de redactar la propuesta. En el caso de las que están como borrador, significa que no tienen siquiera un propuesta para recomendación y apenas se está conformando la especificación final.

La interfaz API del DOM

En el mundo de la programación orientada a objetos, una API DOM común permite a los programadores escribir código que puede interpretar documentos que se ejecutan en cualquier aplicación o plataforma. Si la aplicación soporta el mismo DOM, no tendrá importancia si el programa está escrito en lenguajes como Visual Basic, C o Java, ya que expondrán al programador la misma serie de atributos y métodos.

Los modelos de Objeto de Documento

Hay varios tipos de DOM. Un DOM puede ser completo, permitiendo el modelado y el acceso a los mínimos detalles del documento. También puede ser incompleto, permitiendo sólo un acceso parcial al documento. El DOM del W3C constituye un ejemplo del primero, mientras que el DOM de JavaScript para HTML es un ejemplo del último.

El modelo también puede seguir uno de los tres patrones comunes:

- **Modelos lineales**. Es el modelo más sencillo, detallando el documento de un modo lineal es un modelo lineal. Sin embargo, este modelo tiene una desventaja muy grande, ya que toda la alteración en la primera parte del documento invalidaría cualquiera de las referencias de la última parte del documento.
- **Modelos de árbol**. Describen los documentos con los términos de un árbol. El modelo de árbol habla de la raíz del documento. Cada uno de los elementos del árbol se llama nodo y cada elemento final se suele denominar hoja. Un nodo primario hace referencia a un nodo inmediatamente anterior. Los nodos ascendentes hacen referencia a los nodos anteriores. Un nodo igual es un nodo que comparte el mismo primario. Los modelos de árbol no son tan sensibles como para cambiar a modelos lineales. Sin embargo, siguen siendo algo sensibles, ya que si se elimina un nodo completo, las relaciones numéricas que haya entre los nodos cambiarán y todas las listas de nodos deberán reconstruirse.
- **Modelos de objeto**. Son los menos sensibles a los cambios. Con un modelo de objeto, cada sección de un documento tendrá una propiedad con nombre. Independientemente de como se cambie el documento, la descripción de este objeto seguirá siendo aplicable.

Árboles de documentos y árboles de análisis

Para ver como funciona un árbol de análisis, se tiene el siguiente código XML en la figura 2.1:

```
<?xml version="1.0"?>
<!– Demostración de un árbol de análisis –>
<xdoc>
    <greeting>Hello <emph>parsed</emph>XML!!</greeting>
    <applause type="sustained"/>
</xdoc>
```

Figura 2.1: Demostración sencilla de un árbol de análisis

La raíz del árbol es el propio documento. Esta raíz posee tres nodos secundarios:

- Un nodo de declaración de la versión en dialecto de DOM.
- Un nodo de comentario. Este es ignorado y solo es informativo para el documento.
- Un nodo de elemento. Este nodo de elemento se llama nodo raíz del documento.

En la figura 2.2 se tiene el diagrama de árbol del código XML de la figura 2.1:

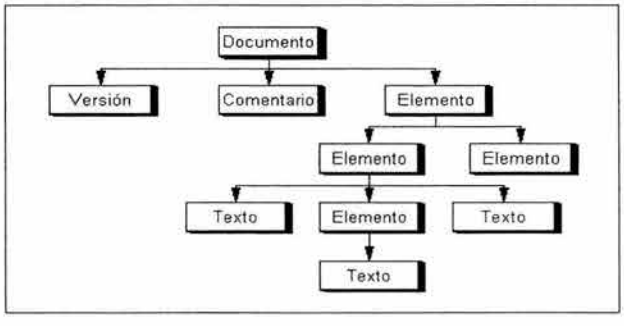

Figura 2.2: Diagrama de árbol del documento XML

En la figura 2.2 se observa que el documento XML de la figura 2.1 tiene 3 nodos principales: un nodo de versión el cual todos los documentos XML bien construidos deben tener, un elemento de comentario que pudo haber ido en cualquier parte del documento y el elemento raíz *<xdoc>*. Del elemento raíz se desprenden dos elementos *<greeting>* y *<applause>*, del elemento *<applause>* se desprende el elemento *<emph>* y dos cadenas de texto "Hello" y "XML!!". Por último, del elemento *<emph>* deriva la cadena de texto "parsed". Al elemento *type* no se le considera dentro del árbol ya que no es un elemento sino un atributo del elemento *<applause>*.

Objetos DOM

Todo objeto que soporte el DOM debe ser capaz de reconocer toda la estructura del documento XML. Además, debe exponer todas las interfaces con los atributos y métodos apropiados de acuerdo con la especificación del DOM. La aplicación con la que se conecte deberá ser capaz de acceder a documentos XML enteros utilizando la sintaxis necesaria.

Interfaces del DOM

El DOM define varias interfaces, algunas de las cuales son bastante útiles.

- **DOMImplementation**. Es una consulta al propio objeto y es independiente del documento que se le cargue. Devuelve información acerca del nivel del DOM que soporta el objeto.
- **DocumentFragment**. Es una parte ligera del documento.
- **Document**. Esta interfaz proporciona información acerca del propio documento principal que se ha cargado en el objeto. Posee varios métodos para crear nodos y atributos XML.
- **Node**. Todo en un documento puede ser considerado un nodo. Los elementos son nodos, los comentarios son nodos, y así sucesivamente. Esta interfaz contiene varios atributos generales apropiados para manipular cualquier tipo de nodo. De forma poco sorprendente, los métodos y atributos del nodo se solapan con los de las demás interfaces. Por ejemplo, al tratar con un elemento, el atributo del nodo *nodeName* devolverá el mismo valor que el atributo del elemento *tagName.*
- **NodeList**. Es un conjunto ordenado de nodos que se puede acceder por medio de un índice.
- **NamedNodeMap**. Es una colección de nodos a los que se puede acceder por nombre. Los atributos de un elemento se almacenan como *NamedNodeMap.*
- **CharacterData**. Es una interfaz que trata sobre los datos de caracteres o con el texto de un documento.
- **Attr**. Esta interfaz trata sobre los atributos XML de un nodo. La versión abreviada *Attr* se usa para evitar la confusión con un atributo.
- **Element**. La mayoría de los nodos de un documento XML son elementos. Esta interfaz posee propiedades y métodos que sirven para tratar con los elementos y sus atributos XML.
- **Text**. Trata sobre el contenido de texto de un elemento. La mayor parte de la funcionalidad que requiere la maneja la interfaz *CharacterData.*

Nodos y Objetos

El DOM del W3C utiliza una referencia doble para describir los objetos de documento. Todo lo que hay en el modelo de objeto es:

- Un nodo.
- Un tipo de objeto con nombre.

Los nodos y los objetos están representados en el árbol de documentos. El nodo *Attr* es un tipo de nodo que no está representado en el árbol, pero es igualmente un nodo. El nodo *Attr* representa el atributo XML de un nodo de elemento. El nodo *Attr* no forma parte de la estructura de árbol. En otras palabras, no se le considera un nodo secundario de ningún nodo. En vez de ello, se le considera una propiedad de un nodo *Element*. Se accede al contenido de un nodo *Attr* por medio de atributos y métodos de la interfaz *Element*.

2.2. Leer un documento XML

Hay dos formas de acceder a un nodo en el DOM. Se puede recorrer el árbol o acceder al nodo directamente por el nombre. En la práctica, se accede a un documento por medio de una combinación de ambos métodos.

Recorrer el árbol

Para recorrer el árbol DOM, se empiza en cualquier parte del árbol y se utilizan los siguientes métodos de la interfaz de nodo del DOM:

- **parentNode().** Este método accede al nodo primario.
- **firstChild().** Este método accede al primer secundario del nodo. Si no existiera este tipo de nodo, se devolvería *null*.
- **nextSibling().** Este método accede al siguiente nodo igual. Si no existiera tal nodo, se devolvería *null*.
- **previousSibling().** Este método accede al nodo igual anterior. Si no existiera este tipo de nodo, se devolvería *null*.

En la práctica estos métodos deben bastar para poder recorrer el árbol de análisis de DOM del documento XML.

Acceder a los nodos por su nombre

Se utiliza el método *getElementsByTagName* para obtener una lista de todos los elementos que tengan ese nombre en el documento o en cualquier parte del documento. Cuando se crea la *nodeList*, el índice podrá acceder al nodo con nombre.

2.3. Generación y recorrido de un árbol de XML

Para poder trabajar con DOM se necesita un analizador DOM. El analizador DOM lee un documento XML y genera una estructura en árbol para él. Hasta ahora no se ha usado ningún API de algún lenguaje en particular, simplemente se ha usado la especificación DOM que el W3C recomienda. Sin embargo, a partir de ahora se utilizara el analizador Java DOM del Apache Xerces.

En la figura 2.3 se observa un ejemplo sencillo realizado con Java y el API Apache Xerces. Este lee un documento XML y genera un árbol DOM para él:

```
import org.w3c.dom.Document;
import java.io.IOException;
import org.xml.sax.SAXException;
import org.apache.xerces.parsers.DOMParser;
public class DOM {
    public static void main(String[] args) {
      if (args.length != 1) {
        System.out.println("Debe haber un parámetro (el archivo)");
      }
      else {
        String arch = args[0];
        System.out.println("Generando árbol XML para: " + arch + "\n");
        try {
            DOMParser parser = new DOMParser();
            parser.parse(arch);
            Document doc = parser.getDocument();
            System.out.println("Fin de la carga. Todo bien \n");
        }
        catch (IOException e) {
            System.out.println(e);
        }
        catch (SAXException e) {
            System.out.println(e);
        }
      }
    }
}
```

Figura 2.3: Lectura de un documento XML con Java Xerces

En el ejemplo se necesita una instancia del analizador (clase DOMParser). El proceso de generación del arbol es:

- Para generar el árbol llamamos al método *parse*, pasándole cualquier archivo.
- Después, con el método *getDocument* obtenemos un objeto que referencia al árbol creado y que se puede manipular de diversas formas.
- Si hay algún problema con la sintaxis del documento XML se genera una excepción del tipo *SAXException* y el proceso se detiene.

2.4. Manipular los elementos XML

Para ilustrar el manejo de DOM se va a imprimir el documento XML que se pone como entrada del programa. Para ello se necesita realizar un procesamiento recursivo: se procesa cada nodo del árbol y después se procesan sus nodos hijos recursivamente. Se añade un método **tratarNodo** que va a ser llamado recursivamente. Dentro de este método se realizarán dos tareas:

- Determinar el tipo del nodo en curso.
- Tratar recursivamente los nodos hijos del nodo en curso.

La primera tarea se realiza con el método *getNodeType,* como puede verse en el siguiente esquema de código de la figura 2.4:

```
import org.w3c.dom.Document;
import org.w3c.dom.Node;
import java.io.IOException;
...
public class DOM {
    public static void tratarNodo(Node nodo) {
      switch (nodo.getNodeType()) {
        case Node.DOCUMENT_NODE:
             break;
        case Node.ELEMENT_NODE:
             break;
        case Node.ATTRIBUTE_NODE:
             break;
        case Node.TEXT_NODE:
             break;
        case Node.CDATA_SECTION_NODE:
             break;
        case Node.PROCESSING_INSTRUCTION_NODE:
             break;
        case Node.ENTITY_REFERENCE_NODE:
             break;
        case Node.DOCUMENT_TYPE_NODE:
             break;
      }
    }
    public static void main(String[] args) {
      ...
       Document doc = parser.getDocument();
       tratarNodo(doc);
       System.out.println("Fin de la carga. Todo bien \n");
       ...
    }
}
```

Figura 2.4: Programa que lee e imprime el documento XML

El primer nodo que se encuentra representa todo el documento (tipo *DOCUMENT_ NODE*). Ahí se imprimirá la declaración XML. Después con *getDocumentElement* se accede al nodo que representa al elemento raíz y lo se pasa recursivamente al método **tratarNodo**.

En la figura 2.5 se puede observar:

```
case Node.DOCUMENT_NODE:
    System.out.println("<xml version=\"1.0\">");
    Document doc = (Document)nodo;
    tratarNodo(doc.getDocumentElement());
break;
```

Figura 2.5: Cuando encuentra nodo tipo DOCUMENT_NODE

Obtener los elementos

Los elementos del documento están representados por un nodo *ELEMENT_ NODE.* Se obtiene el nombre de cada elemento con *getNodeName.* Con *getChildNodes* se obtienen los hijos en una *NodeList,* que hay que recorrer para tratar recursivamente cada hijo (ver figura 2.6).

```
import org.w3c.dom.*;
import java.io.IOException;
...
case Node.ELEMENT_NODE:
    String nombre = nodo.getNodeName();
    System.out.print("<" + nombre);
    System.out.println(">");
    NodeList hijos = nodo.getChildNodes();
    if (hijos != null) {
        for (int i=0; i<hijos.getLength(); i++) {
            tratarNodo(hijos.item(i));
        }
    }
System.out.println("</" + nombre + ">");
break;
```

Figura 2.6: Cuando encuentra nodo tipo ELEMENT_NODE

Obtener los atributos

Para obtener los atributos existe el método *getAttributes,* que devuelve un *NamedNodeMap* el cual se procesa para tratar los atributos recursivamente (ver figura 2.7). Cada atributo es un nodo *ATTRIBUTE_ NODE,* cuyo nombre y valor se recupera con *getNodeName* y *getNodeValue.*

```
import org.w3c.dom.*;
case Node.ELEMENT_NODE:
    String nombre = nodo.getNodeName();
    System.out.print("<" + nombre);
    NamedNodeMap ats = nodo.getAttributes();
    for (int i=0; i<ats.getLength(); i++) {
        tratarNodo(ats.item(i));
    }
    System.out.println(">");
    NodeList hijos = nodo.getChildNodes();
    ...
break;
case Node.ATTRIBUTE_NODE:
    System.out.print(" " + nodo.getNodeName() + "=\"" + nodo.getNodeValue() + "\"");
```

Figura 2.7: Cuando encuentra nodo tipo ATTRIBUTE_NODE

Obtener el texto

El texto de los elementos viene en nodos *TEXT_ NODE* o *CDATA_ SECTION_ NODE,* en los cuales sólo se tiene que hacer una llamada a *getNodeValue* para obtener el texto (ver figura 2.8).

```
case Node.TEXT_NODE:
    String texto = nodo.getNodeValue().trim();
    if (!texto.equals("")) {
        System.out.println(ind + texto);
    }
    break;
case Node.CDATA_SECTION_NODE:
    System.out.println(nodo.getNodeValue());
    break;
```

Figura 2.8: Cuando encuentra nodo tipo TEXT_NODE o CDATA_SECTION_NODE

Obtener la DTD

Solo fálta el código para procesar las instrucciones de proceso, la declaración DTD y las referencias a entidades. En la figura 2.9 se observa como se procesa:

```
import org.w3c.dom.Document;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.DocumentType;
import java.io.IOException;
...
case Node.PROCESSING_INSTRUCTION_NODE:
    System.out.println("<?" + nodo.getNodeName() + " " + nodo.getNodeValue() +"?>");
    break;
case Node.DOCUMENT_TYPE_NODE:
    DocumentType dt = (DocumentType)nodo;
    System.out.println("<!DOCTYPE " + dt.getName()+ " SYSTEM \"" + dt.getSystemId() + "\">");
    break;
case Node.ENTITY_REFERENCE_NODE:
    System.out.println(ind + "&" + nodo.getNodeName() + ";");
    break;
```

Figura 2.9: Se obtienen otros elementos como la DTD y las referencias a entidades

Se modifica el tratamiento del nodo del documento *(DOCUMENT_ NODE)* para que trate todos sus nodos hijos, ya que las instrucciones de proceso y la declaración DTD lo son (ver figura 2.10).

```
case Node.DOCUMENT_NODE:
    System.out.println("<xml version='1.0'>");
    Document doc = (Document)nodo;
    tratarNodo(doc.getDocumentElement(), "");
    NodeList nodos = nodo.getChildNodes();
    if (nodos != null) {
        for (int i=0; i<nodos.getLength(); i++) {
            tratarNodo(nodos.item(i), "");
        }
    }
break;
```

Figura 2.10: Se modifica el tratamiento del nodo DOCUMENT_NODE

2.4.1. Crear documentos XML con DOM

Hasta ahora se ha visto como manipular documentos XML que ya existen, pero ahora se creará un documento desde un inicio con el DOM. Primeramente, se necesita instanciar una clase concreta que implemente el interfaz *Document.* En el caso del API de Xerces es *DocumentImpl.* Los métodos más importantes para ir construyendo el documento son:

- ***createElement,*** crea un nodo para un elemento (Interfaz *Document*).
- ***setAttribute,*** crea un atributo para un elemento (Interfaz *Element*).
- ***createTextNode,*** crea un nodo de tipo texto (Interfaz *Document*).
- ***appendChild,*** para agregar un nodo hijo a un nodo padre, al final de su lista de nodos hijos (Interfaz *Node*).

Una vez que se ha creado el documento, se puede imprimir en un flujo (en pantalla, a un archivo, etc.). Se necesita instanciar la clase *XMLSerializer,* especificando el flujo por el que se va a escribir y un formato. Después se pasa el documento al método *serialize* (ver figura 2.12).

```
import java.io.*;
import org.w3c.dom.*;
import org.apache.xerces.dom.DocumentImpl;
import org.apache.xml.serialize.*;
public class CrearDOM {
    public static void main( String[] argv ) {
      try {
        Document doc= new DocumentImpl();
        Element raiz = doc.createElement("Persona");
        raiz.setAttribute("RFC", "AVRA731225");
        Element elem = doc.createElement("Nombre");
        elem.appendChild(doc.createTextNode("Anibal Avelar"));
        raiz.appendChild(elem);
        elem = doc.createElement("Edad");
        elem.appendChild(doc.createTextNode("29"));
        raiz.appendChild(elem);
        elem = doc.createElement("Altura");
        elem.appendChild(doc.createTextNode("1.83"));
        raiz.appendChild(elem);
        doc.appendChild(raiz);
        OutputFormat formato = new OutputFormat(doc, "UTF-8", true);
        StringWriter s = new StringWriter();
        XMLSerializer ser = new XMLSerializer(s, formato);
        ser.serialize(doc);
        System.out.println(s.toString());
        FileWriter f = new FileWriter("persona.xml");
        ser = new XMLSerializer(f, formato);
        ser.serialize(doc);
      } catch (IOException e) {
        System.out.println(e);
      }
    }
}
```

Figura 2.11: Crear un documento XML e imprimirlo en pantalla

De aquí, se puede observar como usando los metodos *createElement* (nodo) y *createTextNode* (nodo tipo texto) se puede crear nuevos nodos al árbol tantos como se desee. Al final, se obtiene

un árbol XML con el cual se puede hacer con él lo que se requiera (imprimirlo a la pantalla, guardarlo en un archivo, etc).

2.4.2. Modificar documentos

También se puede modificar un documento DOM por medio de los siguientes métodos definidos en *Node:*

- **insertBefore**, agrega un nodo hijo a otro nodo, permitiendo especificar la posición concreta en la lista de hijos (el hijo que va a quedar detrás del nuevo).
- **removeChild**, elimina un nodo hijo.
- ***replaceChild***, reemplaza un nodo hijo por otro.
- **removeAttribute**, elimina un atributo de un nodo (este método se define en *Element*).

Otras operaciones que se pueden aplicar a un objeto DOM, pueden ser:

- **Mover Nodos**

  El interface *Node* define un número de métodos que se pueden usar para mover nodos, incluyendo *getFirstChild, getLastChild, getNextSibling, getPreviousSibling* y *getParentNode.* Estas operaciones son suficientes para obtener algo de una posición del árbol y ponerlo en otra.

- **Crear Atributos**

  El interface *org.w3c.dom.Element* que extiende *Node,* define una operación *setAttribute* que añade un atributo a un nodo. También se pueden usar la operación *createAttribute* de *Document* para crear un ejemplar del *Attribute* y usar una versión sobrecargada de *setAttributes* para añadirlo.

2.5. Resumen

En este capítulo se describió el modelo de objeto de documento (DOM). También se describió como leer un documento XML por medio del lenguaje de programación Java y algunos ejemplos usando el API Xerces. Se vio como generar y recorrer un árbol XML. Se vieron muchos métodos que forman parte de la interfaz Java DOM del API Xerces. Todos estos métodos permiten manipular y controlar todo el árbol DOM generado a partir del documento XML. Muchos de estos métodos ayudarán más adelante a generar la aplicación del editor XML de transformación. Se vieron también, algunos ejemplos de como manipular los elementos de un árbol XML con el API Xerces.

También se describio el DOM de Xerces el cual permitirá manipular el tipo de documento XML que hará uso el editor colaborativo. De esta forma el editor XTE internamente contendrá un procesador de análisis sintáctico para validar el documento y generar un árbol DOM.

El editor permitirá las operaciones de un editor que funcione de forma colaborativa esto lo puede realizar dividiendo al documento en fragmentos donde cada uno de estos será editado de forma independiente y concurrente. El formato XML es ideal para ésta tarea al poder ser representados como árboles de análisis. De esta manera cada uno de los nodos o ramas del árbol será un fragmento independiente del documento que se podrá manipular por medio del DOM. Así se podrá crear, insertar, mover, quitar y borrar cada uno de los nodos y al hacerlo se estará modificando la estructura de cada uno de los fragmentos en el que está dividido el documento.

Se tendrá entonces un analizador DOM del lado del servidor ELXI para certificar que los documentos continuan siendo válidos y también un analizador del lado de los clientes ELXI (el editor XTE) para mantener igualmente la validez de los documentos, pero además la capacidad de poder controlar, manipular y interactuar con cada uno de los elementos visuales presentados al usuario (la representacion visual del documento dividido por fragmentos). El DOM da la capacidad de controlar en su totalidad a los documentos usados por el editor XTE.

Capítulo 3

FORMATO DE LOS DOCUMENTOS XML

XML se utiliza para marcar el contenido de un documento y mantener los datos de forma estructurada por lo que inicialmente no interesa como se muestra el contenido del documento en un navegador Web o en otra aplicación. Para poder visualizar los documentos XML, se debe especificar información como el tamaño de fuente y los colores asociados con los distintos documentos y atributos, lo cual se conoce como dar estilo al documento. El mecanismo general para aplicar estilos de formato a los documentos XML se conoce como hoja de estilos.

3.1. HTML, lenguaje de presentación visual muy básico

En realidad, HTML nunca fue diseñado para ser un lenguaje de presentación visual. De hecho, los lenguajes de marcado están inherentemente diseñados para describir la estructura de documentos por medio de metadatos[1], lo que no tiene nada que ver con el modo en que se presenta un documento. La idea es que se use el marcado para describir la estructura de documentos, para luego aplicar estilos de diseño al contenido para mostrarlo. SGML proporciona la sintaxis necesaria para crear lenguajes de marcado que pueden utilizarse de esta manera y hay muchos lenguajes que se usan de esta forma. HTML no es uno de ellos aunque está basado en SGML. En su origen, HTML era un lenguaje de marcado puramente basado en contenido (estaba diseñado para permitir que los investigadores compartieran notas técnicas). Los primeros programas cliente de Web soportaban la visualización de documentos HTML, pero el programa cliente era quien determinaba el diseño de los documentos y no el marcado de HTML. Por ejemplo, en un determinado programa cliente, todos los párrafos marcados con la etiqueta **<p>** podrían aparecer en una fuente *Times New Roman* de 12 puntos. Un programa cliente distinto podría haber utilizado una fuente *Courier* de 14 puntos. Lo importante es que los programas cliente tomaban las decisiones sobre el diseño, y no los documentos, lo que estaba en contraparte con la idea de un lenguaje de marcado universal.

[1]Los metadatos consisten en información que caracteriza datos. Los metadatos son utilizados para suministrar información sobre los datos producidos.

3.1.1. Extraer la presentación de HTML

Una hoja de estilos define las reglas de diseño que le indican a un programa cliente como mostrar partes de un documento HTML. De forma más específica, las reglas de las hojas de estilos se utilizan para traducir la estructura lógica de un documento a una forma que se adapte bien a la presentación. XML es un lenguaje de marcado basado en contenido, por lo que depende completamente de las hojas de estilos para su visualización. Así, mientras que las hojas de estilos mejoran la estructura y la organización de los documentos HTML, son absolutamente esenciales para mostrar documentos XML.

3.2. Mostrar XML con las hojas de estilo

Las CSS proporcionan una forma de aplicar reglas de diseño para XML (así como lo hicieron para HTML), para que pueda visualizarse. La solución que aportan las CSS para dotar de estilo a XML no es lo suficientemente potente como para tratar con el problema de los distintos medios de visualización, puesto que no puede transformar un documento XML en un formato diferente, lo que suele ser necesario para mostrar un documento en un medio distinto. Pero esto no es un problema de las CSS, ya que no estaban diseñadas para transformar documentos. Las CSS no implican ninguna transformación, sino que proporcionan una forma de describir como deben aparecer las distintas partes de un documento.

La tecnología de hojas de estilos que puede transformar los documentos XML es el lenguaje extensible de hojas de estilo de transformación (XSLT), que es un subconjunto de la tecnología XSL. XSLT soporta la transformación de un documento XML de un formato a otro. Así, se podría tomar un documento XML y transformarlo en un documento HTML o en otro tipo de documento XML personalizado. XSLT permite transformar los documentos XML en función de una serie de patrones estructurados. Para fines de visualización, se puede usar XSLT para transformar un documento XML en un documento HTML. Además, dado que XSLT es una tecnología de transformación, se puede seguir usando las CSS con el fin de ayudar a definir el estilo.

