VNIVERSIDAD NACIONAL AVFNOMA DE MEXICO

UNIVERSIDAD NACIONAL AUTÓNOMA DE MEXICO

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

DISEÑO, IMPLEMENTACIÓN Y EVALUACIÓN DE UN SISTEMA COLABORATIVO PARA LA ELABORACIÓN Y VISUALIZACIÓN DE DOCUMENTOS TÉCNICOS Y CIENTÍFICOS

T E S I S

QUE PARA OBTENER EL GRADO DE:

MAESTRO EN INGENIERÍA (COMPUTACIÓN)

P R E S E N T A :

RAYMUNDO SANTANA CARRILLO

DIRECTOR DE TESIS: DR. MANUEL ROMERO SALCEDO

MÉXICO, D.F. 2005

Autorizo a la Dirección General de Bibliotecas de la UNAM a difundir en formato electrónico e impreso el contenido de mi trabajo recepcional.

NOMBRE: Raymundo Santana Castillo

FECHA: 14/11/2015

FIRMA:

Diseño, implementación y evaluación de un sistema colaborativo para la elaboración y visualización de documentos técnicos y científicos

Ing. Raymundo Santana Carrillo

7 de febrero de 2005

A mis padres y a mi hermana.

Al Instituto Mexicano del Petróleo.

Gracias al Dr. Manuel Romero S.
por todo el interés y esfuerzo que enfocó en este trabajo.

Gracias al Mat. Jose Francisco Martínez
por todo su apoyo e interés.

...Nada es para siempre.

A Mao, Carlos, Germán, Sun y Gaby
por su amistad y por su apoyo.

...por ese cuaderno que no pudo ser escrito.
Tengo muy claro que sin tu esfuerzo, palabras, dedicación, trabajo y forma de hacerme ver la vida no habría podido llegar aquí.

Índice general

Índice de figuras

Capítulo 1

Introducción

Durante estos últimos años, la Edición Colaborativa Asistida por Computadora (*Computer Supported Collaborative Writing*) se ha consolidado como un nuevo tema de investigación de gran importancia. Esto se debe a que, por un lado, varios trabajos