Otra tecnología que permite la visualización de los documentos XML son los objetos de formateo del lenguaje de hojas de estilo extensible (XSL-FO), los cuales son también un subconjunto de la tecnología XSL. Estos permiten dar formato a los documentos XML, pero sin llevar cabo ninguna transformación, son un lenguaje de marcas cuya finalidad es definir colores, tamaños y posición de los elementos del documento XML para poder visualizarlos.

Se cree que XSL y CSS son tecnologías competidoras, pero esto no es así, XSL y CSS son ambas tecnologías de hojas de estilos, pero la realidad es que abordan problemas distintos. La tecnología de XSL-FO hará todo lo que las CSS puedan hacer y mucho más. Sin embargo, está diseñada como superconjunto funcional de las CSS, lo que significa que sustituirá a las CSS en vez de competir con ellas.

Hay dos diferencias clave entre XSL y CSS:

- Las CSS se pueden usar también para dar estilo a los documentos HTML, mientras que XSL no lo permite.
- XSL se puede usar para transformar documentos XML, mientras que las CSS no lo permite.

Este trabajo de tesis trata sobre XML, no sobre HTML, por lo que la primera diferencia podría no importar mucho. Pero es importante, porque en el editor XTE se hará uso de las CSS para darles estilo a los documentos HTML recién transformados a partir de los documentos XML originales.

La segunda diferencia describe que las CSS no proporcionan una forma directa de transformar documentos XML. Sin embargo, aunque no sea posible utilizar las CSS para transformar los documentos XML, sí pueden utilizarse para darles estilo. Sin embargo, como se acaba de mencionar, esto no será utilizado por el editor XTE.

XSL es una tecnología más potente que las CSS, pero la potencia añadida se incorpora a costa de una complejidad también añadida. Si no importa tener una curva de aprendizaje más larga, las posibilidades de XSL de buscar y reorganizar el contenido del documento serán mayores.

XSL fue concebido como respuesta a las necesidades de estilos de XML. En general, puede imaginarse la relación que hay entre XSL y XML de forma muy parecida a la relación que hay entre las CSS y HTML. Esta comparación no es muy exacta, ya que XSL define un superconjunto de funcionalidad de estilos que está disponible actualmente en las CSS, pero las posibilidades de transformación de XSL están particularmente adaptadas a XML. XSL se basa en el lenguaje de especificación de la semántica de estilos de documentos (DSSSL) y las CSS, por lo que representa la combinación de muchos conocimientos sobre hojas de estilos.

3.2.1. Las CSS

CSS es un lenguaje de hojas de estilos que está diseñado para proporcionar una forma de dotar de estilo a los documentos HTML, permitiendo que los desarrolladores Web separen el contenido de la presentación. Antes de las CSS, las únicas opciones que había para dar estilo a documentos HTML eran los lenguajes de automatización y soluciones híbridas, como el HTML Dinámico (DHTML). Pero las CSS son más fáciles de aprender y usar, lo que las convierte en una tecnología ideal para dar estilo a los documentos HTML. Aunque las CSS fueron diseñadas para ser usadas con HTML, también son muy útiles para dar estilo a los documentos XML. Sin embargo, en el editor XTE las CSS sólo serán usadas para darle estilo a los documentos HTML que fueron generados a partir de una transformación con XSLT. No serán usadas para darle estilo a los documentos XML, para este proposito se usará la tecnología XSL.

3.2.2. La tecnología XSL

Como se mencionó anteriomente, la tecnología XSL es una tecnología de hojas de estilos que contempla dos aspectos de los documentos XML:

- Lenguaje extensible de hojas de estilo de transformación (XSLT). Permite la transformación de documentos XML de un formato a otro.
- Objetos de formateo del lenguaje extensible de hojas de estilo (XSL-FO). Permite dar estilo a los documentos XML en función de reglas de formato. Muchas veces cuando se habla de XSL se está hablando de XSL-FO ya que se pretende sea el estandard de facto.

Los XSLT y XSL-FO se implementan como un vocabulario XML compuesto por elementos y atributos importantes para desempeñar cada parte del proceso de presentación del documento. El uso de ambos vocabularios ofrece a los desarrolladores Web, control sobre la transformación del contenido de los documentos.

3.2.3. Utilización conjunta de XSL y las CSS

A continuación se exponen tres vías en las que XSL y las CSS se pueden usar conjuntamente para proporcionar una solución de hojas de estilos para documentos XML:

1. Utilizar XSL en el servidor para transformar documentos XML en documentos HTML que se dotan de estilo con las hojas de estilos CSS.
2. Utilizar XSL en el servidor para transformar documentos XML en documentos XML que se dotan de estilo con las hojas de estilos CSS.
3. Utilizar XSL en el cliente para transformar documentos XML en documentos HTML que se dotan de estilo con las hojas de estilos CSS.

Estas soluciones al uso de XSL y CSS se enumeran en orden de menor a mayor dificultad. La primera solución presenta la mejor combinación de XSL y CSS, ya que no requiere nada especial de parte de los navegadores Web; todos los navegadores ven documentos HTML normales que se generaron en el servidor.

La segunda solución tiene ventajas, porque implica dar estilo al código XML directamente en los navegadores. Esto significa que no es necesario transformar los documentos XML en documentos HTML. La única transformación es la transformación de un documento XML en otro documento XML que está mejor adaptado a la presentación. Esta solución depende del soporte del navegador para ver directamente documentos XML con CSS.

La tercera solución es la más interesante para el presente trabajo, porque los documentos XML se transforman en documentos HTML y se les asignan estilos directamente en el navegador. Esta

será la vía utilizada por el editor XTE, ya que será el editor quien dote de estilo a los documentos HTML por medio de hojas de estilo CSS. Las hojas de estilo en cascada no se usarán para dar formato a los documentos XML del editor, para esto se utilizará XSL. Sólo se utilizarán las CSS para dar formato a los documentos HTML transformados, por lo tanto, no se profundizará mas en ellas. El editor XTE da la opción de presentar los documentos con y sin hoja de estilos, lo que hace evidente su uso.

3.3. El lenguaje de estilo extensible (XSL)

XSL es una tecnología de hojas de estilos que está específicamente diseñada para XML. Sin embargo, tiene una el problema de que a diferencia de las CSS que posee una tecnología y sintaxis muy fáciles de aprender, XSL incluye algunas construcciones de programación de gran potencia pero que pueden ser complejas.

3.3.1. Procesar una hoja de estilos XSL

Para comprender la totalidad de XSL, se debe entender muy bien como un procesador XSL puede procesar documentos XML. Dos cosas son necesarias antes de que un procesador XSL pueda procesar un documento: la representación del árbol XML del documento y la hoja de estilos XSL. La representación del árbol XML de un documento se obtiene analizando sintácticamente el documento, lo que significa que el procesador XSL necesita un analizador XML para que funcione. Esto también significa que no es tarea del procesador XML analizar sintácticamente documentos XSL. Para que el analizador XSL comienze su trabajo, el analizador XML debió primero validar el documento.

El procesador XSL comienza con el nodo raíz del árbol, utilizándolo para llevar a cabo el cotejo de los patrones en la hoja de estilos. Una hoja de estilos XSL está formada por plantillas que utilizan patrones para determinar qué partes de un documento XML tienen que ser formateadas. El procesador XSL analiza estas plantillas y los patrones asociados para procesar las distintas partes del árbol. Cuando hay una coincidencia, la parte del árbol que coincide con el patrón determinado se pasa a la plantilla de la hoja de estilos para su procesamiento. El procesador XSL sigue las reglas de la plantilla para generar un árbol de resultados. Este punto es importante, ya que toma un árbol como entrada y genera otro árbol como salida.

El procesador XSL sigue procesando cada nodo del árbol XML, aplicando todas las plantillas que se definen en las hoja de estilos. Cuando se han procesado todos los nodos y se han aplicado todas las plantillas de hojas de estilos, el procesador XSL devolverá el árbol de resultados completo, que usualmente está en un formato adaptado a su visualización (es posible usar XSL para convertir un documento entre un vocabulario XML y otro, lo que no tiene por qué afectar necesariamente a la visualización del documento).

3.3.2. La arquitectura de XSL

XSL ha evolucionado en diversas tecnologías, algunas de las cuales son útiles fuera de XSL. Estas tecnologías forman parte importante de XSL y de XML. Para poder comprender la importancia de estas tecnologías, es importante examinar el papel del procesador XSL.

El procesador XSL lleva a cabo dos acciones fundamentales:

- La construcción de un árbol de resultados a partir de un árbol de origen.
- La interpretación del árbol de resultados para fines de formateo.

El primer papel del procesador XSL implica transformar un árbol de origen en un árbol de resultados, lo que se conoce como transformación de árboles. Básicamente, este proceso consiste en transformar contenido XML de un vocabulario a otro. La segunda acción del procesador XSL implica examinar el árbol de resultados para ofrecer información de formato y luego dar formato al contenido de cada nodo.

Las dos tareas fundamentales que lleva a cabo directamente el procesador XML se corresponden con dos tecnologías XSL que ya se habían mencionado antes:

- **Lenguaje extensible de hojas de estilo de transformación (XSLT).** El XSLT es el lenguaje de transformación XSL que se usa para transformar documentos XML de un vocabulario a otro.
- **Objetos de formateo del lenguaje extensible de hojas de estilo (XSL-FO).** Los objetos de formateo de XSL es el lenguaje de formato XSL que se usa para aplicar estilos de formato a los documentos XML con el fin de visualizarlos.

Estas dos tecnologías se implementan como vocabularios XML, que hacen que la sintaxis sea muy conocida. Esto también significa que las hojas de estilos creadas a partir de ellos sean documentos XML. Lo más interesante de estos dos componentes de XSL es que se pueden usar de forma separada. Puede usar XSLT para transformar documentos sin preocuparse por qué formato tienen los documentos. Análogamente, es posible utilizar los objetos de formateo XSL para dar formato a los documentos XML sin tener que llevar a cabo necesariamente ninguna transformación.

En el editor XTE se utilizarán ambas tecnologías. La tecnología XSLT para transformar a HTML (con y sin CSS) y la tecnología XSL-FO para transformar al formato PDF.

3.3.3. XSLT

XSLT es el componente de transformación de la tecnología de hojas de estilos XSL. XSLT consta de un vocabulario XML que se usa para crear hojas de estilos para transformar documentos XML. Funciona con datos XML analizados sintácticamente en forma de árbol y muestra un árbol de

resultados formado por los datos transformados. Utiliza un patrón muy potente para seleccionar partes de un documento XML para su transformación. Cuando se coteja un patrón con una parte del árbol, se usa una plantilla para determinar como se transforma esa parte del árbol.

La habilidad de transformar datos de Internet en múltiples formatos, como XML, HTML o WML[2] se está convirtiendo cada vez en lo más importante. El XSLT se puede utilizar para transformar XML en cualquier formato deseado. Un motor de transformación o un procesador tomaría un documento XML como entrada y utilizaría la hoja de estilo de transformación XSL para crear un nuevo formato de documento, según lo mostrado en la figura 3.1.

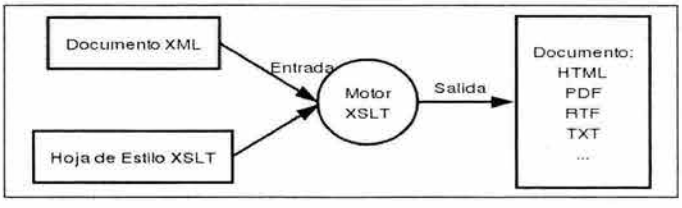

Figura 3.1: Usar un motor XSLT

3.3.4. XSL-FO

FO[3] quiere decir objetos de formateo. Un archivo XSL-FO es un conjunto de *tags* que permiten definir colores, tamaños y posición. Es un archivo que no preserva nada de la semántica de la información original, solamente describe como debe mostrarse en pantalla, o en papel. Este lenguaje no se limita a definir qué estilo aplicar a cada elemento del documento XML, además puede realizar pequeñas instrucciones típicas de los lenguajes de programación y la salida no tiene porque ser un documento HTML, sino pueden ser otros tipos como PDF, un documento de texto plano (TXT) u otro documento XML.

Este tipo de archivo ayuda a crear una hoja de estilo que despues podrá ser interpretada por algún convertidor de estilo, que interpreta los objetos de formateo, para transformar el archivo XML en otro formato conocido (PDF, PS, XML, TXT, etc.). Más adelante se verá como construir plantillas con XSL-FO.

Procesadores XSL-FO

Un procesador XSL es la aplicación que procesa un documento XML compuesto de archivos de plantilla XSL-FO y lo presenta de manera que una persona lo pueda leer fácilmente. La figura 3.2 muestra el modelo de procesamiento de XSL-FO.

[2]Del inglés "Wireless Markup Language".
[3]Del inglés "*formatting objects*".

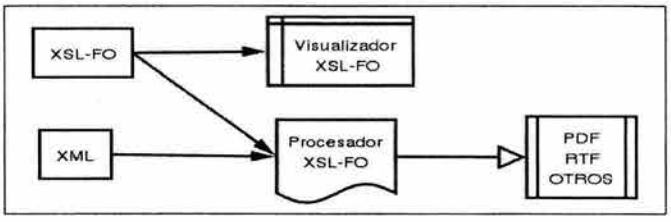

Figura 3.2: Modelo de procesamiento de XSL-FO

Como se observa en la figura anterior el procesamiento XSL-FO necesita de un documento XML una plantilla XSL-FO y un procesador XSL-FO para convertir el documento en algún formato visualizable. Opcionalmente, existen visualizadores de plantillas XSL-FO que permiten ver como están estructuradas las plantillas y modificarlas si fuera necesario.

Existen varios procesadores XSL-FO. Estos son algunos de los más significativos:

- **Apache's FOP** [?], que es un procesador de XSL-FO desarrollado en Java por Apache XML Project. Es gratuito y aunque no es tan potente como el XEP, es sin duda alguna el más utilizado en la actualidad (es gratuito). Entre sus ventajas cabe destacar que permite trabajar con documentos SVG. FOP es parte del proyecto Apache XML.
- **XEP** [Xep], desarrollado por RenderX. Es sin duda alguna el más avanzado. El único inconveniente es que se trata de una aplicación comercial aunque no se puede bajar una versión de demostración evidentemente limitada en sus funcionalidades. Esta desarrollada en Java.
- **Unicorn Formatting Objects (UFO),** que es un procesador de XSL-FO implementado en C++. La salida de esta herramienta es TEX y a partir de este formato se puede generar PostScript, PDF, etc.
- **REXP** [Rexp], herramienta que permite convertir a formatos PDF como las dos anteriores, pero además, convierte a PS (Postcript) y HTML. Es un proyecto basado en FOP.
- **JFOR** [JFor], convierte documentos XML conforme a la especificación XSL-FO al formato RTF (Rich Text Format). Lo interesante aquí, es que esta herramienta puede usar la misma plantilla XSL-FO que se usa con FOP para convertir a PDF, para generar un RTF.

Para generar un archivo XSL-FO, se puede con un simple editor de texto, y realizarlo a mano. Sin embargo, existen ya, algunas herramientas para hacerlo por medio de programación. Por ejemplo:

- **JFO** [Jfo], es un API java que ayuda a generar archivos XSL-FO de forma programativa. No es gratuita, pero se permite bajar una versión de evaluación.
- **XSLfast** [Xslfast], es una herramienta que permite editar archivos XSL-FO de forma gráfica. Es un editor gráfico que permite crear y definir documentos XSL-FO, de una forma fácil y rápida.

3.3.5. Crear hojas de estilos XSL

Aunque las hojas de estilos XSL son muy potentes y pueden llegar a ser algo complejas al aplicarse a documentos XML complejos, su estructura básica es muy clara. Una hoja de estilos XSL consta de una o más plantillas que describen los patrones, que se usan para cotejar el contenido XML en relación a los estilos. Las dos construcciones fundamentales de una hoja de estilos XSL son:

- Plantillas
- Patrones

El elemento *stylesheet* es el elemento de documento de las hojas de estilos XSL. Este elemento forma parte de los espacios de nombres XSL, junto con los demás elementos y atributos que conforman XSL. Es necesario que se declare el espacio de nombres XSL para poder usar los elementos y atributos XSL. Un ejemplo de declaración de espacio de nombres XSL dentro del elemento *stylesheet* podría ser:

```
<xsl:stylesheet xmlns:xsl=http://www.w3.org/TR/WD-xsl>
```

En este ejemplo, el prefijo del espacio de nombres XSL se establece a XSL, que es la solución común que se usa en las hojas de estilos XSL. Hay que anteponer este prefijo a todos los elementos y atributos XSL.

Plantillas

Una plantilla es una estructura XSL que describe la salida a generar en base a ciertos criterios de coincidencia de criterios. La idea consiste en definir una estructura de transformación que se aplique a una cierta parte de un documento XML. Es posible crear hojas de estilos compuestas por una sola plantilla, en cuyo caso ésta manejará los detalles de la navegación por un árbol XML para dar formato a un contenido específico. Las plantillas se definen en las hojas de estilos XSL utilizando el elemento *xsl:template,* que es, principalmente, un elemento contenedor de los patrones y datos de transformación. El elemento *xsl:template* utiliza un atributo opcional llamado *match* para hacer coincidir los patrones en un documento XML. El atributo *match* especifica una porción del árbol XML de un documento. La coincidencia más amplia posible de un documento consiste en establecer el atributo *match* a "/", lo que indica que la raíz del árbol tiene que coincidir. El resultado es que en la plantilla selecciona la totalidad del árbol, como en la figura 3.3:

```
<xsl:template match=/>
...
</xsl:template>
```

Figura 3.3: El atributo *match*

El atributo *match* de una plantilla es algo así como una consulta de un lenguaje de base de datos (los datos del documento XML que coinciden con el patrón del atributo *match* se pasan a la plantilla para su procesamiento). Por ejemplo, en la figura 3.4 lo siguiente sólo coincide con los elementos llamados *state* :

```
<xsl:template match=state>
<xsl:value-of/>
</xsl:template>
```

Figura 3.4: El atributo *match* buscando valores que coinciden con *state*

Patrones

Los patrones se usan en las plantillas XSL para llevar a cabo cotejos que son los responsables de determinar qué porciones de un documento XML se pasan a través de una determinada plantilla para su transformación. Un patrón describe una rama de un árbol XML, que a su vez consiste en una serie de nodos jerárquicos. La sintaxis que utilizan los patrones XSL es parecida en parte a la que se usa para especificar las rutas de una unidad de disco.

Para seleccionar un árbol de documento completo, se usa el patrón raíz, que consta de una sola barra (/). El patrón raíz se presupone que está en otros patrones si no se sustituye. Los patrones se utilizan en XSL para describir porciones de un árbol de documento con el fin de aplicar plantillas.

3.3.6. Construcción de plantillas XSLT

El vocabulario XSLT define varias construcciones de plantillas que controlan la aplicación de las plantillas en las hojas de estilos XSL. Estas construcciones son elementos que se definen en el espacio de nombres XSL. A continuación algunos de los elementos XSLT más utilizados:

- **xsl:value-of**
- **xsl:if**
- **xsl:for-each**
- **xsl:apply-templates**

El elemento XSL:VALUE-OF

El elemento *xsl:value-of* se usa para insertar el valor de un elemento o atributo en la salida resultante de la hoja de estilos. Proporciona el mecanismo necesario para transformar los documentos XML, ya que permite ofrecer datos XML en cualquier contexto, como por ejemplo, dentro del marcado HTML. En la figura 3.5 un ejemplo que utiliza el elemento *xsl:value-of* para establacer el valor de un elemento llamado *name* como encabezado *h2* de HTML:

```
<xsl:template match="name">
    <h2><xsl:value-of/></h2>
</xsl:template>
```

Figura 3.5: Se imprime el valor de el elemento *name*

El elemento XSL:IF

El elemento *xsl:if* se usa para llevar a cabo cotejos condicionales en las plantillas. Este elemento utiliza el mismo atributo *match* que el elemento *xsl:template* para establecer la ramificación condicional de las plantillas. En la figura 3.6 se ve un ejemplo de como usar el elemento *xsl:if* para probar si el atributo *rating* de una película equivale a 5:

```
<xsl:if match="@rating=5">
    <xsl:apply-templates select="movie"/>
</xsl:if>
```

Figura 3.6: El elemento condicional *xsl:if*

El elemento XSL:FOR-EACH

El elemento *xsl:for-each* se utiliza para establecer un bucle para recorrer los elementos de un documento. El atributo *select* determina qué elementos se seleccionan como parte del recorrido del bucle. Otro atributo importante del elemento *xsl:for-each* es *order-by,* que especifica los criterios que se usan para ordenar los datos que procesa el elemento *xsl:for-each.* En la figura 3.7 se ve un ejemplo del uso de *xsl:for-each* para recorrer una lista ordenada de vehículos:

```
<xsl:for-each order-by="+price" select="vehicles/vehicle">
    <tr>
        <td><xsl:value-of select="@year"/></td>
        <td><xsl:value-of select="@make"/></td>
        <td><xsl:value-of select="@model"/></td>
        <td><xsl:value-of select="mileage"/></td>
        <td><xsl:value-of select="color"/></td>
        <td><xsl:value-of select="price"/></td>
    </tr>
</xsl:for-each>
```

Figura 3.7: El elemento condicional *xsl:for-each*

El elemento XSL:APPLY-TEMPLATES

El elemento *xsl:apply-templates* se usa para aplicar plantillas que se definen en una hoja de estilos. Soporta los atributos *select* y *order-by,* que llevan a cabo papeles similares a los del elemento *xsl:for-each.* Cuando el procesador XSL encuentra un elemento *xsl:apply-templates* en una hoja de estilos, la plantilla que corresponde al patrón del atributo *select* se aplica, lo que significa que los datos relevantes del documento se colocan en la plantilla y se transforman.

Un ejemplo de como aplicar una plantilla utilizando el elemento *xsl:apply-templates* es:

```
<xsl:apply-templates select=state/>
```

Desarrollar hojas de estilos XSL

La clave para comprender las hojas de estilos XSL es construir una. En la figura 3.8 se observa el ejemplo de libreta de direcciones AddressBook.xml, que se uso para las CSS:

```
<?xml version="1.0"?>
<?xml – stylesheet type="text/css" href="AddressBook.css"?>
<!DOCTYPE addressbook SYSTEM "AddressBook.dtd">
<addressbook>
    <contacto>
        <nombre>Rafael Montante</nombre>
        <calle>Tlacotlalpan 123</calle>
        <ciudad>Distrito Federal</ciudad>
        <pais>México</pais>
    </contacto>
    <contacto>
        <nombre>Antonio Cejudo</nombre>
        <calle>Av. Eugenia 345</calle>
        <ciudad>Distrito Federal</ciudad>
        <pais>México</pais>
    </contacto>
</addressbook>
```

Figura 3.8: La libreta de direcciones AddressBook.xml

En el ejemplo, una hoja de estilos XSL necesita recorrer los elementos *contact* para dar formato a esa información por medio de código HTML. El elemento *xsl:for-each* es perfecto para hacer un ciclo por los elementos *contact*. La hoja de estilos se llama Addressbook.xsl (ver figura 3.9).

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xs"/>
    <xsl:template match=/>
      <html><head><title>Address Book XML Example</title></head>
      <body bgcolor="#FFFFFF">
          <xsl:for-each select=addressbook/contacto>
            <xsl:apply-templates select="nombre"/>
            <xsl:apply-templates select="calle"/>
            <xsl:apply-templates select="ciudad"/>
          </xsl:for-each>
      </body>
      </html>
    </xsl:template>
<xsl:template match="nombre">
    <h2><xsl:value-of/></h2>
</xsl:template>
<xsl:template match="calle">
    <xsl:value-of/><br/>
</xsl:template>
<xsl:template match="ciudad">
    <xsl:value-of/>
</xsl:template>
</xsl:stylesheet>
```

Figura 3.9: La hoja de estilos XSL, Addressbook.xsl

Esta hoja de estilos muestra como usar el elemento *xsl:for-each* para establecer un ciclo que recorra los elementos *contact*. El atributo de selección del elemento *xsl:for-each* está establecido a "*addressbook/contact*", que sólo selecciona los elementos *contact* del documento para procesarlos por el ciclo. El elemento *xsl:value-of* se usa para insertar el contenido de los elementos de libreta de direcciones en el árbol de resultados de la hoja de estilos. Observese que el espacio de nombres XSL se declara en el elemento *xsl:stylesheet*.

Construcción de plantillas XSL-FO

Mediante los objetos de formateo y sus propiedades podemos describir cómo se van a visualizar los componentes de un documento. Con estos objetos se definen las características de la página, los párrafos, las listas, las tablas, los enlaces, etc. El siguiente código de la figura 3.10, es un pequeño ejemplo de archivo XSL-FO:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
        <fo:layout-master-set>
            <fo:simple-page-master master-name="simple"
                  page-height="29.7cm"
                  page-width="21cm"
                  margin-top="1cm"
                  margin-bottom="2cm"
                  margin-left="2.5cm"
                  margin-right="2.5cm">
              <fo:region-body margin-top="3cm"/>
              <fo:region-before extent="3cm"/>
              <fo:region-after extent="1.5cm"/>
            </fo:simple-page-master>
        </fo:layout-master-set>
        <fo:page-sequence master-reference="simple">
        <fo:flow flow-name="xsl-region-body">
        <fo:block font-size="18pt" font-family="sans-serif" line-height="24pt"
                  space-after.optimum="15pt" text-align="center" padding-top="3pt">
                    Mi primer XSL-FO
        </fo:block>
        <fo:block font-size="12pt"
                  font-family="sans-serif" line-height="15pt" space-after.optimum="3pt"
                  text-align="justify">
                    Hola este es mi primer XSL-FO.
        </fo:block>
        </fo:flow>
        </fo:page-sequence>
</fo:root>
```

Figura 3.10: Sencillo ejemplo de XSL-FO, *hola.fo*

Del ejemplo anterior se puede observar:

- Que se trata de un vocabulario XML, en el que todos los elementos van precedidos del *NameSpace* 'fo', y que, por tanto, al escribir el elemento raíz del documento XML se debe declarar de la siguiente manera:

```
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
```

- Que está formado por un conjunto de elementos: *fo:simple-page-master, fo:flow, fo:block,* etc. y sus propiedades (definidas en los atributos): *font-size, font-family,* etc. describen como se visualizan de forma genérica los componentes de un documento. La especificación define por tanto todos estos elementos y sus propiedades y como deben expresarse mediante un vocabulario XML.

Si al documento del ejemplo anterior, se usa un procesador FOP:

```
bash$>fop.sh -fo hola.fo -pdf hola.pdf
```

El resultado que se obtiene es un documento en formato PDF[4] que se visualiza en la figura 3.11:

Figura 3.11: Documento convertido a PDF con FOP y una plantilla XSL-FO

XSL-FO y XSLT se complementan uno con el otro. Por ejemplo, si se tuviera un documento XML, se podría transformar con XSLT, primero a un documento XSL-FO, despues tomar este documento para generar un documento PDF. Es decir, la conversión del documento XML original a PDF hubiese constado de dos partes:

- Conversión XML original a XSL-FO mediante XSLT y un procesador XSLT.
- Conversión de XSL-FO a PDF mediante el procesador FOP.

Así entonces, XSL-FO y XSLT son dos herramientas muy poderosas que se complementan y que proporcionan muchas posibilidades.

3.4. Resumen

Se vio que el lenguaje HTML es un lenguaje de presentación visual muy básico y en sus origenes no soportaba ningún tipo de visualización. Eran los clientes de Web los que determinaban la forma

[4]Se uso para visualizar el documento el programa Acrobat Reader 4.0 para Linux Debian 3.0

de visualización, así que se tuvieron que incorporar marcas para visualización y hojas de estilo. El XML, por el contrario, desde el principio se diseñó para ser un lenguaje para control y manejo de los datos. La visualización de los documentos XML se incorporó aparte. En los documentos XML se pueden usar hojas de estilo en cascada (CSS) como en el HTML y se incorporó una nueva tecnología de hojas de estilo llamada XSL diseñada especialmente para XML.

La especificación XSL se subdivide en XSLT y XSL-FO. La primera sirve para transformar los documentos XML en documentos HTML y puede funcionar en conjunto con hojas de estilo en cascada para darles formato. La segunda permite transformar a diversos formatos (como PDF) con una misma plantilla de diseño. Se vieron algunos fundamentos de las plantillas XSL y algunos ejemplos de construcción de las mismas.

Para el editor XTE, se utilizarán ambas tecnologías. Se hará uso de un motor de transformación llamado Xalan[5] realizado en Java, que permitirá hacer la transformación de los documentos XML a documentos HTML. El motor Xalan sólo necesita el documento XML, ya validado y en forma de un objeto DOM y las plantillas XSLT necesarias para la transformación. Al final, al documento HTML generado se le incrustan, opcionalmente, las hojas de estilo en cascada. El documento HTML es usado dentro del editor XTE para presentarlo como una pre-visualización del documento XML que se está editando. En el siguiente capítulo se verá a detalle este proceso.

También, se hará uso de un motor de procesamiento XSL-FO llamado FOP[6], también en Java, que permitirá convertir los documentos XML en documentos con formato PDF. Para esto se necesitan unas plantillas XSL-FO y el documento XML en forma de un árbol DOM. El motor produce un documento PDF listo para exportarse. Igualmente, en el siguiente capítulo se verá con mas detalle.

Ya se tiene todo lo necesario para que el editor XTE funcione internamente. Puede construir un árbol DOM y realizar las transformaciones necesarias para la visualización de los documentos. Sólo falta la interfaz de usuario del editor que debe permitir realizar las operaciones de edición, visualización y colaboración por medio de objetos visuales como ventanas, menús, árboles, etc. Esta aplicación también fue desarrollada en el lenguaje Java y se describe en el siguiente capítulo.

[5]Procesador XSLT que también pertenece el proyecto Apache XML como Xerces.
[6]Procesador XSL-FO que pertenece el proyecto Apache XML.

Capítulo 4

DISEÑO E IMPLEMENTACIÓN DE UN EDITOR DE DOCUMENTOS XML EN JAVA

En el presente capítulo se presenta el desarrollo de un editor para la elaboración de documentos científicos y técnicos que permita la creación de documentos que estén escritos en el formato XML. Se diseñará, implementará y evaluará un editor de autoría colaborativa que use al lenguaje XML y que genere documentos a formatos exportables. Esta aplicación se realizará en el lenguaje de programación Java y se llamará Editor XML de Transformación (XTE).

Esta aplicación será un editor que permitirá la edición de documentos que tendrán el formato XML y puedan ser visualizados desde la aplicación misma. Permitirá exportar en otros formatos como HTML y PDF. La interfaz de usuario permite realizar las operaciones de edición, visualización y colaboración por medio de objetos visuales como ventanas, menús, árboles de navegación, etc. realizados en el lenguaje de programación Java.

Como se mencionó anteriormente, el editor XTE forma parte del sistema ELXI que está en constante desarrollo. La versión que se menciona y describe en esté capítulo es la versión 1.1.

4.1. Sistema de autoría colaborativa ELXI

El sistema ELXI ha sido desarrollado en el lenguaje de programación Java, lo cual presenta la ventaja de poder usar un esquema modular en su diseño, esta característica hace posible que un grupo de desarrolladores puedan trabajar en diferentes partes del sistema, lo cual hace posible incorporarse relativamente fácil al proyecto. Otra ventaja que ha ofrecido Java es la reutilización del código, lo cual ha permitido enfocarse más en el verdadero problema. Asimismo, concede la facilidad de ejecutar la aplicación final en diferentes plataformas. Otra característica de gran interés es la distribución de objetos, que mediante la invocación de métodos remotos es posible

acceder a los recursos de diferentes computadoras.

Como se mencionó al inicio de este trabajo, la arquitectura del sistema ELXI y algunos procesos como la administración y elaboración de documentos, creación de sesiones de trabajo y envíos de mensajes, han sido implementados. Sin embargo, el proyecto ELXI continua en la fase de desarrollo. Uno de los requerimientos que se demanda es una interfaz para el usuario, que permita la edición de documentos de forma colaborativa y exportar los documentos a otros formatos. Con el fin de proponer este mecanismo, primero se describirá el estado inicial del que se parte, es decir cómo está integrado y cómo trabaja ELXI y de tal forma conocer en que parte de la integración de ELXI es parte el editor colaborativo.

4.1.1. Breve panorama de la arquitectura del sistema ELXI

El sistema ELXI permite, a diferentes autores, editar sobre un mismo documento de tipo XML. Diferentes autores pueden estar haciendo sus contribuciones al mismo tiempo (síncronamente), desde lugares cercanos o en tiempos diferidos (asíncronamente), desde lugares diferentes, sin que ello represente problemas de inconsistencia del documento compartido. Los documentos XML producidos se podrán exportar a otros formatos, tales como HTML y PDF para su visualización.

La arquitectura de ELXI es de tipo cliente-servidor. El servidor ELXI está compuesto por una base de datos, un servidor Java RMI[1] y el sistema de archivos. La base de datos mantiene toda la información relacionada con los autores, contraseñas de acceso, grupos de trabajos, documentos, estado de los fragmentos y autores conectados. En el sistema de archivos están físicamente todos los documentos, en formato XML, que van a ser compartidos entre los autores. De esta manera, no se sobrecarga a la base de datos, almacenando los documentos de los autores, realizando así un rápido acceso a ellos. El servidor ELXI es el corazón de la administración de los documentos. Está implementado como un servidor RMI, ésta tecnología permite exportar varios métodos locales a clientes remotos, en el caso de ELXI exporta poco más de treinta métodos para la administración y la utilización del sistema de edición. Tiene a su cargo varias tareas, entre las cuales están: validar a los autores que ingresan al sistema; ejecutar el protocolo de inicio de sesión; realizar la conexión con la base de datos para consultar y actualizar cualquier información; administrar el sistema de archivos para recuperar los documentos XML y realizar actualizaciones sobre ellos; contactar a los autores para enviarles la información que sea de su interés, con respecto a cambios en los documentos o en los autores; etc.

El cliente ELXI consta de más de veinte clases escritas en lenguaje Java, las cuales tienen las siguientes funciones: generar la interfaz gráfica de autor, manteniendo en todo momento informado al autor de los cambios que puedan aparecer en cualquiera de los documentos de su interés y de manera consistente con las vistas que, sobre el mismo documento, tengan los demás autores (se incluye el editor de documentos local y un árbol de documentos); ofrecer los servicios de conexión, con el fin de que el cliente pueda contactar al servidor a través de llamadas a procedimientos

[1] Del inglés "Remote Method Interface".

remotos; manejar y administrar los documentos locales y la comunicación con el sistema de archivos local; exportar su interfaz al servidor RMI para que pueda ser contactado a través de llamadas de RMI; crear y destruir los hilos necesarios para la administración local de los documentos, etc. En la figura 4.1 se muestra la arquitectura de ELXI:

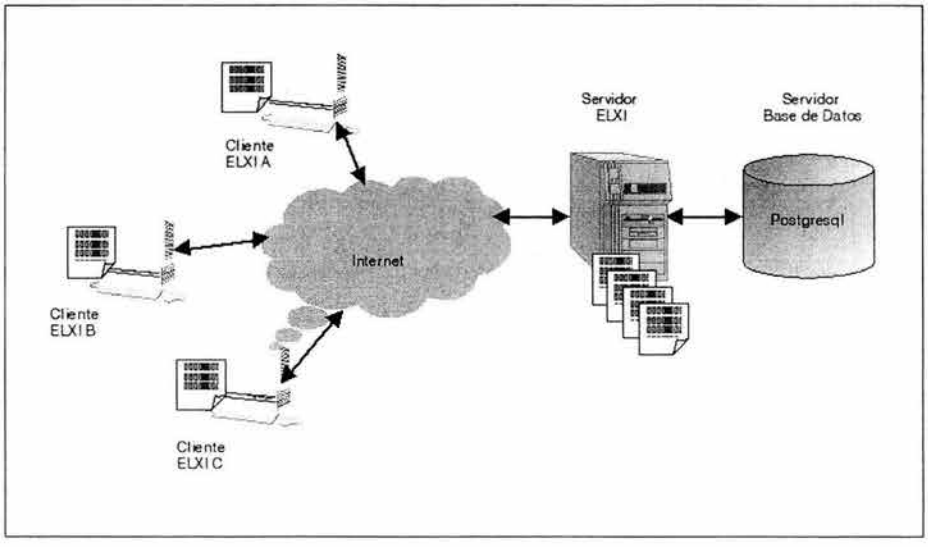

Figura 4.1: Componentes ELXI

En la figura se muestra cómo los clientes ELXI se conectan al servidor ELXI y éste mantiene una conexión permamente a la base de datos. Los clientes ELXI son precisamente el editor XTE. De esta manera, cada cliente ELXI tendrá un editor XTE que permitirá realizar la edición y procesos colaborativos.

A continuación se mencionará la especificación del lenguaje unificado de modelado (UML[2]), porque se usarán algunas herramientas de esa notación. Hay que hacer notar, que esta tesis está enfocada a describir las cualidades y alcances del editor colaborativo XTE y no en llevar a cabo todos los procesos involucrados en el desarrollo de software. Sin embargo, se describirá porqué se hará uso de algunas herramientas de la misma.

4.2. Diagramas de clases con UML

El lenguaje unificado de modelado (UML) es un lenguaje gráfico para visualizar, especificar y documentar cada una de las partes que comprende el desarrollo de software. Además, prescribe un conjunto de notaciones y diagramas estándar para modelar sistemas orientados a objetos y describe la semántica esencial de lo que estos diagramas y símbolos significan. Se puede usar para modelar distintos tipos de sistemas: sistemas de software, sistemas de hardware y organizaciones del mundo real.

[2]Del inglés "Unified Modeling Language".

Ofrece varios diagramas con los cuales es posible modelar sistemas:

- Diagramas de Casos de Uso para modelar los procesos que se requieren solucionar.
- Diagramas de Secuencia para modelar el paso de mensajes entre objetos.
- Diagramas de Colaboración para modelar interacciones entre objetos.
- Diagramas de Estado para modelar el comportamiento de los objetos en el sistema.
- Diagramas de Actividad para modelar el comportamiento de los Casos de Uso u Objetos.
- Diagramas de Clases para modelar la estructura estática de las clases en el sistema.
- Diagramas de Objetos para modelar la estructura estática de los objetos en el sistema.
- Diagramas de Componentes para modelar componentes.

UML es una consolidación de muchas de las notaciones y conceptos más usadas orientados a objetos. Prescribe una notación estándar y semánticas esenciales para el modelado de un sistema orientado a objetos. Previamente, un diseño orientado a objetos podría haber sido modelado con cualquier metodología de diseño de software causando a los revisores tener que aprender las semánticas y notaciones de la metodología empleada antes que intentar entender el diseño en sí. Con UML los diseñadores modelan sistemas diferentes y pueden entender los diseños de los otros.

4.2.1. Diagramas de Clases

Un diagrama de clases sirve para visualizar las relaciones entre las clases que involucran el sistema, las cuales pueden ser asociativas, de herencia, de uso y de contenimiento. El Diagrama de Clase es el el diagrama principal de diseño y análisis para un sistema. En él, la estructura de clases del sistema se especifica con relaciones entre clases y estructuras de herencia.

En este proyecto sólo se hará uso de los diagramas de clases para modelar las clases involucradas en el análisis sintáctico del documento XML y las clases involucradas en la transformación del documento para su visualización. También, el diagrama de clases de los componentes visuales que forman parte de la parte de la interfaz de usuario. Todos los diagramas de clases fueron realizados con el software de fuente abierta llamado Umbrello[3]. Si bien, se puede modelar todo el sistema con los distintos diagramas de UML, no es el objetivo de este trabajo observar todo el alcance de UML.

[3]Umbrello UML Modeller versión 1.2. Software de distribución libre y funciona para Linux. Referencia: http://uml.sf.net

4.3. Editor XTE

Ahora se describirá cómo se implementó el **Editor XML de Transformación** (XTE), el cual fue construido en el lenguaje de programación Java. Esta aplicación es un editor que permite la edición de documentos que contendrán el formato XML y que pueden ser visualizados desde la misma aplicación. Además, tendrá la capacidad de exportar a los formatos HTML y PDF. El editor XTE es la interfaz gráfica que se usará para la edición de los documentos XML dentro de ELXI.

4.3.1. El editor XTE como parte de ELXI

Como se mencionó en la introduccción del presente trabajo el editor XTE es parte fundamental del sistema de autoría colaborativa ELXI que está bajo la tutela del Dr. Manuel Romero Salcedo y su grupo de trabajo. El objetivo de este sistema es la elaboración de documentos de cualquier índole (artículos científicos, reportes técnicos, tesis, revistas, libros, etc.) a través de la colaboración de autores y lectores, situados en lugares que pueden estar geográficamente distantes. Una de las aportaciones de este sistema es el control de la información en documentos estructurados mediante el lenguaje de marcado extensible XML lo que permite que el manejo de cada documento sea más dinámico. El diseño de este sistema se ha basado principalmente en la característica de permitir que existan varios escritores editando un mismo documento a la vez.

Funcionamiento de ELXI a nivel del editor

En la figura 4.2 se puede observar en forma esquematica los componentes y el comportamiento funcional del editor de autoría colaborativa ELXI.

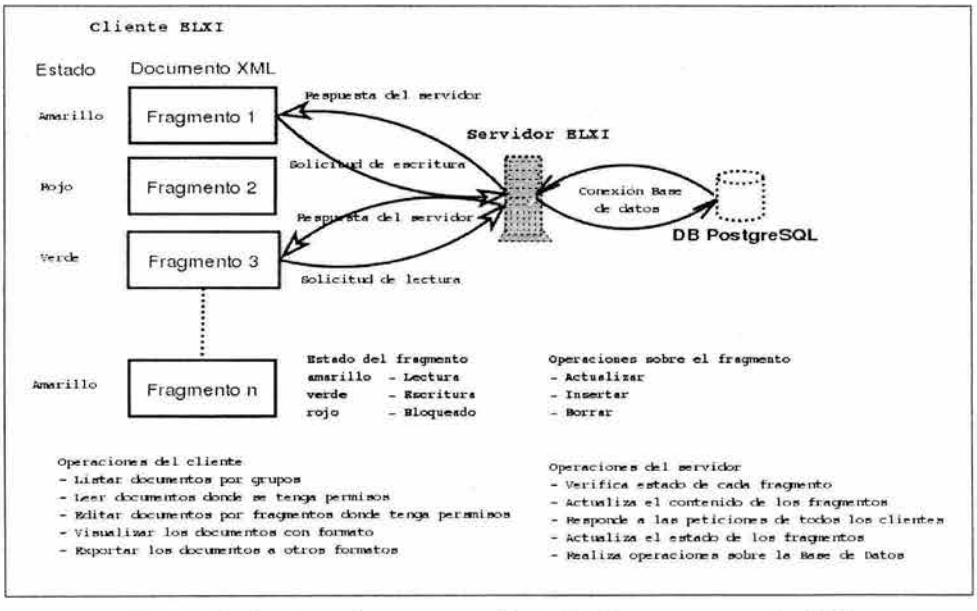

Figura 4.2: Funcionamiento esquemático del editor como parte de ELXI

Para poder tener la edición colaborativa que ELXI realiza, la aplicación necesita dividir los documentos de forma que uno o varios autores puedan estar editando una parte del documento mientras los demas editan otra sección. Así, el documento se divide en varios fragmentos donde cada uno tendrá permisos y atributos que permitirán controlarlo de forma autónoma. Por eso, el uso del formato XML como medio de control de los documentos es evidente, ya que este formato permite tener una descripción detallada y estructurada del documento. Los fragmentos son nodos XML, así cada nodo puede ser controlado como una entidad única que contiene información adicional que servirán para cuatro puntos importantes:

- **Definir permisos del documento.** Cada fragmento mantendrá un grupo de permisos que informarán a quien pertenece ese fragmento del documento y si tiene permisos de escritura, lectura o bloqueado. Con esta información la persona que este editando el documento podrá saber si puede editar ese fragmento del documento y además saber a quien pertenece. Además, en el caso de crear un nuevo fragmento se asignan los permisos que el autor del mismo establezca.

- **Definir el tipo de visualización que se quiere para el fragmento.** El tipo sirve para definir posteriomente cómo será visualizado ese fragmento. Hay un conjunto finito de tipos y todos están pre-definidos. Dependendiendo del tipo asignado será la forma en que será visualizado el fragmento. Este tipo servirá para definir atributos de color, fondo y posición.

- **Definir un estado que informa si es de lectura, escritura o bloqueado.** El estado define si el documento puede ser editado o está bloqueado porque otro autor está escribiendo sobre él. Si el autor observa que el estado es de poder editar el fragmento, hace una solicitud de bloqueo del mismo. Si es aceptada su petición el fragmento cambia su estado a bloqueado para los demas autores. El editor mantendrá de manera visible para el usuario un control de todos los posibles estados del fragmento. Los posibles estado que un fragmento puede tener son: lectura, escritura y bloqueado. El estado de lectura indica que sólo podrá leer el fragmento. El estado de escritura que el autor puede modificar y actualizar el fragmento. El estado de bloqueado indica que el fragmento está siendo editado por algún otro autor y no se podrá editar ni modificar el fragmento hasta que el estado cambie.

- **Un identificador único para diferenciarlo de todos los demas nodos o fragmentos.** El identificador único es para diferenciarlo entre todos los demás nodos o fragmentos. De está manera, ningún nodo será confundido con otro dentro del mismo documento u otros documentos.

Este es el funcionamiento interno básico del editor XTE en cuanto al proceso colaborativo. El editor XTE también contiene los componentes que están involucrados con el manejo de los documentos XML y los procesos de transformación para visualización, pero siempre se debe tener presente su funcionamiento dentro del ámbito colaborativo.

4.4. Componentes de la aplicación

Para realizar la aplicación se necesita ir diseñando cada componente que la constituye. Los componentes que constituyen la aplicación son básicamente:

- **Plantilla DTD.** Esta servirá para definir el tipo de documento que el editor XTE valida. Se debe recordar que para que un documento XML sea válido debe cumplir con una DTD.
- **Documento XML.** Documentos que nos sirven como *machotes*[4] para poder generar documentos nuevos que ya contengan una estructura mínima.
- **Plantilla XSLT** para realizar transformación a HTML. Esta plantilla servirá para poder transformar un documento XML en un documento HTML que puede ser visualizado dentro del mismo editor y también para exportarlo a un documento HTML que puede ser visualizado en cualquier medio que contenga el soporte de este formato.
- **Plantilla XSL-FO** para generar documentos con el formato PDF. Esta plantilla servirá para transformar el documento XML en un documento con formato PDF. Una vez transformado este documento podrá ser visualizado en cualquier motor que permita la visualización PDF como el Acrobat™ Reader[5].
- **Sistema XTE.** Este permitirá la edición de documentos de tipo artículo científico (que son los que maneja el sistema ELXI). Este editor hará uso internamente del formato XML para controlar y manipular los documentos. Permitirá hacer una pre-visualización de los documentos desde la misma aplicación y la posibilidad de exportar los documentos a otros formatos como HTML y PDF.

Se irán desarrollando y diseñando cada uno de estos componentes para que al final se tenga una aplicación que permita la edición de documentos de tipo artículo científico que pueda ser utilizada por investigadores y personas interesadas en la edición de éste tipo de documentos. Está aplicación es parte integral del editor colaborativo ELXI por lo que permitirá además todo el funcionamiento y control que los mecanismos de edición colaborativa necesitan para poder llevarse a cabo.

4.4.1. Herramientas adicionales

Se hará uso de varias herramientas para el desarrollo de está aplicación siendo de las más importantes las siguientes:

- Sistema Operativo Linux Debian GNU/Linux kernel 2.4.20 con escritorio XFCE4[6].

[4] Conocidas en inglés como "Templates".
[5] Acrobat Reader es un programa comercial de Adobe Systems Incorpored.
[6] XFCE es un entorno de escritorio para Linux, disponible en http://www.xfce.org

- Lenguaje de programación Java para construir la aplicación. Usaremos el kit de desarrollo de software (SDK[7]) Java™2, Standard Edition versión 1.3.1_02 del proyecto Blackdown Java-Linux[8]. Con éste se desarrollará toda la aplicación XTE.

- Librerias adicionales de Java para soporte y procesamiento de documentos XML, plantillas XSL y DTD. Algunas de éstas bibliotecas son:

  - **Xerces** - Procesador DOM versión 2.2.1. Es un conjunto de bibliotecas para Java que permiten hacer uso de un procesador DOM que permiten manipular, crear y editar un árbol de ánalisis DOM.
  - **Xalan** - Procesador XSLT versión 2.4.1. Bibliotecas que permiten toda la parte de transformación y procesamiento XSLT. Se usará para transformar al formato HTML para visualización y exportarlos a otros ambientes.
  - **Fop** - Procesamiento XSL-FO versión 0.20.5. Motor para procesamiento XSL-FO y permitir la transformación de documentos XML a otros formatos. En éste trabajo se usará para transformar al formato PDF.

- JBuilder™ 5.0 de Borland. Ambiente de programación Java sobre el sistema operativo Linux.

4.5. Procesar el documento

La primera etapa es generar las plantillas y documentos XML. Primero se debe determinar el tipo de documento que se usará en la aplicación. Para esto, se diseña una plantilla DTD que será la encargada de validar que el documento XML es válido y sintácticamente correcto. Después se crea un documento XML que cumpla con la DTD creada inicialmente. Una vez generado el documento XML, se tomá una plantilla de diseño XSLT para realizar una transformación al formato HTML. Después, se toma una plantilla XSL-FO para poder generar un documento PDF. Toda esta parte es el proceso básico interno de la aplicación.

La segunda etapa de la aplicación es la parte externa o de interfaz de usuario. Ésta es un programa realizado en Java que permite capturar texto y editar los documentos. Contiene todas las herramientas mínimas necesarias para poder editar los documentos, visualizarlos y realizar las transformaciones a otros formatos.

[7]Del inglés "Software Development Kit".

[8]http://www.blackdown.org/. Este proyecto desde los inicios de Java se ha encargado de portar al lenguaje de programación Java™ de Sun Microsystems al sistema operativo Linux. Así como su conjunto de herramientas y bibliotecas conocidas como Software Development Kit (SDK).

4.5.1. Definición de tipo de documento

El primer paso es desarrollar la plantilla de descripción de tipo de documento o DTD. El tipo de documento que describirá esta DTD es del tipo artículo científico o de investigación.

Plantilla DTD

Como se vio anteriormente, los documentos XML que se ajustan a su DTD, se denominan "válidos" o "correctos". El documento XML usado por la aplicación debe ser válido para mantener la integridad de los datos. Además, por ser un tipo de documento que se usará en un ambiente colaborativo y que será editado por muchos clientes al mismo tiempo se tiene la necesidad de mantener la validez, consistencia e integridad de los documentos. La DTD usada para la aplicación se puede observar en la figura 4.3:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!ELEMENT documento (fragmento+)>
<!ATTLIST documento
id_file CDATA #REQUIRED>
<!ELEMENT fragmento (#PCDATA)>
<!ATTLIST fragmento
titulo CDATA #REQUIRED
id_fragment CDATA #REQUIRED
tipo CDATA "">
```

Figura 4.3: DTD del editor XTE

Ésta DTD define que sólo habrá dos tipos de elementos en los documentos XML: el elemento raíz o root llamado "documento" que será único por archivo; y el elemento "fragmento", el cual puede tener de una a muchas ocurrencias por documento. Además, el elemento "documento", tiene un atributo llamado "id_file", el cual define un identificador único por cada documento. Los elementos "fragmento" tienen 3 atributos: "titulo" que define un título para el elemento "fragmento", "id_fragment" que define un identificador único por elemento y "tipo" que define un tipo específico para cada elemento, este último atributo servirá sobre todo para definir posteriormente con las hojas de estilo el modo de visualizar cada "fragmento".

El tipo de documentos XML que la DTD valida se puede observar en la figura 4.4:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE documento SYSTEM "documento.dtd">
<documento id_file="3">
<fragmento id_fragment="1" tipo="INFORMACION NORMA IMP" titulo="INFORMACION NORMA IMP"/>
<fragmento id_fragment="2" tipo="title" titulo="TITULO">prueba de norma</fragmento>
<fragmento id_fragment="3" tipo="abstract" titulo="INTRODUCCION">Segundo</fragmento>
<fragmento id_fragment="4" tipo="section" titulo="OBJETIVO">hola</fragmento>
<fragmento id_fragment="5" tipo="ALCANCE" titulo="ALCANCE"/>
</documento>
```

Figura 4.4: Ejemplo de documento XML que valida la DTD

De forma esquemática el documento XML se define como en la figura 4.5:

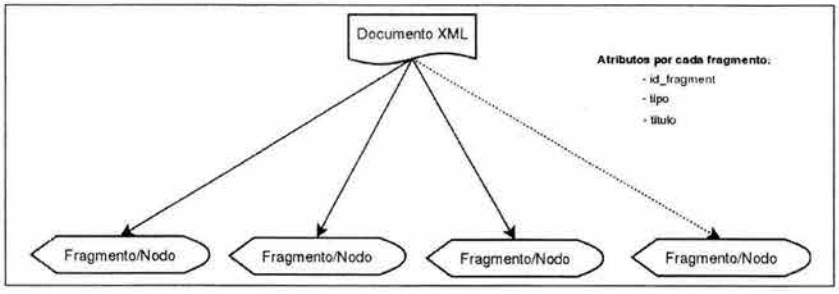

Figura 4.5: Estructura del documento XML

Una vez definido el documento XML y su DTD lo siguiente es procesar el documento para validarlo y poder manipularlo de forma rápida y sencilla. Para eso se hará uso del lenguaje Java y las API de DOM.

4.5.2. Procesar el documento XML

Analizar sintácticamente un documento XML es el primer paso a la hora de procesarlo. Si un documento no está bien formado, el analizador no deberá seguir procesando el documento de una manera normal y debe abortar marcando un error de construcción. Si hay una DTD y el analizador es de validación, éste también comprobará el contenido del documento con la DTD. El documento XML de la aplicación tiene una DTD, por lo que el analizador Java DOM tiene que verificar que esté bien construido y sea válido.

4.5.3. Analizar sintácticamente el documento XML

Como se construirá un editor que hará uso de documentos estructurados en formato XML se necesita tener en todo momento el conocimiento de cuales son los elementos para poder modificarlos en un momento dado. Se hará uso de la interfaz DOM de Xerces para procesar el documento.

El diagrama de clases del procesador sintáctico se visualiza en la figura 4.6:

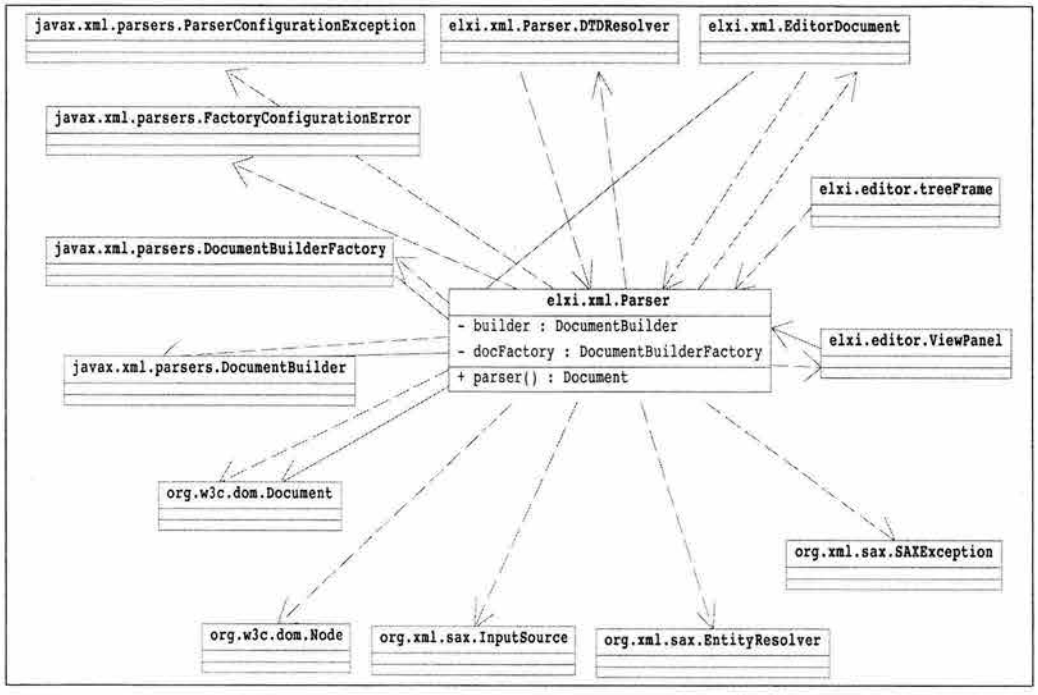

Figura 4.6: Diagrama de clases del analizador sintáctico

La clase principal del analizador sintáctico es*elxi.xml.Parser* devuelve una instancia de *Document* la cual permitirá interactuar con el árbol DOM que se generá a partir del archivo XML. Se utiliza el API de Xerces y los paquetes *elxi.xml* y *elxi.editor* del proyecto ELXI.

Obtener un documento DOM

Al crear los documentos, desde un inicio la aplicación necesita obtener un documento vacío que se irá generando poco a poco conforme se vaya creando cada fragmento.

La clase Java que permite procesar el documento se llama *elxi.xml.Parser* (ver figura 4.7).

```
package elxi.xml;
import javax.xml.parsers.*;
import org.xml.sax.*;
import org.w3c.dom.*;
    public final class Parser{
      public static DocumentBuilder builder;
      private static DocumentBuilderFactory docFactory;
      public transient Document doc;
      private boolean validating = false;
        static {
          try {
            docFactory = DocumentBuilderFactory.newInstance();
            docFactory.setCoalescing(true);
            docFactory.setExpandEntityReferences(true);
            builder = docFactory.newDocumentBuilder();
          }
          catch (Exception e) {
            e.printStackTrace();
          }
        }
        public Parser(boolean validating){
            docFactory.setValidating(validating);
            builder.setEntityResolver(new DTDResolver());
        }
        public synchronized Document parse(InputSource source) throws Exception{
            source.setEncoding("ISO-8859-1");
            doc = builder.parse(source);
            return doc;
        }
      }
```

Figura 4.7: Código Java del procesador DOM con Xerces

Una vez que se tiene el documento XML, procesado y en memoria, se puede por medio del DOM, modificar cualquiera de sus nodos o elementos e interacturar con ellos. Se puede crear tantas instancias como se necesite. Sin embargo, se necesita en este código alguna forma de decirle al procesador XML donde está colocada la DTD para poder validar al documento (ver figura 4.8).

```
        public static class DTDResolver implements EntityResolver {
            String separador = System.getProperty("file.separator");
            String rutaDTD;
            public DTDResolver(){
              rutaDTD = System.getProperty("user.dir") + separador + "conf" + separador + "do cumento.dtd";
            }
            public DTDResolver(String rutaDTD){
              this.rutaDTD = rutaDTD;
            }
            public InputSource resolveEntity (String publicId,String systemId){
              System.out.println("Cargando la DTD "+rutaDTD);
              return new InputSource(rutaDTD);
            }
        }
```

Figura 4.8: Clase que carga la DTD

Todos los documentos, nuevos o existentes, se validan con la clase *elxi.xml.Parser*, si ésta no marca ningún error de sintaxis y cumple con la DTD entonces el documento es válido y puede ser usado por la aplicación. En caso contrario, marca un error y se detiene la validación.

En la figura 4.9 se observa el código Java que permite crear nuevos objetos DOM a partir de archivos XML:

```
package elxi.xml;
import java.io.*;
import java.util.*;
import org.w3c.dom.*;
import org.xml.sax.SAXException;
import elxi.xml.Parser;
import elxi.editor.*;
    public class EditorDocument {
        public Parser dom;
        protected Document doc;
        protected File file;
        public EditorDocument(Document doc) {
            this.doc = doc;
        }
        public Document createNewDocument(File def) {
            dom = new Parser(true);
            try {
                doc = dom.parse(def.getAbsolutePath());
            } catch (IOException e) {
                System.out.println("Error de I/O:"+"No puede abrir archivo");
                return null;
            } catch (org.xml.sax.SAXParseException se){
                System.out.println("Error de síntaxis:"+se.getMessage()+" Linea:"+se.getLineNumber()+" Columna:
"+se.getColumnNumber());
                return null;
            } catch (org.xml.sax.SAXException se){
                System.out.println("Error de sintaxis:"+se.getMessage());
                return null;
                } catch (java.lang.Exception e){
                    System.out.println("Error desconocido:"+e.getMessage());
                    return null;
            }
            return doc;
    }
```

Figura 4.9: Clase que crea un objeto DOM de un archivo XML

La clase *elxi.xml.Parser* es llamada desde la clase *elxi.editor.EditorDocument* la cual es usada dentro de la aplicación gráfica. La clase *elxi.editor.EditorDocument* devuelve una instancia de la interfaz *Document* la cual permite manipular todo el arbol DOM del archivo XML.

Ya se tiene un objeto DOM que no es más que la representación en memoria del árbol XML. Se tiene un objeto DOM por cada archivo XML que se necesite leer y manipular. Al objeto DOM se le pueden agregar nuevos elementos (o nodos del árbol), borrarlos o cambiarlos de lugar. Pero ahora el siguiente problema es cómo visualizar la estructura de árbol DOM de una forma coherente y que pueda ser leída por el autor del documento.

4.6. Dar formato a los documentos

Una vez procesado el documento con el analizador sintáctico, se tiene todo el documento en memoria dentro de un objeto o árbol DOM. A este objeto se le puede manipular y modificar. Ahora sólo falta poder visualizarlo. Como se mencionó anteriormente, XML por sí mismo no permite dar formato a los documentos, por este motivo se desarrolló la especificación XSL. El mecanismo general para aplicar formato de visualización a los documentos XML se conoce como hoja de estilos. Se úsara para tal proceso al lenguaje de estilo extensible (XSL) en sus dos tecnologías: XSLT para transformar a HTML y XSL-FO para generar los documentos en formato PDF. El diagrama de clases del motor de transformación XSLT se visualiza en la figura 4.10:

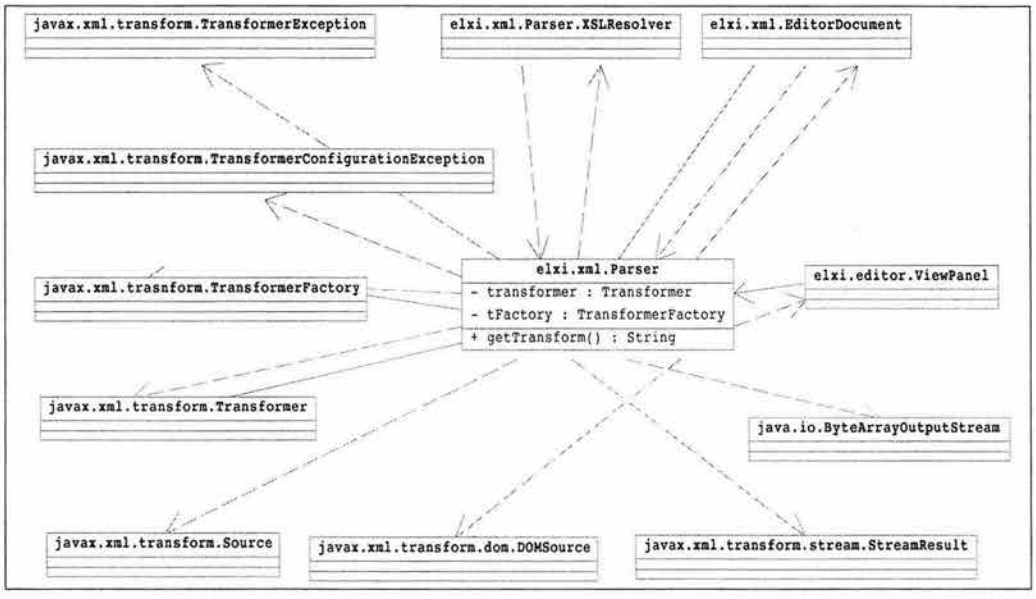

Figura 4.10: Diagrama de clases del motor de trasnformación XSLT

XSLT para transformar a HTML

El motor de transformación XSLT tomará el documento XML como entrada y utilizará la plantilla de transformación XSLT para crear el documento en formato HTML. El uso de la especificación XSLT es usado para transformar al formato HTML. Sin embargo, podría transformar a otros formatos.

Se uso el lenguaje HTML porque es el lenguaje de presentación mas fácil de implementar a partir de un documento XML y porque en el editor de transformación (XTE) tiene entre sus funcionalidades esta la de permitir hacer una pre-visualización del documento en formato HTML. Otro motivo para seleccionar al formato HTML es porque en el lenguaje Java sólo existe un componente (la clase *JTextPane)* capaz de visualizar un documento de ese formato. Se desarrollaron las plantillas XSLT que transforman a HTML y así poder visualizar el documento XML. El proceso para llevar a cabo la transformación con las plantillas XSLT es el siguiente:

- Se toma un documento XML y una plantilla XSLT ya definida. EL documento XML es el objeto DOM que ya se tenía procesado y en memoria.
- Se pasan por un motor de transformación XSLT. El motor de transformación se obtiene del paquete Xalan y se incrusta dentro de la clase *elxi.xml.Parser.*
- Se obtiene a la salida un documento en formato HTML. Lo que se obtiene es un documento en formato HTML que puede ser visto dentro de la aplicación.

Se desarrollaron 3 plantillas XSLT que tranforman a HTML. Cada una permite visualizar, de forma diferente, el documento XML. La lista de plantillas XSLT que se diseñaron y se usaron son:

- Plantilla *standard.xsl.* Contiene una muy básica presentación.
- Plantilla *normas.xsl.* Plantilla basada en estandares que usa Pemex.
- Plantilla *w3c.xsl.* Plantilla basada en los estandares que usa el organismo llamado W3C.

La clase Java que permite transformar el documento XML a HTML usando las plantillas XSLT es parte de la clase *elxi.xml.Parser* y partes importantes de ese código se observan en la figura 4.11:

```
package elxi.xml;
import javax.xml.parsers.*;
import javax.xml.parsers.dom.*;
import javax.xml.transform.stream.*;
import org.xml.sax.*;
import org.w3c.dom.*;
import java.io.*;
    public final class Parser{
      public static final String ENCODING="ISO-8859-1";
      public static Transformer transformer;
      private static javax.xml.transform.TransformerFactory tFactory;
        static {
          try {
            //Definimos la fabrica de transformaciones
            tFactory = javax.xml.transform.TransformerFactory.newInstance();
            //Método que permite realizar la transformación XSLT
            public synchronized String getTransform(Source ds,String name,String css) throws Exception {
            Transformer transformer = tFactory.newTransformer(ds);
            transformer.setParameter("nombre_archivo",name);
            transformer.setParameter("css",css);
            transformer.setOutputProperty(OutputKeys.ENCODING,ENCODING);
            transformer.setOutputProperty(OutputKeys.INDENT,"yes");
            transformer.setOutputProperty(OutputKeys.OMIT_XML_DECLARATION,"yes");
            DOMSource source = new DOMSource(doc);
            ByteArrayOutputStream bufStream = new ByteArrayOutputStream();
            StreamResult result = new StreamResult(bufStream);
            transformer.transform(source,result);
            String salida=bufStream.toString(ENCODING);
            return salida;
          return null;
        }
      }
```

Figura 4.11: Métodos de la clase *elxi.xml.Parser que sirven para XSLT*

Un documento XML que se sufrió una transformación XSLT con la plantilla *standard.xsl* genera un archivo HTML, el cual se visualiza en la figura 4.12 en el navegador Mozilla:

Figura 4.12: Transformación XSLT a HTML con la plantilla standard.xsl

Ahora, ya se tienen las plantillas XSL y se puede transformar por medio de un motor XSLT a HTML. El siguiente objetivo es la transformación a PDF. Para esto se hará uso de otra especificación del XSL llamada XSL-FO.

Plantillas usando los Objetos de formateo XSL-FO

Un archivo XSL-FO es un archivo que no preserva nada de la semántica de la información original, solamente describe cómo debe mostrarse en pantalla o en papel. Este tipo de archivo ayuda a crear una hoja de estilo que después podrá ser interpretada por algún convertidor de estilo, que interpreta los objetos de formateo, para transformar el archivo XML en otro formato conocido (PDF, PS, TXT, etc.). El diagrama de clases del motor XSL-FO se visualiza en la figura 4.13:

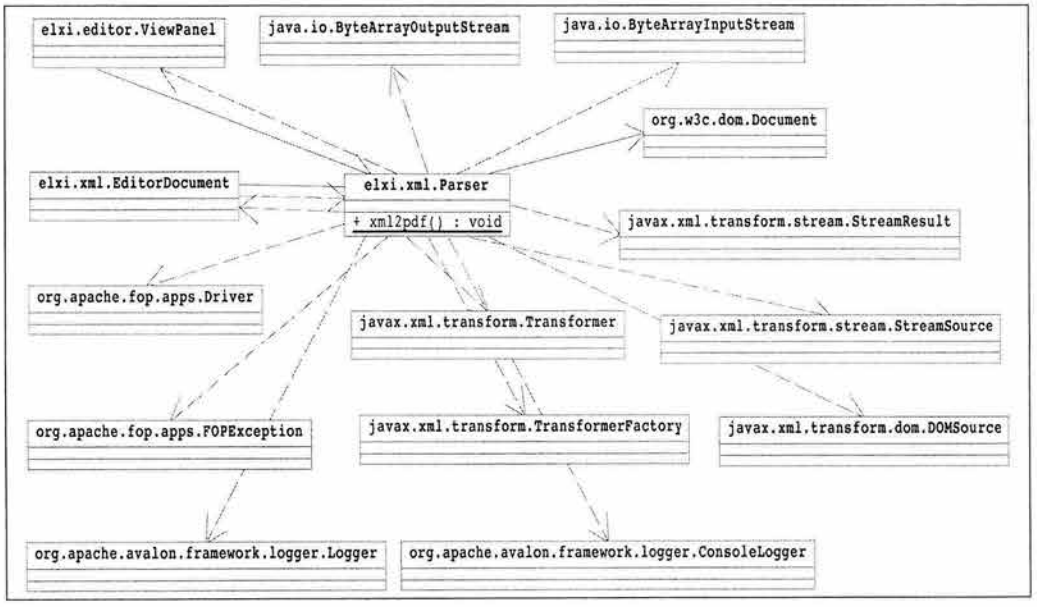

Figura 4.13: Diagrama de clases del motor XSL-FO

Procesador XSL-FO

Un procesador XSL es la aplicación que procesa un documento XML y toma plantillas XSL-FO y lo presenta de manera que una persona lo pueda leer facilmente. Para el proyecto se usará el procesador Java Apache FOP, por su fácil integración al analizador Xerces. Para el editor XTE, se necesita generar varias plantillas XSL-FO que permitan generar formatos distintos a los documentos XML. Estas plantillas se usarán exclusivamente para transformar el documento XML al formato PDF. Las plantillas XSL-FO generadas para el editor son tres:

- Plantilla *standard.fo*. Proporciona una presentación muy básica pero en formato PDF.
- Plantilla *normas.fo*. Basada en los estandares que usa PEMEX pero en PDF.
- Plantilla *w3c.fo*. Basada en estandares del W3C.

La clase Java que permite transformar el documento XML a PDF usando las plantillas XSL-FO es parte también de la clase *elxi.xml.Parser* y el código mas importante se observa en la figura 4.14.

```
package elxi.xml;
import javax.xml.parsers.*;
import org.xml.sax.*;
import org.w3c.dom.*;
import java.io.*;
import elxi.xml.EditorDocument;
    public final class Parser{
      public static Transformer transformer;
      private static javax.xml.transform.TransformerFactory tFactory;
        static {
          try {
            //Definimos la fabrica de transformaciones
            tFactory = javax.xml.transform.TransformerFactory.newInstance();
        //Método que permite realizar la transformación XSL-FO
          public synchronized static void xml2pdf(String name, Document xmlDoc, String xslFile, OutputStream out)
            throws Exception{
            ByteArrayOutputStream bout = null;
            ByteArrayInputStream bin = null;
            Driver driver = null;
            try {
              bout = new ByteArrayOutputStream();
              transformer = tFactory.newTransformer(new StreamSource(xslFile));
              transformer.setParameter("nombre_archivo",name);
              transformer.transform(new DOMSource(xmlDoc), new StreamResult(bout));
              bin = new ByteArrayInputStream(bout.toByteArray());
              Logger logger = new ConsoleLogger(ConsoleLogger.LEVEL_INFO);
              MessageHandler.setScreenLogger(logger);
              MessageHandler.setQuiet(true);
              driver = new Driver(new InputSource(bin),out);
              driver.setLogger(logger);
              driver.setRenderer(Driver.RENDER_PDF);
              driver.run();
            }
            catch (Exception e) {
              System.out.println("Excepción en XMLTransformer.xml2pdf : \n" + e);
            }
          }
    }
```

Figura 4.14: Clase que transforma de XML a PDF por medio de las plantillas XSL-FO

4.7. La aplicación

Ya se tiene la DTD y la clase Java que permite procesar los documentos XML, además de validarlos. También ya están generadas las plantillas XSLT y XSL-FO que permiten la transformación a HTML y PDF y la clase Java que permite estas transformaciones. De esta manera, la aplicación podrá exportar la información que está en formato XML a formatos conocidos y visualizables. Lo que falta es la aplicación o interfaz de usuario que permitirá integrar estos componentes.

El diagrama de clases del editor XTE se visualiza en la figura 4.15:

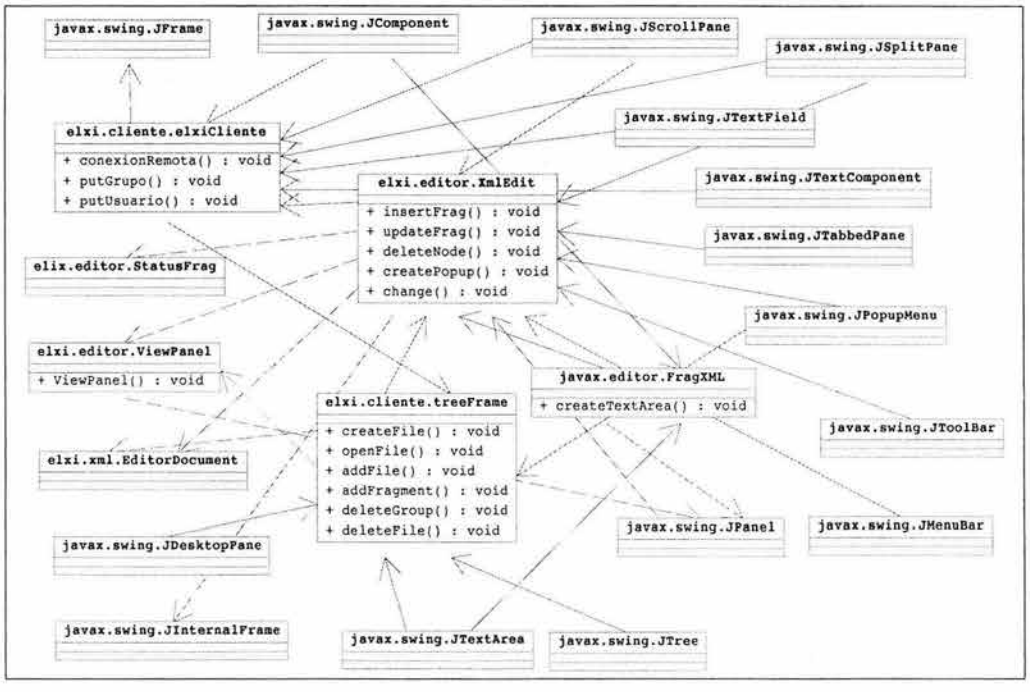

Figura 4.15: Diagrama de clases editor XTE

4.7.1. Componentes del editor XTE

Ahora se necesita desarrollar la interfaz gráfica que permita la edición de los documentos XML y la visualización de los mismos con las plantillas de estilo. Para eso, se desarrolló el paquete en Java llamado *elxi.editor* que es parte del proyecto ELXI. En la figura 4.16 se describen los componentes que conforman al editor colaborativo XTE.

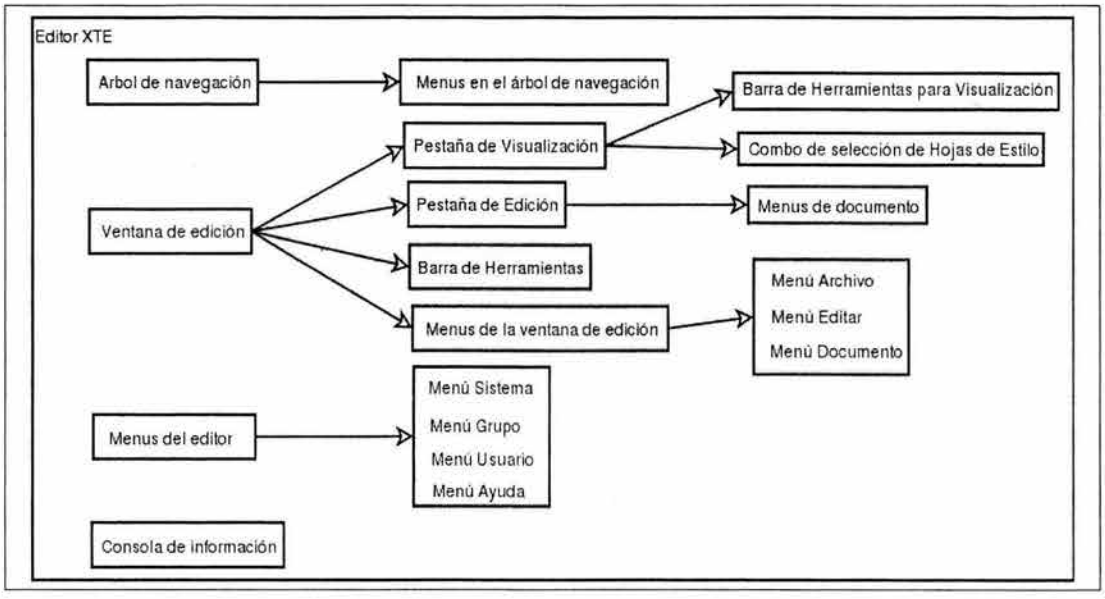

Figura 4.16: Componentes gráficos del editor

Como se puede observar en la figura 4.16, el editor XTE tiene 4 componentes principales: "El árbol de navegación", "Ventana de edición", "Menús del editor" y "Consola de informacion".

En la figura 4.17 se observa como están distribuidos esos componentes dentro de un componente *javax.swing.JFrame* de Java:

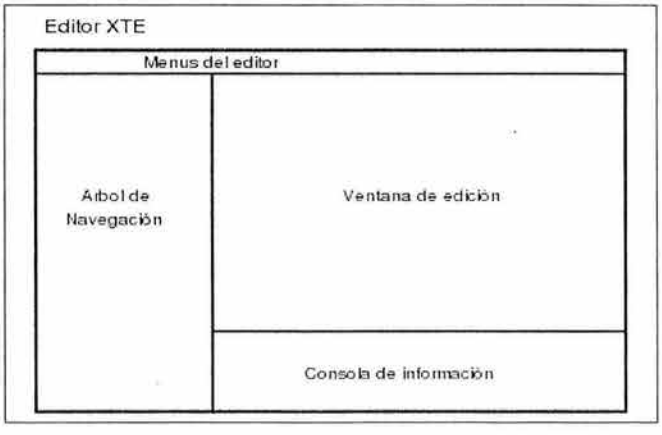

Figura 4.17: Distribución de los componentes principales del editor XTE

A continuación se describen los componentes del editor XTE. Todas las pantallas de los componentes del editor se pueden observar en el apéndice A del presente trabajo.

El árbol de navegación

Permite visualizar en forma de un árbol jerárquico, los grupos, los documentos y los fragmentos que el usuario tiene permiso de visualizar y usar. Este árbol se generá con la información que el servidor ELXI le envía. La clase Java principal de este componente es *elxi.cliente.treeFrame*.

Descripción

En el primer nivel o raíz se tiene un nodo con el nombre del sistema ELXI. En el siguiente nodo o nivel del árbol se tienen a los grupos de trabajo del sistema ELXI. Los grupos pueden ser propios (que unicamente pueden ser vistos por el propio usuario) y otros son grupos compartidos con otros usuarios. En el siguiente nivel del árbol se tienen los documentos, que al igual que los grupos, se tienen documentos propios y otros compartidos con otros usuarios del sistema. No puede haber un usuario que tenga permisos sobre un documento y que no pertenezca al grupo donde pertenece ese documento. En el último nivel del árbol se encuentran los fragmentos del documento. En cada nivel del árbol existe un menú (que se puede abrir con el botón derecho del *ratón*[9] o puntero) que permite interactuar con los elementos del árbol. Seleccionando un grupo se abre el menú de operaciones de grupo. Si es un documento serán operaciones sobre los documentos. Si es un fragmento aparece un menú de operaciones sobre el fragmento. En la figura 4.18 se puede visualizar como esta organizado el árbol de navegación del editor XTE.

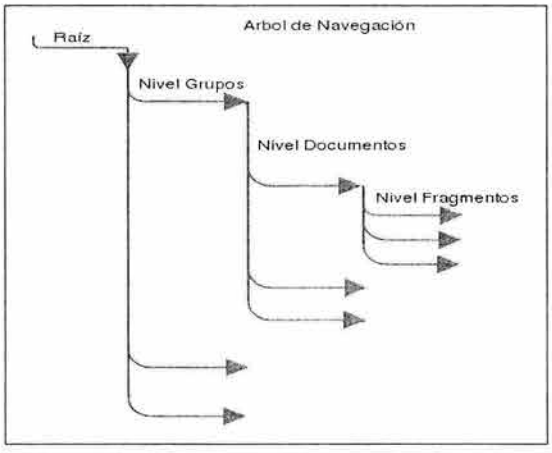

Figura 4.18: Organización del árbol de navegación

En la figura anterior, se puede observar que la raíz está en el primer nivel, después el nivel de los grupos. En el nivel siguiente esta el nivel de los documentos. Después, en el último nivel están los fragmentos. El árbol de navegación puede desplegar un menú en cada nivel de la navegación que permiten realizar operaciones según el nivel en que se encuentre.

- **Menús en el árbol de navegación.** Son un conjunto de menús que permiten interacturar con los documentos y fragmentos según el nivel de navegación.

[9]Del inglés "mouse".

Se tienen los siguientes menús:

- **Nivel raíz.** Se tienen las opciones de:
  - **Agregar grupo.** Permite agregar un nuevo grupo al sistema.
  - **Mostrar grupos.** Muestra la información detallada en una ventana sobre todos los grupos del sistema que sean públicos y que el usuario tenga permiso de visualizar.
  - **Ingresar a un grupo.** Permite ingresar a un grupo en particular. Pero en esta versión no esta activada aun.
  - **Enviar mensaje a todos los usuarios del sistema.** Permite enviar un mensaje instantaneo a todos los usuarios que esten en sesión.
- **Nivel Grupos.** Se tienen las opciones de:
  - **Crear nuevo documento.** Permite crear un nuevo documento dentro del grupo seleccionado y abrirlo en la ventana de edición.
  - **Información del grupo.** Muestra información detallada del grupo seleccionado en una ventana.
  - **Mostrar usuarios del grupo.** Despliega información en una ventana de todos los usuarios que pertenecen al grupo de trabajo.
  - **Agregar usuario al grupo.** Permite agregar un usuario al grupo seleccionado.
  - **Borrar usuario del grupo.** Permite borrar un usuario del grupo seleccionado si el usuario no es dueño de algún documento del grupo y no es el creador del grupo.
  - **Borrar grupo.** Permite borrar el grupo seleccionado. Borra todo su contenido. Solo el dueño del grupo puede eliminarlo.
  - **Enviar mensaje a todos los usuarios del grupo.** Permite enviar un mensaje a todos los usuarios que estén en sesión y pertenezcan al grupo seleccionado.
- **Nivel Documentos.** Se tienen las opciones de:
  - **Abrir.** Permite abrir el documento seleccionado en la ventana de edición.
  - **Información del documento.** Despliega información detallada del documento seleccionado en una ventana.
  - **Permisos del documento.** Despliega información detallada sobre los permisos del documento seleccionado. Muestra la siguiente información:

Usuario	Nombre	Administrador	Agregar fragmentos	Editar-Leer documento	Sin permisos
Usuario del sistema.	El nombre del usuario.	Que usuario es el dueño del documento.	Que usuarios pueden agregar fragmentos al documento.	Que usuarios pueden leer o editar el documento.	Que usuarios no tienen ningún permiso sobre el documento.

Tabla 4.1: Permisos del documento

- **Borrar.** Permite borrar el documento seleccionado si el usuario es el administrador del documento.
- **Enviar mensaje a todos los usuarios del documento.** Permite enviar un mensaje a todos usuarios que estén en sesión y tengan permiso de Administrador o tengan el permiso de Editar-Leer sobre el documento seleccionado.

- **Nivel Fragmentos.** Se tienen las opciones de:
  - **Información del fragmento.** Despliega información detallada del fragmento seleccionado.
  - **Permisos del fragmento.** Despliega información detallada sobre los permisos del fragmento seleccionado. Si hay mas de un usuario que tenga permisos sobre el fragmento se despliegan varios renglones por cada usuario. Describe la siguiente información:

Usuario	Nombre	Administrador	Escritura	Lectura
Usuario del sistema.	El nombre del usuario.	Que usuario es el dueño del fragmento.	Que usuario tiene permiso de escribir sobre el fragmento seleccionado.	Que usuario tiene permiso de leer el fragmento seleccionado.

Tabla 4.2: Permisos del fragmento

Ventana de edición

En esta ventana se realizan las operaciones de edición y visualización de los documentos. Aparece cuando requiere editar un documento existente desde el árbol de navegación en el menú de *"Nivel Documento"* $\longrightarrow$ *"Abrir"*. También aparece cuando se requiere editar un documento nuevo y desde el árbol de navegación en el menú de *"Nivel Grupos"* $\longrightarrow$ *"Crear nuevo documento"*.

Descripción

Es la ventana donde se realiza la edición de los documentos y su visualización. Aparece del lado derecho del área de trabajo del editor XTE. El usuario abre o crea un documento y éste aparece en el área de edición dentro de la pestaña de edición. Una vez abierto el documento, el usuario puede visualizarlo dando un click en la pestaña de visualización.

En la figura 4.19 se describen los componentes de la ventana de edición.

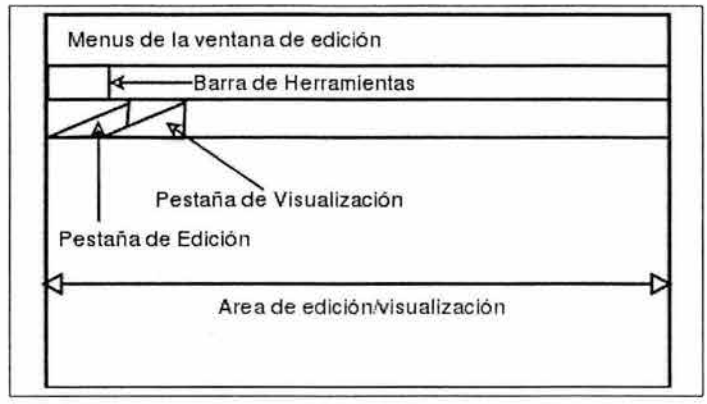

Figura 4.19: Componentes de la ventana de edición

- **Pestaña de visualización.** Permite visualizar el documento que el usuario esta editando a manera de pre-visualización del documento. Dentro del panel de visualización, se podrá observar cual será el formato del documento para su posterior exportación a HTML o PDF. Este panel permite escojer el tipo de hoja de estilo para generar distintos tipo de salida y formato. Cuando se esta dentro de la pestaña de visualización el área de edición se convierte en el área de visualización. La figura 4.20 describe sus componentes.

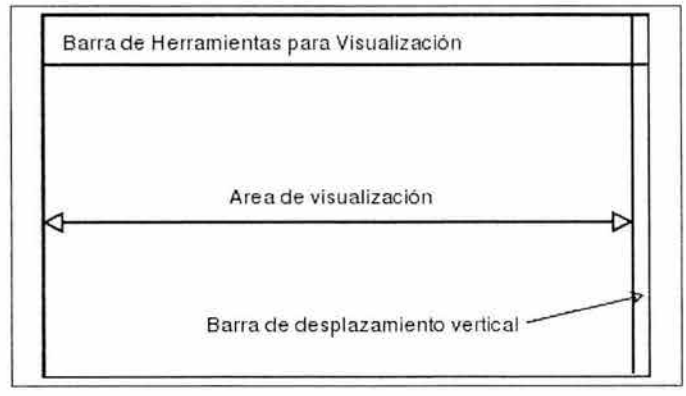

Figura 4.20: Componentes de la pestaña de visualización

- **Barra de Herramientas para Visualización.** Contiene las siguientes opciones:
  - **Re-construir el documento XML**. Es un botón que permite procesar el documento realizando la transformación del formato XML al formato HTML con el motor de transformación XSLT Xalan. El resultado es desplegado en el área de edición.
  - **Salvar HTML.** Es un botón que permite salvar el documento HTML recien generado.
  - **Generar PDF.** Es un botón que permite exportar el documento XML al formato PDF por medio del motor de transformación XSL-FO.
  - **Combo de selección de hojas de estilo.** Es un componente que permite seleccionar entre tres hojas de estilo en cascada (*W3C, Normas o Standard*), para establecer que

formato de salida se desea. Esta hoja de estilo se aplica al documento HTML que se genera despues de la transformación XSLT para enriquecer el formato de salida.

- **Barra de progreso horizontal**. Es una barra que despliega el estado de progreso del proceso de transformación XSLT. Va desplegando un porcentaje y rellenando la barra de color azul conforme el proceso de transformación XSLT va llevandose a cabo hasta alcanzar el cien por ciento del proceso.

- **Pestaña de edición.** Permite visualizar un documento para su edición. El documento se despliega en el panel para su edición. Esta pestaña tiene una barra de desplazamiento vertical del lado derecho del panel. Esta barra de desplazamiento sirve para recorrer el documento de arriba a abajo. Dentro de la pestaña de edición el documento parece no estar dividido por fragmentos, pero cuando el usuario da un click con el ratón en alguna región del área de edición, todo el contorno del área que comprende a un solo fragmento se ilumina de color rojo permitiendo visualizar de forma detallada el tamaño del fragmento. Una vez seleccionado el fragmento, inicialmente se encuentra en modo de solo lectura. Si se desea editar el fragmento, se debe solicitar un bloqueo del fragmento, si el fragmento no es editado por otro usuario desde otro editor XTE y tiene permiso de escritura, el editor permite la edición del fragmento. Para editar el fragmento, el área de su contorno va creciendo hacia abajo conforme se edita una nueva línea. Al llegar al final de la línea, el editor salta a una nueva línea de forma automática. No hay restricción en el número de lineas que puede contener un fragmento y la barra de desplazamiento se mueve hacia abajo si el número de lineas aumenta. La figura 4.21 describe sus componentes.

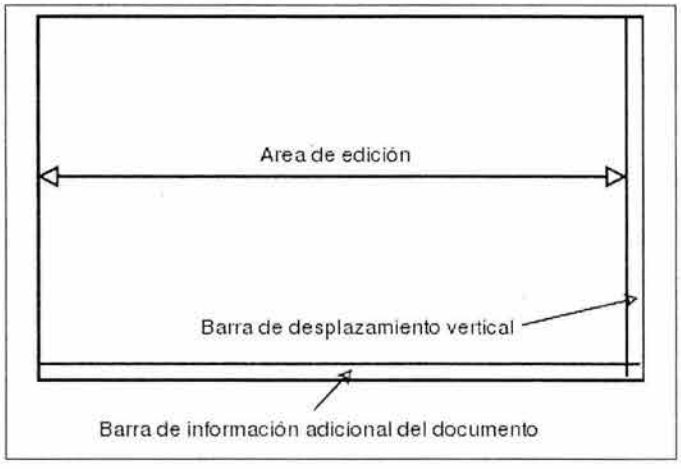

Figura 4.21: Componentes de la pestaña de edición

El documento puede ser seleccionado de dos formas:

- ***Documento existente.*** Para abrir un documento existente se debe seleccionar del árbol de navegación en el nivel de documentos y con el botón derecho del ratón escoger la opción de *"Abrir"*, el documento se abre y se visualiza en el área de edición.

- ***Documento nuevo.*** En el caso de nuevos documentos se debe seleccionar un grupo (en el nivel de grupos) en el árbol de navegación y con el botón derecho del ratón escojer la opción *"Crear nuevo documento"*. Aparece un recuadro con la leyenda de *"Crear nuevo documento"*. Este recuadro requiere los siguientes datos:
  - Nombre. El nombre del documento a crear.
  - Tipo. El tipo de documento que puede ser únicamente de dos tipos: *"General"* y *"Norma IMP"*. Por default es *"General"*. Esta opción sirve para determinar como será procesado el documento durante la visualización.
  - Fragmentos. El número de fragmentos determina en cuantos pedazos estará dividido el documento para realizar la edición colaborativa. Al inicio los fragmentos creados pertenecerán al usuario que creó el documento.
  - Grupo. Es el grupo que ya fue establecido desde que se selecciono en el árbol de navegación y no es un campo modificable.

Al dar click al botón *"Crear"*, el editor XTE se conecta al servidor ELXI, crea el documento y abre un nuevo documento en el área de edición. Se actualiza el árbol de navegación, en el nivel de documentos, aumentando con un nodo más con el nombre del documento recien creado.

- **Menús de documento.** Una vez abierto el documento en el área de edición y listo para ser editado (si el usuario tiene los permisos necesarios), aparecen sobre el documento menús desplegables que funcionan dependiendo si el usuario se encuentra conectado o desconectado. Estos menús aparecen cuando se da click con el botón derecho del ratón en algún fragmento del documento.
  - *Menú del documento cuando el usuario está conectado al servidor.* Este menú aparece cuando se selecciona un fragmento del documento dando un click con el botón derecho del ratón. Aparecen las opciones de:
    - **Cortar.** Permite cortar el texto plano que haya sido seleccionado y colocarlo en memoria para edición.
    - **Copiar.** Permite copiar texto plano que haya sido seleccionado y colocarlo en memoria para edición.
    - **Pegar.** Permite pegar texto plano y colocarlo en el área de edición.
    - **Información.** Permite visualizar datos adicionales del fragmento. Despliega la siguiente información: ***Creador, Versión.***
    - **Permisos.** Permite visualizar los permisos del fragmento en una tabla. Despliega la misma información de la Tabla 4.2.
    - **Guardar fragmento.** Esta opción permite guardar la información en el servidor ELXI y el servidor se encarga de replicar la información a todos los usuarios que estén en sesión en esos momentos. Para poder realizar está operación

se necesita que este bloqueado el fragmento, en caso contrario despliega un mensaje de error.

- ◇ **Cambiar título.** Permite cambiar el título del fragmento. Este título es el nombre del fragmento que se despliega en el árbol de navegación.
- ◇ **Establecer tipo.** Esta opción permite establecer el tipo del fragmento. El tipo permite definir como el fragmento será tratado cuando se necesite visualizarlo en la pestaña de visualización.
- ◇ **Bloquear.** Esta opción permite solicitar un bloqueo de fragmento al servidor ELXI. El servidor responde si es posible bloquearlo cambiando el estado del fragmento de no editable a editable. El servidor replicá el nuevo estado del fragmento a todos los demas usuarios que estén en sesión y se encuentren editando el mismo documento.
- ◇ **Fragmento.** Está opción permite desplegar otro menú de operaciones de fragmento que son:
  - ⇒ Insertar un fragmento arriba. Inserta un nuevo fragmento arriba del seleccionado. Se actualiza el árbol de navegación con el nuevo nodo de fragmento.
  - ⇒ Insertar un fragmento abajo. Inserta un nuevo fragmento abajo del seleccionado. Se actualiza el árbol de navegación con el nuevo nodo de fragmento.
  - ⇒ Borrar fragmento. Borra el fragmento seleccionado. Se actualiza el árbol de navegación borrando el nodo.

- ○ *Menú del documento cuando el usuario está desconectado.* Este menú aparece cuando se selecciona un fragmento del documento dando un click con el botón derecho del ratón. Aparecen las opciones de: *Cortar, Copiar y Pegar.* Este menú es el mismo al de *"Ventana de edición"* ⟶ *"Menús de documento"* ⟶ *"Menú del documento cuando el usuario está conectado".*

- **Barra de Herramientas.** La barra de herramientas permite las operaciones clásicas de edición de texto que son *Cortar, Copiar y Pegar.* Este menú es el mismo al de *"Ventana de edición"* ⟶ *"Menús de documento"* ⟶ *"Menú del documento cuando el usuario está conectado".*

- **Menús de la ventana de edición.**

  Estos menús permiten realizar operaciones sobre los documentos y sus fragmentos.

  - ○ **Menú Archivo.** Este menú solo contiene la opción *"Salir"* y permite cerrar la ventana de edición.
  - ○ **Menú Editar.** Este menú tiene las mismas opciones que la barra de herramientas. Este menú es el mismo al de *"Ventana de edición"* ⟶ *"Menús de documento"* ⟶ *"Menú del documento cuando el usuario está conectado".*
  - ○ **Menú Documento.** Este menú contiene dos opciones:

- ◇ **Información del documento.** Esta opción es la misma descrita en *"Menús en el árbol de Navegación"⟶ "Nivel Documentos"⟶ "Información del documento"*.
- ◇ **Permisos del documento.** Está opción es la misma descrita en *"Menús en el árbol de Navegación"⟶ "Nivel Documentos"⟶ "Permisos del documento"*.

Menús del editor

Son menús que permiten realizar operaciones generales del editor XTE. Se encuentran en la ventana principal. A continuación se describen.

- **Menú Sistema.** Este menú realiza operaciones de conexión al servidor ELXI. Contiene las opciones de:
  - **Ingresar.** Esta opción permite conectarse el servidor ELXI. Por default, cuando el usuario se conecta y no es su primera vez en sesión, el cliente ELXI trabaja sin conexión. De esta forma, todo lo que realiza se hace sin estar conectado. Se dice esta en modo *no colaborativo* porque todo lo que realize el usuario sobre los documentos se verá reflejado solamente localmente. Cuando se conecta el cliente ELXI se sincroniza con el servidor. De esta manera, todos los cambios que hayan realizado el usuario y los demás usuarios de ELXI se combinan generando una versión única y actualizada de todos los documentos y grupos. También se debe estar conectado para poder realizar operaciones colaborativas dentro del sistema ELXI, como son *"Agregar grupo"*, *"Crear nuevo documento"*, *"Borrar grupo"*, etc.
  - **Desconectar.** Esta opción permite desconectar del servidor ELXI y trabajar en modo *no colaborativo*. Cuando se está desconectado aparece la leyenda "Para acceder a todas las opciones debes conectarte remotamente" en muchas de las opciones del editor XTE que tiene que se necesita estar conectado para que funcionen normalmente.
  - **Salir.** Permite salir del editor XTE.
- **Menú Grupo.** Este menú permite realizar operaciones sobre grupos de trabajo del sistema ELXI. Estos grupos, como se mencionó antes, pueden ser públicos o privados. Todas las opciones de éste menú ya fueron descritas en el árbol de navegación.
  - **Agregar grupo.** Este menú es el mismo al de *Arbol de Navegación ⟶ Menús en el árbol de navegación ⟶ Nivel Raíz ⟶ Agregar grupo.*
  - **Borrar grupo.** Este menú es el mismo al de *Arbol de Navegación ⟶ Menús en el árbol de navegación ⟶ Nivel Raíz ⟶ Nivel Grupos ⟶ Borrar grupo.*
  - **Mostrar grupos.** Este menú es el mismo al de *Arbol de Navegación ⟶ Menús en el árbol de navegación ⟶ Nivel Raíz ⟶ Mostrar grupo.*

- **Ingresar a un grupo.** Este menú es el mismo al de *Arbol de Navegación* $\longrightarrow$ *Menús en el árbol de navegación* $\longrightarrow$ *Nivel Raíz* $\longrightarrow$ *Ingresar a un grupo.*

- **Menú Usuario.**

  - **Agregar usuario.** Este menú permite agregar un usuario al sistema. Sólo los usuarios que tengan el permiso de *"Administrador"* pueden agregar un usuario. Al seleccionar aparece una ventana que solicita los datos de: *Clave, Contraseña, Nombre y Correo.* Un botón de *Agregar* permite agregar al usuario.
  - **Borrar usuario.** Sólo los usuarios con permiso de *"Administrador"* están autorizados a borrar usuarios.
  - **Mostrar usuarios.** Despliega una ventana con información de todos los usuarios del sistema. Los datos que proporciona son: *Clave, Usuario, Email, Conectado y Grupo.*
  - **Enviar mensaje a un usuario.** Permite enviar un mensaje a un usuario que este conectado el sistema. Al seleccionar aparece una ventana donde solicita: *Usuario y Mensaje.* Un botón de *Enviar* permite enviar el mensaje.
  - **Enviar mensaje a todos los usuarios del sistema.** Este menú es el mismo al de *"Arbol de Navegación"* $\longrightarrow$ *"Menús en el árbol de navegación"* $\longrightarrow$ *"Nivel Raíz"* $\longrightarrow$ *"Enviar mensaje a todos los usuarios del sistema".*
  - **Cambiar contraseña.** Esta opción permite cambiar la contraseña del usuario. Al seleccionar aparece una ventana donde solicita los datos de: *Contraseña anterior, Nueva contraseña y Nueva Contraseña (repetición).* Un botón de *Cambiar* permite actualizar la contraseña.

- **Menú Ayuda.** Esta opción sólo tiene una opción de *Acerca de,* la cual despliega una ventana donde muestra información de versión, fecha y nombre de la aplicación.

Consola de información

Es una ventana que despliega información relacionada a operaciones que se realicen dentro del sistema. Todos los mensajes que aparecen en esta ventana son informativos. Sirve para que los usuarios puedan estar seguros que muchas de las operaciones realizadas dentro del editor se llevaron con éxito. Si se desea esta ventana puede ocultarse.

4.8. Resumen

Se describió el editor XTE en su funcionamiento básico y la forma en que está integrado al sistema de edición colaborativa ELXI. Se muestra cada proceso involucrado en el editor que permite

su funcionamiento colaborativo. Se describió el proceso de análisis sintáctico, el cual es el responsable de procesar los documentos XML para mantener su validez. También se describió el proceso de generación de plantillas para la visualización por medio de la transformaciones de XSLT y XSL-FO.

Se describe cómo el editor XTE permite editar los documentos de forma colaborativa con manejo de permisos, grupos, bloqueo de fragmentos y todo por medio de la interfaz de usuario. Además, permite la visualización de los documentos de una forma amigable para el ojo humano. Se realizó una descripción de cómo ésta compuesto el editor y sus diferentes componentes.

Finalmente, se llevo a cabó el desarrollo de la aplicación, descrito por etapas y por componentes fundamentales. Cada uno de los componentes es descrito y desglosado en su interacción con el usuario. También se describen los diagramas de clases Java desarrolladas para crear la aplicación. La aplicación es funcional y tiene las suficientes herramientas para realizar con éxito la edición colaborativa.

Conclusiones y perspectivas

El presente trabajo permitió observar en funcionamiento varias tecnologías recientes que están haciendo su aparición en los últimos años y que cada vez están convirtiendose en los nuevos estandares que funcionarán sobre la red Internet y los sistemas de cómputo en los años subsecuentes.

La tecnología de XML es una de las más recientes tecnologías que están cambiando el cómo está estructurada la información que se maneja sobre los sistemas de computo y sobre las redes de computadoras. Un aspecto muy importante que se observó es la funcionalidad y exportabilidad que da el formato XML. Para los programadores es un formato que al ser de texto plano es de fácil depuración y permite una identificación rápida de los datos. Además, al ser jerárquico (debido a su estructura de árbol) permite busquedas rápidas y faciles de los datos.

A su vez, otras nuevas tecnologías están surgiendo a partir del XML, como los espacios de nombre ***(NameSpaces)*** que permiten la construcción de nombres universales y únicos sin posibilidad de colisión, vital para poder intercambiar información sin riesgo de errores, el lenguaje de busquedas XML (XQL[10]) que permite realizar búsquedas especializadas y extraer datos de los documentos XML, XLINK que permite inscrustar vínculos entre los documentos XML, etc.

Nuevas tecnologías de bases de datos hacen uso del formato XML para guardar, ordenar y clasificar la información. Las aplicaciones cliente hacen uso del formato XML para configurar la forma en que la aplicación se visualiza, como se configura e incluso de cual es la posición de todos los elementos dentro de la misma. Las aplicaciones servidor lo usan para configurar todos sus parámetros de sus aplicaciones y de sus modulos. Estas mismas aplicaciones servidor usan al XML como formato de comunicación y distribución de los datos. Empresas que desarrollan nuevos protocolos de comunicación en la red Internet implementan sus aplicaciones haciendo uso de XML como base de comunicación e intercambio de los datos. Por ejemplo, el servidor de mensajería instantánea *Jabber*[11] y los servicios de SMS[12] para dispositivos de comunicación móbil (GSM[13]), entre otros.

La tecnología XML permite al fin, tener el control de los datos y de toda la información. Además,

[10]Del inglés "XML Query Language".

[11]Jabber es un protocolo abierto basado en el estándar XML para el intercambio en tiempo real de mensajes. El sitio oficial del proyecto es http://www.jabber.org

[12]Del inglés "Short Message Service".

[13]Del inglés "Global System for Mobile Communications".

separa perfectamente la parte de contenido de la parte de visualización por lo que el usuario ya no se debe preocupar por cómo se visualiza el documento para concentrarse exclusivamente en la edición y el contenido. Para la visualización basta con definir en que formato se quiere visualizar y escojer que formato se desea obtener.

Los lenguajes de programación incorporan cada día nuevas herramientas para la manipulación de los documentos XML. El lenguaje Java es el lenguaje que más apoyo ha generado al estandar XML y existen herramientas de todo tipo para ese lenguaje.

Algunas de esas herramientas fueron utilizadas en el presente trabajo como el analizador sintáctico Java Xerces que es una herramienta que permite validar correctamente los documentos XML y realizar todo tipo de manipulación de los archivos (insertar, borrar y modificar nodos de documentos XML). Además, el motor de transformación XSLT Java Xalan permite realizar las transformaciones necesarias para todo el proceso de visualización. El motor Java FOP de transformación XSL-FO que es una herramienta muy poderosa de transformación que permite generar documentos PDF por medio de hojas de estilo XSL-FO.

También otro punto muy relevante en el presente trabajo, es el proceso colaborativo que la aplicación lleva a cabo para realizar la edición de los documentos. El editor permite realizar la edición colaborativa de una forma rápida y sencilla. Hay que tener siempre presente que el editor necesita de los otros componentes del sistema ELXI y sin ellos no sería una herramienta completa y funcional. El corazón del sistema ELXI por lo tanto, no es el editor solamente, sino también son la comunicación entre los distintos cliente ELXI, el control de concurrencia y la consistencia de los documentos replicados. Todo en conjunto constituyen un sistema cliente/servidor que permite realizar las procesos para el cual fueron creados.

Otro aspecto importante y de aporte del sistema ELXI es el corazón del sistema, el cual puede ser utilizado no sólo para la edición colaborativa de documentos, sino también, para otros procesos que requieran este tipo de funcionamiento, como podrían ser pizarrones de dibujo colaborativos, juegos colaborativos, salas de chat colaborativas, etc. Ya hay herramientas de ese tipo en Internet, pero la mayoría utiliza tecnologías propietarias que no permiten ser reutilizadas, pero el sistema ELXI utiliza al lenguaje de programación Java que permite realizar aplicaciones que pueden funcionar sobre distintas arquitecturas, es modular y la reutlización de código es bastante sencilla; y utiliza el lenguaje XML que es una tecnología abierta y estandar, por lo que es relativamente fácil la reutilización de código para otros fines y propositos.

El editor es funcional y continua en constante desarrollo pero ya constituye una opción real y funcional para el trabajo de edción colaborativa de documentos de tipo científico. Los investigadores y usuarios en general pueden utilizarla como una herramienta que les permite editar sus documentos en forma conjunta y sin necesidad de tener que esperar a los otros para poder ver el resultado final y sin generar inconsistencias en el documento compartido.

En un ambiente empresarial puede ser usada como una herramienta lo suficientemente madura, estable y funcional para ser utilizada en producción. En las áreas donde se requiera una herramienta para editar documentos de texto que necesiten ser editados de forma conjunta y sincro-

nizada, con control de grupos, permisos de documento incluso a nivel de fragmentos, asignación de roles con niveles distintos de jerarquia. Es una opción de fácil implementación y que comparada a productos comerciales del mismo tipo (que son bastante caros y complejos) representa una opción que puede ser evaluada.

Sin embargo, aun se debe mejorar en varios aspectos como son: dar mayor facilidad en la edición que permita que la edición sea aun mas amigable por medio de más y mejores herramientas gráficas; la interfaz de usuario debe ser mas robusta e intuitíva con el usuario; la posibilidad de dar mayor formato de visualización a los documentos por medio de plantillas de diseño mas poderosas e incluso que sean configurables desde la misma aplicación; la posibilidad de poder crear mucho más tipos de documentos y plantillas que permitan incrustar objetos mas complejos como tablas, imagenes, vinculación interna en los documentos, etc.

El sistema ELXI y el editor XTE continua en desarrolló y ofrece muchas posibilidades de crecimiento. Cada día se ha ido convirtiendo en una herramienta más poderosa.

Evaluación, resultados y discusión

Todo el proceso de evaluación fue documentado con anterioridad en un artículo realizado por el Dr. Manuel Romero Salcedo, Silvia Ramirez, Cesar Murillo y quien suscribe, para el evento que tenía por nombre "Computo Inteligente en la Industria Petrolera" [Elx02]. Las siguientes líneas son un resumen y análisis del mismo.

La evaluación del sistema ELXI y resultados obtenidos fue realizada gracias un cuestionario y proceso de evaluación llevado a cabo en el Instituto Mexicano del Petróleo, el cual se llevó a cabo gracias a la entusiasta participación de un grupo de autores. Desde un inicio, los autores mostraron un gran interés en conocer el prototipo y en el hecho de trabajar en equipo para la elaboración de un tipo de documento. La versión que fue sujeta a evaluación fue la 1.0, la cual es la que se presenta en esta tesis.

Antes que nada, los autores recibieron algunas horas de capacitación para usar y familiarizarse con el sistema ELXI. En todos los sitios donde los autores trabajaron se instaló el sistema ELXI. Antes de arrancar la edición colaborativa, los autores se reunieron frente a frente para decidir quien seria el administrador del documento y qué roles tomarían el resto de los autores. El trabajo de edición colaborativa duró una semana. Al finalizar, cada uno de los autores evaluó el sistema ELXI, respondiendo a un cuestionario de aproximadamente 35 preguntas relacionadas con diferentes temas (conciencia de grupo, funcionalidad del editor, control de concurrencia, replicación de la información, etc.). Además, durante la semana de trabajo, tuvimos periodos de observación de las acciones de edición de cada uno de los autores, a quienes visitamos en sus oficinas y aprovechamos para hablar con ellos y registrar sus comentarios y resolver sus dudas.

Se presentarán los resultados de evaluación referentes a la edición colaborativa. Con el fin de conocer el grado de funcionalidad y aceptación del mecanismo de control de concurrencia, analizaremos las respuestas a las preguntas concernientes a ese tema. Las preguntas aplicadas se reproducen en la figura 1a. Sus respuestas fueron computadas, asignando valores a cada una de las cinco opciones que tenían para contestar cada pregunta: Excelente (valor=5), Bueno (valor=4), Regular (valor=3), Deficiente (valor=2) y Malo (valor=1). Así, el máximo valor que puede tener una respuesta es 5 y el mínimo es 1.

1. ¿Cómo califica la opción de crear grupos en ELXI para identificar a los diferentes equipos de trabajo?
2. ¿Considera una buena medida que cualquier autor de ELXI pueda crear un grupo de trabajo?
3. ¿Es una buena medida que los creadores de documentos (administradores de documento) establezcan que miembros del grupo pueden obtener el documento?
4. ¿Cómo califica la división del documento en fragmentos?
5. ¿Cómo califica la existencia de diferentes roles de documento para controlar la creación del documento?
6. ¿Cómo califica la existencia de diferentes roles de fragmento para controlar la edición de fragmentos?
7. ¿Qué opina de los diferentes colores que el icono, ubicado en la parte inferior izquierda del editor, adquiere para indicar el estado del fragmento seleccionado? Los estados son BLANCO(disponible para editar), ROJO(no disponible para editar) y VERDE(editando).
8. ¿Cómo considera usted la opción de especificar el tiempo de bloqueo para editar un fragmento?
9. El máximo tiempo de bloqueo es de 23:59 hrs. ¿Considera que es suficiente este tiempo para editar un fragmento?
10. ¿Cómo considera usted la medida de apartar su turno de bloqueo de un fragmento cuando el fragmento ya esta bloqueado por otra persona?
11. Si ha apartado su turno de bloqueo ¿Cómo califica la forma en que obtiene el bloqueo cuando su turno de edición llega?
12. ¿Cómo califica el proceso de bloquear-editar-guardar, para hacer efectivos sus aportes a un fragmento?

Figura 1a: Selección de preguntas concernientes al editor colaborativo

Los resultados obtenidos del cuestionario de evaluación se muestran en la gráfica de la figura 2b.

Figura 2b: Representación gráfica de los resultados obtenidos del cuestionario

En la figura 2b es posible notar que la respuesta a la pregunta 1 no fue tan positiva. Sin embargo, este resultado no se debió a que la forma de poder crear grupos estuviera funcionando mal, sino que se hizo notar que, al incorporar un conjunto de autores a un grupo, era preferible poder seleccionarlos de una lista y agregarlos al mismo tiempo, en lugar de especificar el nombre de cada uno de manera individual. De cualquier forma, la mayoría de los autores opinaron que la creación de los grupos contribuye a la organización del trabajo de edición colaborativa, así como al control de acceso de los documentos. Un punto negativo que encontraron los autores, el cual es importante destacar, concierne a las respuestas obtenidas de la pregunta 2. Todos los autores consideraron una mala opción que cualquier autor en el sistema ELXI pudiera crear un grupo de trabajo. Las razones que expusieron fueron que se trataba de un principio que iba en contra de las políticas de seguridad del Instituto. En efecto, todo el personal cuenta con perfiles bien definidos que le permiten manipular o accesar a ciertas informaciones del Instituto. Esto con el fin de prevenir robos de información o acceso a la misma sin autorización. Por tanto, los autores opinaron que la seguridad estaba en riesgo si se permitía que cualquier autor de ELXI pudiera crear un grupo de trabajo sin autorización. Una solución que propusimos a este problema fue la de establecer que el

administrador del sistema fuera quien pudiera crear los grupos. En caso de mayor necesidad de flexibilidad, entonces se podría implementar un rol adicional de autor en el documento que les permitiera crear grupos. En las respuestas a la pregunta 4, los autores expusieron que la división de un documento fragmentos era un concepto nuevo para ellos. No tuvieron problema para asimilarlo ya que las diferentes partes del documento se identificaban claramente. Sin embargo, opinaron que era importante que el sistema ELXI integrara una herramienta síncrona para llevar a cabo una fase de planeación y/o consolidación que les permitiera discutir y decidir, entre otras cosas, sobre cual sería la mejor división y repartición de fragmentos de un documento.

Debido a la forma en cómo el grupo de autores organizó su forma de trabajo, se observó que no explotaron al máximo algunas de las características que ofrece el mecanismo de concurrencia. Por ejemplo, en ELXI se estableció que se podía bloquear un fragmento, cuyo tiempo máximo podía ser hasta 23:59 hrs. Ésto con el fin de brindar al autor la oportunidad de seguir editando aún cuando se encontrara trabajando en algún otro sitio. Sin embargo, en la pregunta 9, observamos que las respuestas que los autores dieron sobre el tiempo de bloqueo fueron que éste era suficiente con 8 hrs., ya que eso duraba la jornada de trabajo y no tenían la necesidad de cambiarse físicamente de lugar para seguir trabajando en su documento.

Todos los autores estuvieron de acuerdo con las medidas expuestas en la pregunta 10 y 11. Sin embargo, hicieron algunos comentarios concernientes a que se deberían también considerar prioridades en las solicitudes de los autores. De tal manera que cuando llegue una solicitud de carácter urgente se lleve a cabo antes que las otras que se encuentran en lista de espera.

Asimismo, estuvieron de acuerdo en que los administradores de documentos deben ser los únicos que deben definir qué miembros del grupo pueden recuperar una copia de los documentos. Del mismo modo, manifestaron su buena aceptación a los roles de los autores en el documento (pregunta 5) y en los fragmentos (pregunta 6) como una forma de asignación de responsabilidades a los autores del sistema.

Aunque el concepto de fragmentos de documento era nuevo para ellos, el proceso de bloquear, editar y guardar las modificaciones fue calificado como bueno, ya que comprobaron que realmente se controla la edición concurrente de una sección del documento.

Por otro lado, opinaron que el administrador de documento debería contar con mayores facultades, como podría ser la de controlar totalmente las operaciones que se pueden hacer sobre los documentos (ej. borrar documentos, quitar bloqueos aun cuando el autor esta escribiendo, modificar los roles de los autores en los fragmentos, etc). Estas opiniones fueron muy interesantes. Sin embargo, después de un trabajo de reflexión, pensamos que la implementación de estas opiniones podría acarrear consecuencias y penalizaciones durante el trabajo de edición colaborativa (ej. no disponibilidad de la información, perdidas de información, etc.). Gracias a los roles de los autores, es posible controlar toda acción de escritura sobre los fragmentos de un documento. Un aspecto interesante es que los roles de un administrador, ya sea de fragmento o de documento, no lo habilita para borrar la información que algún autor haya incorporado en un fragmento.

La idea de todo esto es que no se pierda ninguna contribución, ni ningún cambio, sino hasta que sea el mismo autor, responsable de su texto, quien lo decida.

Dentro de las opiniones que se recabaron del grupo de autores, se sugirió incorporar en el sistema ELXI una herramienta que permitiera configurar la asignación de responsabilidades a los autores del sistema. Aquí tenemos como idea que en cada grupo se elabore un diagrama de jerarquías, que permita especificar si existirá una máxima autoridad y quiénes serán los autores subordinados. Asociado a cada elemento de este diagrama, se especificarían las actividades que el autor en el documento puede desempeñar.

En general, pudimos concluir, una vez analizadas todas las respuestas en los diferentes temas, que el grupo de autores se sintió de verdad con la oportunidad de externar abiertamente sus opiniones, ya que sabían que iban a tomarse en cuenta para mejorar el sistema ELXI. Ahora bien, antes de hacer que todas estas opiniones se conviertan en diseños e implementaciones para ELXI, deseamos reflexionar más sobre cada una de ellas. Sobre todo, deseamos estar seguros de que siempre mantendremos un sistema configurable a las necesidades de los diferentes autores según el tipo de documentación técnica que decidan elaborar y un sistema flexible a la forma de trabajo de la institución o empresa donde se utilice.

APENDICES

Apéndice A

Descripción de pantallas del editor XTE

- La ventana de Login.

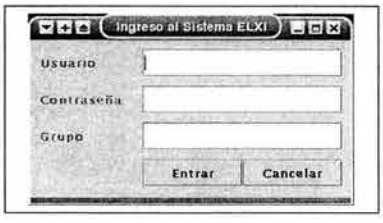

Figura 1: Ventana de Login

- El editor XTE.

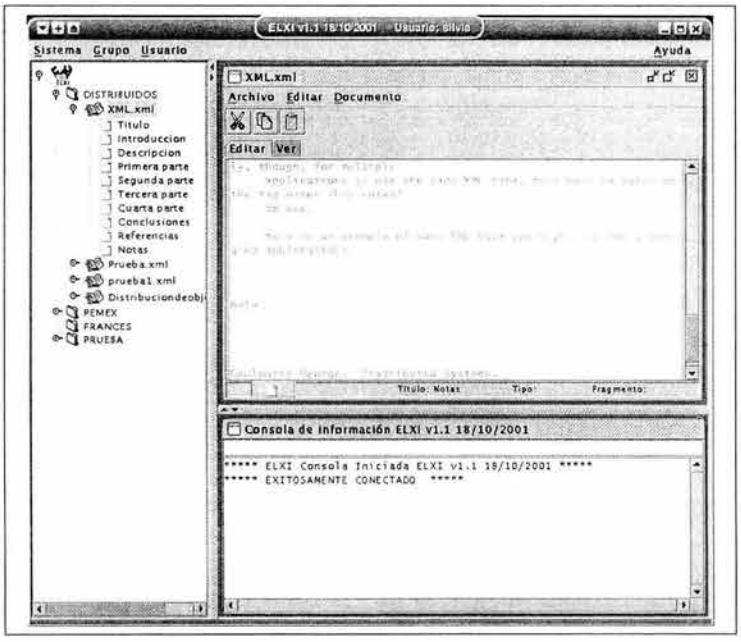

Figura 2: Editor XTE

- El arbol de navegación.

Figura 3: Arbol de navegación

- Ventana de edición y barra de herramientas

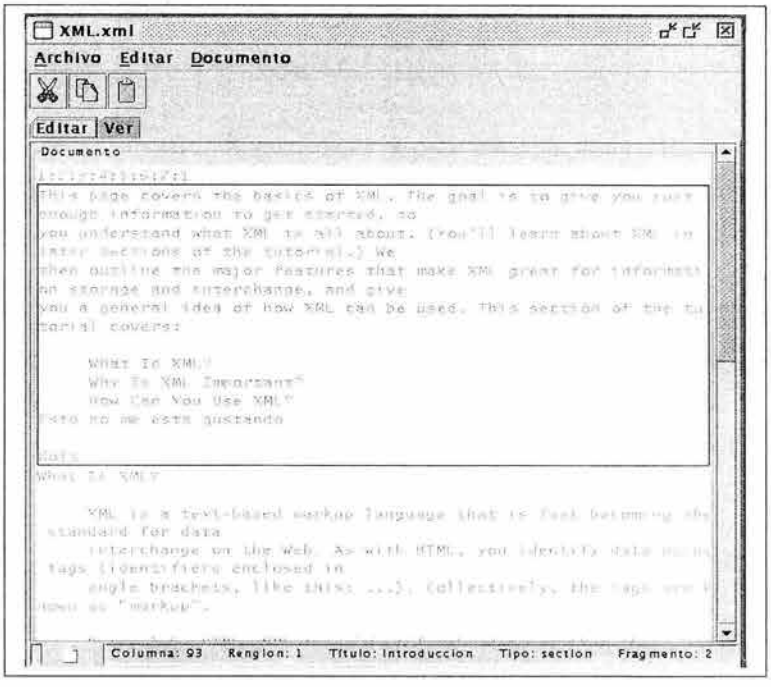

Figura 4: Ventana de edición

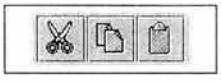

Figura 5: Barra de herramientas de la ventana de edición

- Menús de la ventana de edición.

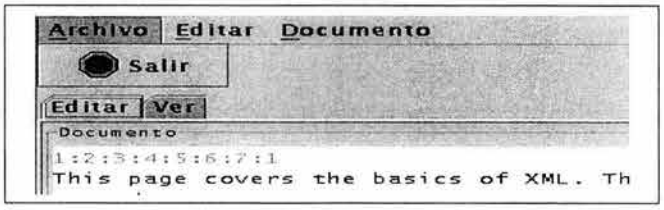

Figura 6: Menú Archivo

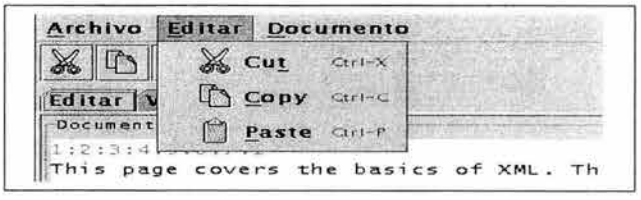

Figura 7: Menú Editar

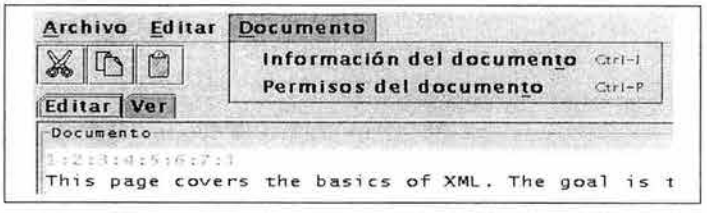

Figura 8: Menú Documento

- Pestaña de visualización.

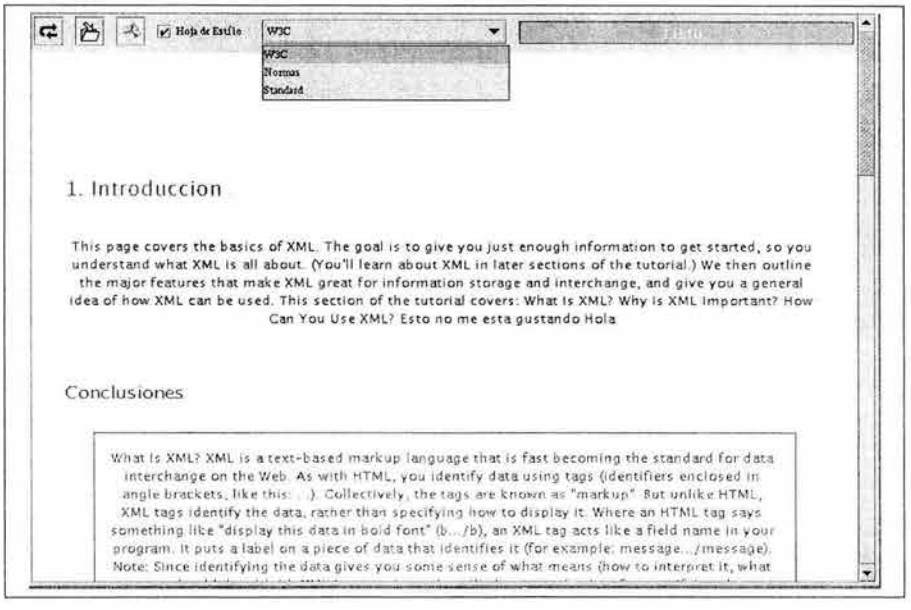

Figura 9: Pestaña de visualización

- Pestaña de edición.

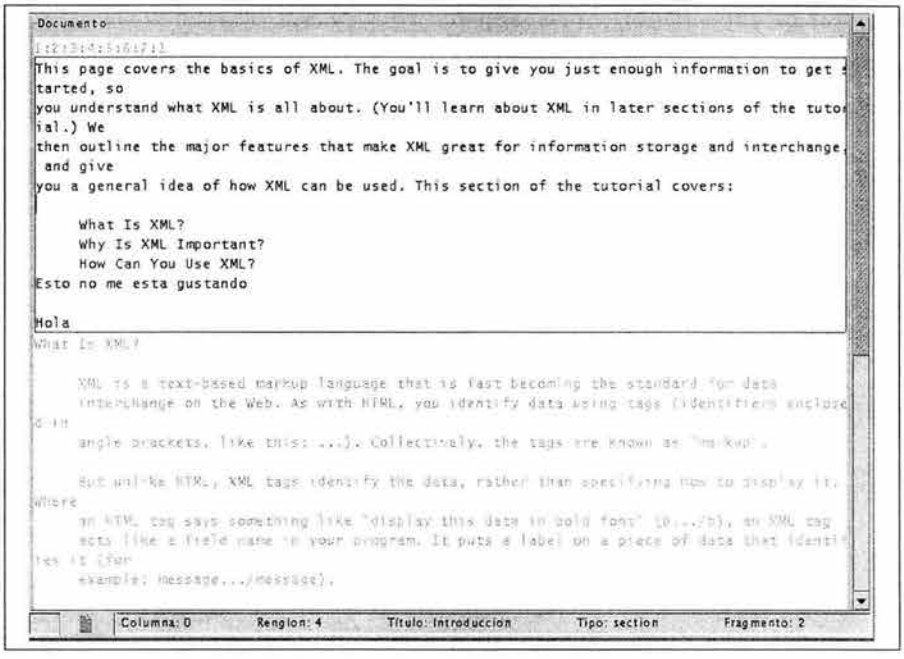

Figura 10: Pestaña de edición

- Menús de documento.

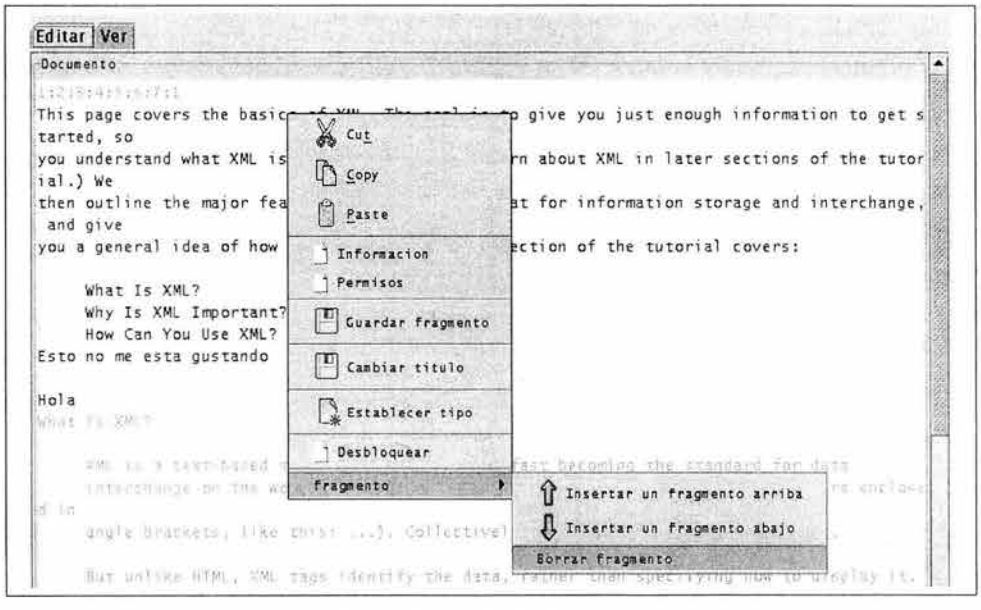

Figura 11: Menús de documento

- Menús del editor.

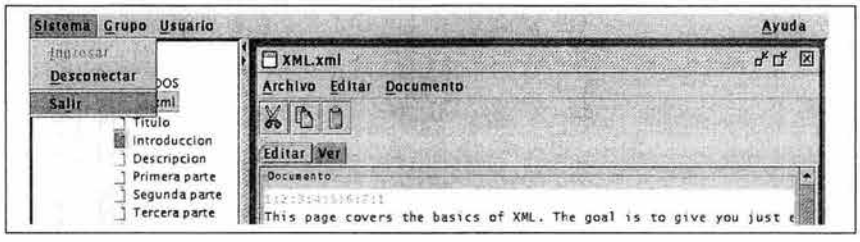

Figura 12: Menú Sistema

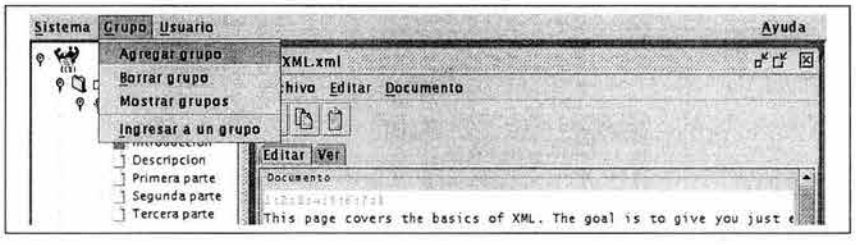

Figura 13: Menú Grupo

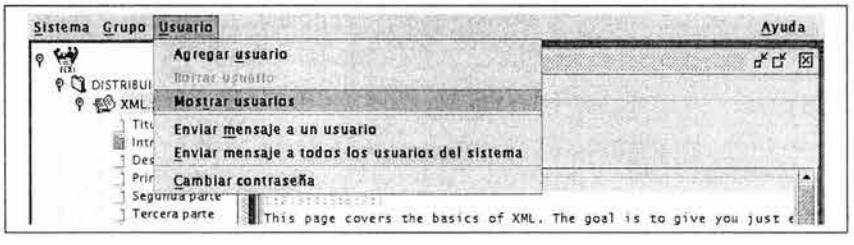

Figura 14: Menú Usuario

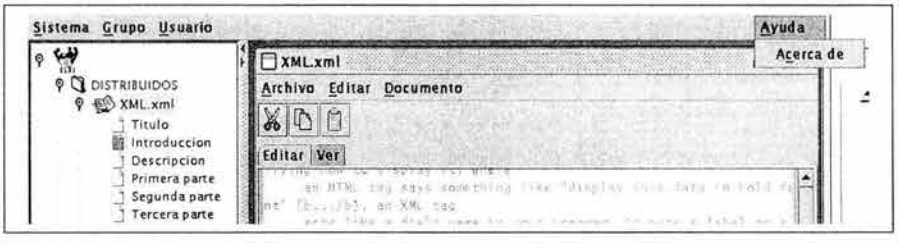

Figura 15: Menú Ayuda

- Menús en el árbol de navegación.

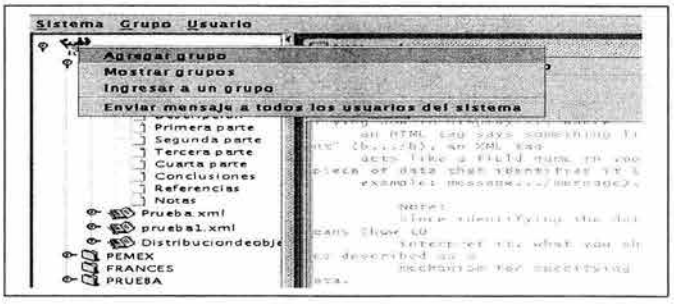

Figura 16: Menú Nivel Raíz

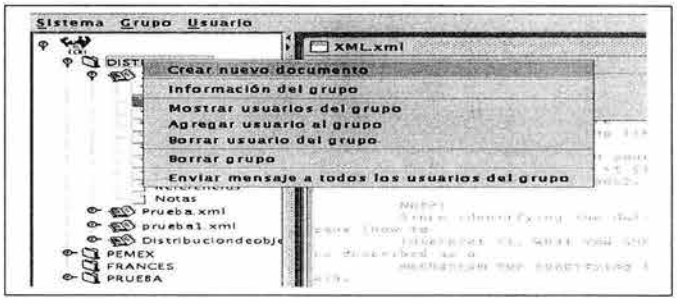

Figura 17: Menú Nivel Grupos

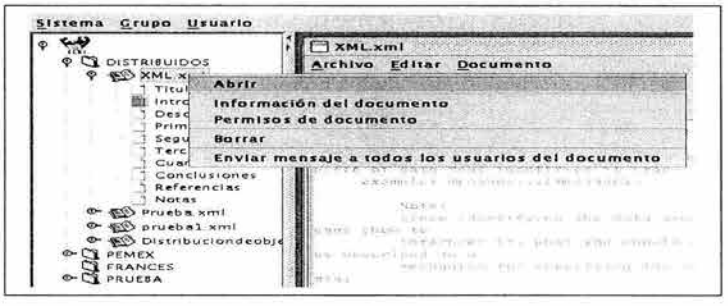

Figura 18: Menú Nivel Documentos

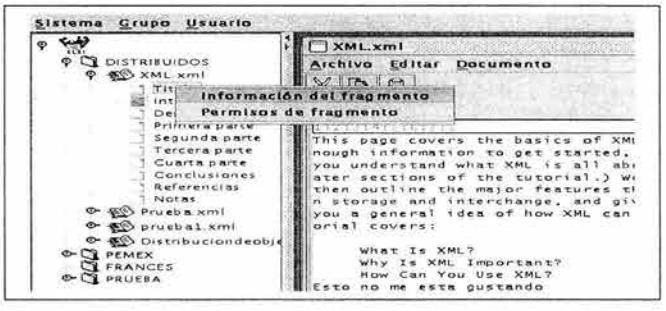

Figura 19: Menú Nivel Fragmentos

Apéndice B

Plantillas de Estilo XSLT

B.1. La plantilla *standard.xsl*

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<!DOCTYPE xsl:stylesheet [<!ENTITY nbsp " ">]>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" encoding="iso-8859-1" doctype-public="//w3c//dtd html 4.0 transitional//en"/>
<xsl:param name="nombre_archivo" select="'Sin título'"/>
<xsl:param name="css" select="''"/>
<xsl:template match="documento">
<html>
    <head>
        <title>
            <xsl:text></xsl:text><xsl:value-of select="$nombre_archivo"/>
        </title>
        <style type="text/css">
          <xsl:value-of select="$css"/>
        </style>
    </head>
<body>
<div style="text-align: center">
<xsl:text></xsl:text>
<xsl:text></xsl:text>
<xsl:apply-templates select="fragmento[@tipo='title']"/>
<xsl:apply-templates select="fragmento[@tipo='subtitle']"/>
<xsl:apply-templates select="fragmento[@tipo='authors']"/>
<xsl:apply-templates select="fragmento[@tipo='date']"/>
</div>
<div class="head">
<xsl:apply-templates select="fragmento[@tipo='abstract']"/>
<xsl:apply-templates select="fragmento[@tipo='keywords']"/>
<br/>
<xsl:apply-templates select="fragmento[@tipo='section']|fragmento[@tipo='subsection']"/>
<br/>
```

Continua en la página siguiente

Continuación de la página anterior

```
<xsl:apply-templates select="fragmento[@tipo='result']"/>
<xsl:apply-templates select="fragmento[@tipo='conclu']"/>
<xsl:apply-templates select="fragmento[@tipo='agrade']"/>
<xsl:apply-templates select="fragmento[@tipo='referen']"/>
</div>
  <xsl:text>
  </xsl:text>
</body>
</html>
</xsl:template>
  <!-- *** Titulo *** -->
<xsl:template match="fragmento[@tipo='title']">
<br/>
    <h1>
        <xsl:apply-templates/>
    </h1>
</xsl:template>
<xsl:template match="fragmento[@tipo='subtitle']">
<br/>
<h2>
        <xsl:apply-templates/>
</h2>
</xsl:template>
  <!-- *** Autores ** -->
<xsl:template match="fragmento[@tipo='authors']">
<br/>
<h5>
            <xsl:apply-templates/>
</h5>
</xsl:template>
  <!-- *** Fecha ** -->
<xsl:template match="fragmento[@tipo='date']">
<h5>
 Fecha: <xsl:apply-templates/>
</h5>
</xsl:template>
  <!-- ** Resumen ** -->
<xsl:template match="fragmento[@tipo='abstract']">
<br/>
        <h3>Resumen</h3>
        <h4>
            <xsl:apply-templates/>
        </h4>
</xsl:template>
  <!-- ** Keywords ** -->
<xsl:template match="fragmento[@tipo='keywords']">
<br/>
        <h3>Palabras clave:</h3>
        <h4>
            <xsl:apply-templates/>
```

Continua en la página siguiente

Continuación de la página anterior

```
            </h4>
        </xsl:template>
        <!– Section && Subsection–>
        <xsl:template match="fragmento[@tipo='section']|fragmento[@tipo='subsection']">
            <div>
                <xsl:if test="@tipo='section'">
                <xsl:for-each select=".">
                    <br/>
                    <h2>
                        <xsl:number count="fragmento[@tipo='section']" format = "1. "/>
                        <xsl:apply-templates select="@titulo"/>
                    </h2>
                    <br/>
                    <p align="justify">
                        <xsl:apply-templates select="text()"/></p>
                </xsl:for-each>
                </xsl:if>
                <xsl:if test="@tipo='subsection'">
                <xsl:for-each select=".">
                  <br/>
                    <h3>
                        <xsl:apply-templates select="@titulo"/>
                    </h3>
                    <br/>
                    <p align="justify">
                        <xsl:apply-templates select="text()"/></p>
                </xsl:for-each>
                </xsl:if>
            </div>
        </xsl:template>
        <xsl:template match="fragmento[@tipo='result']">
        <br/>
                <h3>Resultados</h3>
                <p align="justify">
                    <xsl:apply-templates/>
                </p>
        </xsl:template>
        <xsl:template match="fragmento[@tipo='conclu']">
        <br/>
                <h3>Conclusiones</h3>
                <p align="justify">
                    <xsl:apply-templates/>
                </p>
        </xsl:template>
        <xsl:template match="fragmento[@tipo='agrade']">
        <br/>
                <h3>Agradecimientos</h3>
                <p align="justify">
                    <xsl:apply-templates/>
                </p>
```

Continua en la página siguiente

Continuación de la página anterior

```
</xsl:template>
<xsl:template match="fragmento[@tipo='referen']">
<br/>
        <h3>Referencias bibliográficas</h3>
        <p align="justify">
            <xsl:apply-templates/>
        </p>
</xsl:template>
</xsl:stylesheet >
```

B.1: La plantilla *standard.xsl*

B.2. La plantilla *w3c.xsl*

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<!DOCTYPE xsl:stylesheet [<!ENTITY nbsp " ">]>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" encoding="iso-8859-1" doctype-public="//w3c//dtd html 4.0 transitional//en"/>
<xsl:param name="nombre_archivo" select="'Sin título'"/>
<xsl:param name="css" select="''"/>
<xsl:template match="documento">
<html>
    <head>
        <title>
            <xsl:text></xsl:text><xsl:value-of select="$nombre_archivo"/>
        </title>
        <style type="text/css">
          <xsl:value-of select="$css"/>
        </style>
    </head>
<body class="articleStyle">
<div style="text-align: center">
<xsl:text></xsl:text>
<xsl:text></xsl:text>
<xsl:apply-templates select="fragmento[@tipo='title']"/>
<xsl:apply-templates select="fragmento[@tipo='subtitle']"/>
<xsl:apply-templates select="fragmento[@tipo='authors']"/>
<xsl:apply-templates select="fragmento[@tipo='date']"/>
</div>
<div class="head">
<xsl:apply-templates select="fragmento[@tipo='abstract']"/>
<xsl:apply-templates select="fragmento[@tipo='keywords']"/>
<br/>
<xsl:apply-templates select="fragmento[@tipo='section']|fragmento[@tipo='subsection']"/>
<br/>
<xsl:apply-templates select="fragmento[@tipo='result']"/>
<xsl:apply-templates select="fragmento[@tipo='conclu']"/>
<xsl:apply-templates select="fragmento[@tipo='agrade']"/>
```

Continua en la página siguiente

Continuación de la página anterior

```
<xsl:apply-templates select="fragmento[@tipo='referen']"/>
</div>
  <xsl:text>
  </xsl:text>
</body>
</html>
</xsl:template>
  <!– *** Titulo *** –>
<xsl:template match="fragmento[@tipo='title']">
<br/>
    <h1>
        <xsl:apply-templates/>
    </h1>
</xsl:template>
<xsl:template match="fragmento[@tipo='subtitle']">
<br/>
<h2>
        <xsl:apply-templates/>
</h2>
</xsl:template>
  <!– *** Autores ** –>
<xsl:template match="fragmento[@tipo='authors']">
<br/>
<h5 class="authorsStyle">
            <xsl:apply-templates/>
</h5>
</xsl:template>
  <!– *** Fecha ** –>
<xsl:template match="fragmento[@tipo='date']">
<br/>
  <h3>Fecha:</h3>
<h5 class="authorsStyle">
          <xsl:apply-templates/>
</h5>
</xsl:template>
  <!– ** Resumen ** –>
<xsl:template match="fragmento[@tipo='abstract']">
<br/>
        <h3>Resumen</h3>
        <h4 class="abstractStyle">
            <xsl:apply-templates/>
        </h4>
</xsl:template>
  <!– ** Keywords ** –>
<xsl:template match="fragmento[@tipo='keywords']">
<br/>
        <h3>Palabras clave:</h3>
        <h4 class="keyStyle">
            <xsl:apply-templates/>
        </h4>
```

Continua en la página siguiente

Continuación de la página anterior

```
</xsl:template>
<!– Section && Subsection–>
<xsl:template match="fragmento[@tipo='section']|fragmento[@tipo='subsection']">
    <div>
        <xsl:if test="@tipo='section'">
        <xsl:for-each select=".">
            <br/>
            <h2 class="sectionStyle">
                <xsl:number count="fragmento[@tipo='section']" format = "1. "/>
                <xsl:apply-templates select="@titulo"/>
            </h2>
            <br/>
            <p align="justify">
                <xsl:apply-templates select="text()"/></p>
        </xsl:for-each>
        </xsl:if>
        <xsl:if test="@tipo='subsection'">
        <xsl:for-each select=".">
          <br/>
            <h3 class="subsectionStyle">
                <xsl:apply-templates select="@titulo"/>
            </h3>
            <br/>
            <p align="justify">
                <xsl:apply-templates select="text()"/></p>
        </xsl:for-each>
        </xsl:if>
    </div>
</xsl:template>
<xsl:template match="fragmento[@tipo='result']">
<br/>
        <h3>Resultados</h3>
        <p align="justify">
            <xsl:apply-templates/>
        </p>
</xsl:template>
<xsl:template match="fragmento[@tipo='conclu']">
<br/>
        <h3>Conclusiones</h3>
        <p align="justify" class="concluStyle">
            <xsl:apply-templates/>
        </p>
</xsl:template>
<xsl:template match="fragmento[@tipo='agrade']">
<br/>
        <h3>Agradecimientos</h3>
        <p align="justify" class="agradeStyle">
            <xsl:apply-templates/>
        </p>
</xsl:template>
```

Continua en la página siguiente

Continuación de la página anterior

```
        <xsl:template match="fragmento[@tipo='referen']">
        <br/>
                <h3>Referencias bibliográficas</h3>
                <p align="justify">
                    <xsl:apply-templates/>
                </p>
        </xsl:template>
Aqui me quede
        <xsl:template match="notes">
            <div id="Notes">
                <h3 class="noteHeadStyle">Notes</h3>
                <xsl:for-each select="note" >
                        <xsl:text></xsl:text>
                        <xsl:choose>
                            <xsl:when test="p/bib[@published = 'no']">
                                <a name="@id"></a>
                                <p class="noteStyle" id="@id">
                                    <b>
                                        [<xsl:value-of select="@num" />]
                                        <xsl:text></xsl:text>
                                    </b>
                                    <xsl:apply-templates select=".//bib.tagged"/>
                                        <xsl:if test=".//bib.link">
                                            <a href=".//bib.link/@href">
                                            <xsl:apply-templates select=".//bib.link"/></a>
                                        </xsl:if>
                                </p>
                            </xsl:when>
                            <xsl:when test="p/bib">
                                <a name="@id"></a>
                                <p class="noteStyle" id="@id">
                                    <b>
                                        [<xsl:value-of select="@num" />]
                                        <xsl:text></xsl:text>
                                    </b>
                                    <xsl:for-each select="p">
                                        <xsl:apply-templates select=".//au"/>
                                        <xsl:text>, </xsl:text>
                                        <i>
                                            <xsl:apply-templates select=".//jn"/>
                                        </i>
                                        <xsl:text></xsl:text>
                                        <b>
                                            <xsl:apply-templates select=".//vol"/>
                                        </b>
                                        <xsl:text></xsl:text>
                                        <xsl:apply-templates select=".//pp"/>
                                        <xsl:text></xsl:text>
                                        (<xsl:apply-templates select=".//yr"/>)
                                        <xsl:text></xsl:text>
```

Continua en la página siguiente

Continuación de la página anterior

```
                                <xsl:if test=".//bib.link">
                                    <a href=".//bib.link/@href">
                                        <xsl:apply-templates select=".//bib.link"/>
                                    </a>
                                </xsl:if>
                            </xsl:for-each>
                        </p>
                    </xsl:when>
                    <xsl:otherwise>
                        <a name="@id"></a>
                            <p class="noteStyle" id="@id">
                                <b>
                                    [<xsl:value-of select="@num" />]
                                    <xsl:text></xsl:text>
                                </b>
                                <xsl:apply-templates/>
                            </p>
                    </xsl:otherwise>
                </xsl:choose>
            </xsl:for-each>
    </div>
</xsl:template>
<xsl:template match="tr">
    <tr class="tableStyle" align="@align" valign="@valign" colspan="@colspan" rowspan="rowspan">
        <xsl:apply-templates/>
    </tr>
</xsl:template>
<xsl:template match="th">
    <th class="tableHeadStyle" align="@align" valign="@valign" colspan="@colspan" rowspan="rowspan">
        <xsl:apply-templates/>
    </th>
</xsl:template>
<xsl:template match="td">
    <td class="tableStyle" align="@align" valign="@valign" colspan="@colspan" rowspan="rowspan">
        <p class="tableStyle">
            <xsl:apply-templates/>
        </p>
    </td>
</xsl:template>
<xsl:template match="tn">
    <tn>
        <xsl:apply-templates/>
    </tn>
    <br/>
</xsl:template>-->
</xsl:stylesheet >
```

B.2: La plantilla *w3c.xsl*

Apéndice C

Plantillas de Estilo XSL-FO

C.1. La plantilla *standard.fo*

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                                                    xmlns:fo="http://www.w3.org/1999/XSL/Format">
<xsl:param name="nombre_archivo" select="'Sin título'"/>
<xsl:template match="documento">
<fo:root>
    <fo:layout-master-set>
      <!– layout for the first page –>
        <fo:simple-page-master master-name="first"
                        page-height="29.7cm"
                        page-width="21cm"
                        margin-top="1cm"
                        margin-bottom="2cm"
                        margin-left="2.5cm"
                        margin-right="2.5cm">
            <fo:region-body margin-top="3cm"/>
            <fo:region-before extent="3cm"/>
            <fo:region-after extent="1.5cm"/>
        </fo:simple-page-master>
        <!– layout for the other pages –>
    <fo:simple-page-master master-name="rest"
                                    page-height="29.7cm"
                                    page-width="21cm"
                                    margin-top="1cm"
                                    margin-bottom="2cm"
                                    margin-left="2.5cm"
                                    margin-right="2.5cm">
        <fo:region-body margin-top="2.5cm"/>
        <fo:region-before extent="2.5cm"/>
        <fo:region-after extent="1.5cm"/>
      </fo:simple-page-master>
```

Continua en la página siguiente

Continuación de la página anterior

```
            <fo:page-sequence-master master-name="basicPSM" >
                        <fo:repeatable-page-master-alternatives>
                                    <fo:conditional-page-master-reference master-reference="first"
                                        page-position="first" />
                                    <fo:conditional-page-master-reference master-reference="rest"
                                        page-position="rest" />
                                    <!– recommended fallback procedure –>
                                    <fo:conditional-page-master-reference master-reference="rest" />
                        </fo:repeatable-page-master-alternatives>
            </fo:page-sequence-master>
    </fo:layout-master-set>
    <fo:page-sequence master-reference="basicPSM">
        <fo:static-content flow-name="xsl-region-before">
            <fo:block text-align="end" font-size="10pt" font-family="serif" line-height="14pt" color="red" >
            <xsl:value-of select="$nombre_archivo"/>- Página <fo:page-number/>
            </fo:block>
        </fo:static-content>
            <fo:flow flow-name="xsl-region-body" font-family="Times Roman" font-size="9pt">
              <fo:block>
                <xsl:apply-templates select="fragmento[@tipo='title']"/>
                <xsl:apply-templates select="fragmento[@tipo='subtitle']"/>
                <xsl:apply-templates select="fragmento[@tipo='authors']"/>
                <xsl:apply-templates select="fragmento[@tipo='date']"/>
                <xsl:apply-templates select="fragmento[@tipo='abstract']"/>
                <xsl:apply-templates select="fragmento[@tipo='keywords']"/>
                <xsl:apply-templates select="fragmento[@tipo='section']|fragmento[@tipo='subsection']"/>
                <xsl:apply-templates select="fragmento[@tipo='result']"/>
                <xsl:apply-templates select="fragmento[@tipo='conclu']"/>
                <xsl:apply-templates select="fragmento[@tipo='agrade']"/>
                <xsl:apply-templates select="fragmento[@tipo='referen']"/>
              </fo:block>
            </fo:flow>
    </fo:page-sequence>
       </fo:root>
    </xsl:template>
    <xsl:template match="fragmento[@tipo='title']">
        <fo:block font-size="24pt"
font-family="sans-serif"
line-height="24pt"
space-after.optimum="15pt"
background-color="white"
text-align="center"
padding-top="3pt"
font-variant="small-caps">
            <xsl:value-of select="."/>
        </fo:block>
    </xsl:template>
    <!– defines text title level 2–>
    <xsl:template match="fragmento[@tipo='subtitle']">
        <fo:block font-size="16pt"
```

Continua en la página siguiente

Continuación de la página anterior

```
font-family="sans-serif"
line-height="20pt"
space-before.optimum="10pt"
space-after.optimum="10pt"
background-color="white"
text-align="center"
padding-top="3pt">
                <xsl:value-of select="."/>
            </fo:block>
        </xsl:template>
        <xsl:template match="fragmento[@tipo='authors']">
            <fo:block font-size="9pt"
font-family="sans-serif"
line-height="15pt"
space-after.optimum="3pt"
text-align="center">
            <xsl:value-of select="."/>
            </fo:block>
        </xsl:template>
          <!– *** Fecha ** –>
        <xsl:template match="fragmento[@tipo='date']">
        <fo:block font-size="9pt"
font-family="sans-serif"
line-height="15pt"
space-after.optimum="3pt"
text-align="center">
Fecha: <xsl:value-of select="."/>
            </fo:block>
        </xsl:template>
          <!– ** Keywords ** –>
        <xsl:template match="fragmento[@tipo='keywords']">
            <fo:block font-size="12pt"
font-family="sans-serif"
line-height="10pt"
space-before.optimum="10pt"
space-after.optimum="10pt"
text-align="left"
background-color="white"
padding-top="20pt">
Palabras clave:
            </fo:block>
            <fo:block font-size="10pt"
font-family="sans-serif"
line-height="10pt"
space-before.optimum="10pt"
space-after.optimum="10pt"
text-align="center"
padding-top="1pt">
                <xsl:value-of select="."/>
            </fo:block>
```

Continua en la página siguiente

Continuación de la página anterior

```
        </xsl:template>
        <xsl:template match="fragmento[@tipo='abstract']">
        <fo:block padding-top="30pt"
                text-indent="1em"
                text-align="left"
                font-family="sans-serif"
                font-size="12pt"
                font-weight="bold"
                line-height="8mm">
Resumen
        </fo:block>
        <fo:block padding-top="5pt"
                text-indent="1em"
                text-align="justify"
                font-family="sans-serif"
                font-size="10pt"
                line-height="8mm">
            <xsl:value-of select="."/>
        </fo:block>
        </xsl:template>
        <!– Section && Subsection–>
        <xsl:template match="fragmento[@tipo='section']|fragmento[@tipo='subsection']">
                <xsl:if test="@tipo='section'">
                <xsl:for-each select=".">
                        <fo:block font-size="14pt"
                                font-family="sans-serif"
                                line-height="20pt"
                                space-before.optimum="10pt"
                                space-after.optimum="10pt"
                                background-color="white"
                                text-align="center">
                        <xsl:number count="fragmento[@tipo='section']" format = "1. "/>
                        <xsl:apply-templates select="@titulo"/>
                        </fo:block>
                        <fo:block font-size="10pt"
                                font-family="sans-serif"
                                space-before.optimum="10pt"
                                space-after.optimum="10pt"
                                text-align="justify"
                                line-height="8mm"
                                padding-top="1pt">
                          <xsl:apply-templates select="text()"/>
                        </fo:block>
                </xsl:for-each>
                </xsl:if>
                <xsl:if test="@tipo='subsection'">
                <xsl:for-each select=".">
                        <fo:block font-size="14pt"
                                font-family="sans-serif"
                                line-height="20pt"
```

Continua en la página siguiente

Continuación de la página anterior

```
                        start-indent="30pt"
                        space-before.optimum="10pt"
                        space-after.optimum="10pt"
                        background-color="white"
                        text-align="left">
                    <xsl:apply-templates select="@titulo"/>
                </fo:block>
                <fo:block font-size="10pt"
                        font-family="sans-serif"
                        space-before.optimum="10pt"
                        space-after.optimum="10pt"
                        text-align="justify"
                        line-height="8mm"
                        padding-top="1pt">
                    <xsl:apply-templates select="text()"/>
                </fo:block>
            </xsl:for-each>
            </xsl:if>
    </xsl:template>
    <xsl:template match="fragmento[@tipo='result']">
        <fo:block font-size="12pt"
font-family="sans-serif"
line-height="10pt"
space-before.optimum="10pt"
space-after.optimum="10pt"
text-align="left"
background-color="white"
padding-top="20pt">
Resultados
        </fo:block>
        <fo:block font-size="10pt"
                        font-family="sans-serif"
                        space-before.optimum="10pt"
                        space-after.optimum="10pt"
                        text-align="justify"
                        line-height="8mm"
                        padding-top="1pt">
                <xsl:value-of select="."/>
            </fo:block>
    </xsl:template>
    <xsl:template match="fragmento[@tipo='conclu']">
    <fo:block font-size="12pt"
font-family="sans-serif"
line-height="10pt"
space-before.optimum="10pt"
space-after.optimum="10pt"
text-align="left"
background-color="white"
padding-top="20pt">
Conclusiones
```

Continua en la página siguiente

Continuación de la página anterior

```
        </fo:block>
    <fo:block padding-top="5pt"
            text-indent="1em"
            text-align="justify"
            font-family="sans-serif"
            font-size="10pt"
            line-height="8mm"
            width="4in">
        <xsl:value-of select="."/>
    </fo:block>
    </xsl:template>
    <xsl:template match="fragmento[@tipo='agrade']">
    <fo:block font-size="12pt"
font-family="sans-serif"
line-height="10pt"
space-before.optimum="10pt"
space-after.optimum="10pt"
text-align="left"
background-color="white"
padding-top="20pt">
Agradecimientos
        </fo:block>
          <fo:block font-size="10pt"
                        font-family="sans-serif"
                        space-before.optimum="10pt"
                        space-after.optimum="10pt"
                        text-align="justify"
                        line-height="8mm"
                        padding-top="1pt">
              <xsl:value-of select="."/>
          </fo:block>
    </xsl:template>
    <xsl:template match="fragmento[@tipo='referen']">
    <fo:block font-size="12pt"
font-family="sans-serif"
line-height="10pt"
space-before.optimum="10pt"
space-after.optimum="10pt"
text-align="left"
background-color="white"
padding-top="20pt">
Referencias bibliográficas
        </fo:block>
    <fo:block font-size="10pt"
                        font-family="sans-serif"
                        space-before.optimum="10pt"
                        space-after.optimum="10pt"
                        text-align="justify"
                        line-height="8mm"
                        padding-top="1pt">
```

Continua en la página siguiente

Continuación de la página anterior

```
            <xsl:value-of select="."/>
        </fo:block>
</xsl:template>
</xsl:stylesheet >
```

C.1: La plantilla *standard.fo*

C.2. La plantilla *w3c.fo*

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                                                    xmlns:fo="http://www.w3.org/1999/XSL/Format">
<xsl:param name="nombre_archivo" select="'Sin título'"/>
<xsl:template match="documento">
<fo:root>
    <fo:layout-master-set>
      <!– layout for the first page –>
        <fo:simple-page-master master-name="first"
                        page-height="29.7cm"
                        page-width="21cm"
                        margin-top="1cm"
                        margin-bottom="2cm"
                        margin-left="2.5cm"
                        margin-right="2.5cm">
            <fo:region-body margin-top="3cm"/>
            <fo:region-before extent="3cm"/>
            <fo:region-after extent="1.5cm"/>
        </fo:simple-page-master>
        <!– layout for the other pages –>
    <fo:simple-page-master master-name="rest"
                                    page-height="29.7cm"
                                    page-width="21cm"
                                    margin-top="1cm"
                                    margin-bottom="2cm"
                                    margin-left="2.5cm"
                                    margin-right="2.5cm">
        <fo:region-body margin-top="2.5cm"/>
        <fo:region-before extent="2.5cm"/>
        <fo:region-after extent="1.5cm"/>
      </fo:simple-page-master>
        <fo:page-sequence-master master-name="basicPSM" >
                    <fo:repeatable-page-master-alternatives>
                                    <fo:conditional-page-master-reference master-reference="first"
                                        page-position="first" />
                                    <fo:conditional-page-master-reference master-reference="rest"
                                        page-position="rest" />
                                    <!– recommended fallback procedure –>
                                    <fo:conditional-page-master-reference master-reference="rest" />
```

Continua en la página siguiente

Continuación de la página anterior

```
                    </fo:repeatable-page-master-alternatives>
            </fo:page-sequence-master>
    </fo:layout-master-set>
    <fo:page-sequence master-reference="basicPSM">
        <fo:static-content flow-name="xsl-region-before">
            <fo:block text-align="end" font-size="10pt" font-family="serif" line-height="14pt" color="red" >
            <xsl:value-of select="$nombre_archivo"/>- Página <fo:page-number/>
            </fo:block>
        </fo:static-content>
            <fo:flow flow-name="xsl-region-body" font-family="Times Roman" font-size="9pt">
              <fo:block>
                <xsl:apply-templates select="fragmento[@tipo='title']"/>
                <xsl:apply-templates select="fragmento[@tipo='subtitle']"/>
                <xsl:apply-templates select="fragmento[@tipo='authors']"/>
                <xsl:apply-templates select="fragmento[@tipo='date']"/>
                <xsl:apply-templates select="fragmento[@tipo='abstract']"/>
                <xsl:apply-templates select="fragmento[@tipo='keywords']"/>
                <xsl:apply-templates select="fragmento[@tipo='section']|fragmento[@tipo='subsection']"/>
                <xsl:apply-templates select="fragmento[@tipo='result']"/>
                <xsl:apply-templates select="fragmento[@tipo='conclu']"/>
                <xsl:apply-templates select="fragmento[@tipo='agrade']"/>
                <xsl:apply-templates select="fragmento[@tipo='referen']"/>
              </fo:block>
            </fo:flow>
    </fo:page-sequence>
      </fo:root>
    </xsl:template>
    <xsl:template match="fragmento[@tipo='title']">
        <fo:block font-size="24pt"
font-family="sans-serif"
line-height="24pt"
space-after.optimum="15pt"
color="#005A9C"
background-color="white"
text-align="center"
padding-top="3pt"
font-variant="small-caps">
            <xsl:value-of select="."/>
        </fo:block>
    </xsl:template>
    <!- defines text title level 2-->
    <xsl:template match="fragmento[@tipo='subtitle']">
        <fo:block font-size="16pt"
font-family="sans-serif"
line-height="20pt"
space-before.optimum="10pt"
space-after.optimum="10pt"
color="#005A9C"
background-color="white"
text-align="center"
```

Continua en la página siguiente

Continuación de la página anterior

```
padding-top="3pt">
                <xsl:value-of select="."/>
            </fo:block>
        </xsl:template>
        <xsl:template match="fragmento[@tipo='authors']">
            <fo:block font-size="9pt"
font-family="sans-serif"
line-height="15pt"
space-after.optimum="3pt"
text-align="center">
            <xsl:value-of select="."/>
            </fo:block>
        </xsl:template>
          <!-- *** Fecha ** -->
        <xsl:template match="fragmento[@tipo='date']">
        <fo:block font-size="9pt"
font-family="sans-serif"
line-height="15pt"
space-after.optimum="3pt"
text-align="center">
Fecha: <xsl:value-of select="."/>
            </fo:block>
        </xsl:template>
          <!-- ** Keywords ** -->
        <xsl:template match="fragmento[@tipo='keywords']">
            <fo:block font-size="12pt"
font-family="sans-serif"
line-height="10pt"
space-before.optimum="10pt"
space-after.optimum="10pt"
text-align="left"
color="#005A9C"
background-color="white"
padding-top="20pt">
Palabras clave:
            </fo:block>
            <fo:block font-size="10pt"
font-family="sans-serif"
line-height="10pt"
space-before.optimum="10pt"
space-after.optimum="10pt"
text-align="center"
padding-top="1pt">
                <xsl:value-of select="."/>
            </fo:block>
        </xsl:template>
        <xsl:template match="fragmento[@tipo='abstract']">
        <fo:block padding-top="30pt"
                text-indent="1em"
                text-align="left"
```

Continua en la página siguiente

Continuación de la página anterior

```
                font-family="sans-serif"
                font-size="12pt"
                color="#005A9C"
                font-weight="bold"
                line-height="8mm">
Resumen
        </fo:block>
        <fo:block padding-top="5pt"
                text-indent="1em"
                text-align="justify"
                font-family="sans-serif"
                font-size="10pt"
                background-color="#EEEEEE"
                line-height="8mm"
                border-after-color="blue"
                border-after-style="outset"
                border-after-width=".1mm"
                border-before-color="blue"
                border-before-style="outset"
                border-before-width=".1mm"
                border-end-color="blue"
                border-end-style="outset"
                border-end-width=".1mm"
                border-start-color="blue"
                border-start-style="outset"
                border-start-width=".1mm">
            <xsl:value-of select="."/>
        </fo:block>
        </xsl:template>
        <!– Section && Subsection–>
        <xsl:template match="fragmento[@tipo='section']|fragmento[@tipo='subsection']">
                <xsl:if test="@tipo='section'">
                <xsl:for-each select=".">
                        <fo:block font-size="14pt"
                                font-family="sans-serif"
                                line-height="20pt"
                                space-before.optimum="10pt"
                                space-after.optimum="10pt"
                                color="#005A9C"
                                background-color="white"
                                text-align="left">
                        <xsl:number count="fragmento[@tipo='section']" format = "1. "/>
                        <xsl:apply-templates select="@titulo"/>
                        </fo:block>
                        <fo:block font-size="10pt"
                                font-family="sans-serif"
                                space-before.optimum="10pt"
                                space-after.optimum="10pt"
                                text-align="justify"
                                line-height="8mm"
```

Continua en la página siguiente

Continuación de la página anterior

```
                        padding-top="1pt">
                    <xsl:apply-templates select="text()"/>
                </fo:block>
            </xsl:for-each>
            </xsl:if>
            <xsl:if test="@tipo='subsection'">
            <xsl:for-each select=".">
                <fo:block font-size="14pt"
                        font-family="sans-serif"
                        line-height="20pt"
                        start-indent="30pt"
                        space-before.optimum="10pt"
                        space-after.optimum="10pt"
                        color="#005A9C"
                        background-color="white"
                        text-align="left">
                    <xsl:apply-templates select="@titulo"/>
                </fo:block>
                <fo:block font-size="10pt"
                        font-family="sans-serif"
                        space-before.optimum="10pt"
                        space-after.optimum="10pt"
                        text-align="justify"
                        line-height="8mm"
                        padding-top="1pt">
                    <xsl:apply-templates select="text()"/>
                </fo:block>
            </xsl:for-each>
            </xsl:if>
    </xsl:template>
    <xsl:template match="fragmento[@tipo='result']">
        <fo:block font-size="12pt"
font-family="sans-serif"
line-height="10pt"
space-before.optimum="10pt"
space-after.optimum="10pt"
text-align="left"
color="#005A9C"
background-color="white"
padding-top="20pt">
Resultados
        </fo:block>
        <fo:block font-size="10pt"
                        font-family="sans-serif"
                        space-before.optimum="10pt"
                        space-after.optimum="10pt"
                        text-align="justify"
                        line-height="8mm"
                        padding-top="1pt">
                <xsl:value-of select="."/>
```

Continua en la página siguiente

Continuación de la página anterior

```
                </fo:block>
    </xsl:template>
    <xsl:template match="fragmento[@tipo='conclu']">
    <fo:block font-size="12pt"
font-family="sans-serif"
line-height="10pt"
space-before.optimum="10pt"
space-after.optimum="10pt"
text-align="left"
color="#005A9C"
background-color="white"
padding-top="20pt">
Conclusiones
        </fo:block>
    <fo:block border-after-color="red"
              border-after-style="outset"
              border-after-width=".1mm"
              border-before-color="red"
              border-before-style="outset"
              border-before-width=".1mm"
              border-end-color="red"
              border-end-style="outset"
              border-end-width=".1mm"
              border-start-color="red"
              border-start-style="outset"
              border-start-width=".1mm"
              padding-top="5pt"
              text-indent="1em"
              text-align="justify"
              font-family="sans-serif"
              font-size="10pt"
              background-color="#EEEEEE"
              line-height="8mm"
              width="4in">
        <xsl:value-of select="."/>
    </fo:block>
    </xsl:template>
    <xsl:template match="fragmento[@tipo='agrade']">
    <fo:block font-size="12pt"
font-family="sans-serif"
line-height="10pt"
space-before.optimum="10pt"
space-after.optimum="10pt"
text-align="left"
color="#005A9C"
background-color="white"
padding-top="20pt">
Agradecimientos
        </fo:block>
          <fo:block font-size="10pt"
```

Continua en la página siguiente

Continuación de la página anterior

```
                                font-family="sans-serif"
                                space-before.optimum="10pt"
                                space-after.optimum="10pt"
                                text-align="justify"
                                line-height="8mm"
                                padding-top="1pt">
                        <xsl:value-of select="."/>
                    </fo:block>
        </xsl:template>
        <xsl:template match="fragmento[@tipo='referen']">
        <fo:block font-size="12pt"
    font-family="sans-serif"
    line-height="10pt"
    space-before.optimum="10pt"
    space-after.optimum="10pt"
    text-align="left"
    color="#005A9C"
    background-color="white"
    padding-top="20pt">
    Referencias bibliográficas
            </fo:block>
        <fo:block font-size="10pt"
                                font-family="sans-serif"
                                space-before.optimum="10pt"
                                space-after.optimum="10pt"
                                text-align="justify"
                                line-height="8mm"
                                padding-top="1pt">
                        <xsl:value-of select="."/>
                    </fo:block>
        </xsl:template>
        </xsl:stylesheet >
```

C.2: La plantilla *w3c.fo*

BIBLIOGRAFIA

[App87] Apple Computer Inc. *Human Interface Guidlines: The Apple desktop Interface.* Adison Wesley, 1987, pp. 144.

[MiMo00] Michael, Morrison *XML: Al descubierto* 2000: Prentice Hall Madrid.

[AlaGa98] Aladro, García *El lenguaje XML: la nueva forma de estructurar los contenidos* 1998: Net Magazine 74-77

[MaS98] Mace, S. *Tejer mejor la red* Marzo 1998: Byte España, p. 118-13

[PeTre98] Peña, Tresancos *Estándar XML 1.0: tecnologías para Internet* PC World, nº 144, junio 1998, p. 281-288

[MaPla98] Mack, E. S. Platt *HTML 4.0* Madrid: Anaya Multimedia, 1998

[JaLo92] Jarczyk A., Loffler P., Volksen G. *Computer Supported Cooperative Work (CSCW) State of the Art.* Version 1.0, Internal Report, Siemens AG, Munich, 1992.

[Vol93] Völksen, G. *Approach Strategies to Groupware.* Proccedings Groupware ́93 Europe Conferences in Stockholm, London, Frankfurt, The Conference Group, Scottsdale, AZ, 1993.

[HaRe94] Henry Pérez Luna. *Un Espacio de Trabajo Compartido como Apoyo a la Edición Colaborativa en Internet.* Tesis de Maestría, Posgrado en Ciencia e Ingeniería de la Computación, México, 2001.

[Ell91] Clarence Ellis, Simon Gibs y Gail Rein. *Groupware, Some Issues and Experiences.* Communications of the ACM, vol. 34, No. 1, enero 1991, pp. 30-37.

[Gut96] Carl Gutwin, Saul Green y Mark Roseman. *Workspace Awareness in Real-Time Distributed Groupware: Framework, Widgets, and Evaluation.* Proceedings of the HCI?96, Londres U.K, pp. 281-298.

[NePe91] R. E. Newman-Wolfe y Harsha K. Pelimuhandiram. *MACE: a fine grained concurrent editor.* Conference proceedings on Organizational computing systems. 1991, pp. 240-254

[Ste87] M. Stefik, D. G. Bobrow, G. Foster, S. Lanning y D. Tratar. *WYSIWIS Revised: Early Experiences with Multiuser Interfaces.* ACM Transactions on Information Systems, 1987, pp. 147-167.

[Elx02] Manuel Romero Salcedo, Silvia Ramirez, Cesar Murillo, Anibal Avelar. *ELXI: un Editor Colaborativo de documentos XML en Internet para la elaboración de documentación técnica en el IMP.* PIMAyC, Instituto Mexicano del Petróleo, 2002.

Direcciones URL

[XmlQue] http://mundolinux.iespana.es/mundolinux/dis-web/xml/xml-id3b.htm. ¿Qué es el XML?.

[oasis] http://www.oasis-open.org/cover. Esta enorme página posee mucha información relativa a todos los aspectos de SGML y XML, y se actualiza frecuentemente.

[xml] http://www.xml.com/. Este sitio cubre una serie de herramientas XML e incluye revisiones del rendimiento y compatibilidad de los analizadores.

[wdvl] http://wdvl.com/. La biblioteca virtual del desarrollador web posee una serie de recursos para las aplicaciones web cliente.

[w3Xsl] http://www.w3.org/TR/xsl/Overview.html. Extensible Stylesheet Language (XSL) Version 1.0.

[Axer] http://xml.apache.org/xerces2-j/. Xerces2 Java Parser.

[Afop] http://xml.apache.org/fop/index.html. FOP.

[Axal] http://xml.apache.org/xalan-j/. Xalan-Java.

[Xep] http://www.renderx.com. Sitio de RenderX.

[Rexp] http://www.esng.dibe.unige.it/REXP. Sitio del proyecto REXP.

[JFor] http://www.jfor.org. Proyecto JFor.

[Jfo] http://www.dig-dreams.de. Sitio del proyecto JFO.

[Xslfast] http://www.xslfast.com. Proyecto XSLFast.

[XmlEditor] http://www.treelight.com/software/XmlEditor.html. Design Notes for an XML Editor.

[eFirst] http://www.openly.com/eFirst/. eFirst XML for Scholarly Articles.

[NorBas] http://www.unet.edu.ve/frey/varios/decinv/investigacion/normasbasicas.html. Normas básicas para la redacción de un artículo científico.

[LaEsp] http://laespiral.org/articulos/zaragoza02/index.html. XML: estándar para la estructuración e intercambio de documentos.

[Xslt] http://geneura.ugr.es/~jmerelo/XSLT/. Generación de páginas Web usando XSLT y XML.

[Xslfo] http://www.brics.dk/~amoeller/XML/transformation/fo.html. XSL Formatting Objects.

[Xsltoo] http://neptune.irit.fr/Biblio/xsl_tools.shtml. XSLT and XSL-FO tools.

[Xslpdf] http://programacion.com/articulo.php?id=joa_pdf. Convertir XML en PDF utilizando XSL-FO y FOP.

[Xmlnot] http://gsyc.escet.urjc.es/actividades/linuxprog/xml/xml-notas.html. Introducción al XML.

[Xep] http://www.renderx.com/Tests/doc/html/tutorial.html. Using XSL FO with XEP 3.7.

[Xslsam] http://www.antennahouse.com/XSLsample/XSLsample.htm. Stylesheet Tutorial, Sample Files of Formatting Objects and Sample Stylesheets.

[Algorit] http://ce.sharif.edu/ dic-ads/implement.php. Algorithms and Data Structures.

[JaXm] http://www.programacion.com/java/tutoriales/JavaXML/. Java y XML.

Dedicatorias y Agradecimientos

Dedicada a mi esposa Lidia. Sin ti, nunca hubiera podido concluir este trabajo. Gracias Lidy.

Un agradecimiento a toda la comunidad de usuarios de Debian GNU/Linux por sus consejos y buenos trucos. El presente trabajo fue realizado en su totalidad con herramientas de software libre. En la construcción de la aplicación y la edición del presente documento.

Una especial mención a mi asesor y tutor de la maestría en Ingeniería y Ciencias de la Computación, el Dr. Manuel Romero Salcedo.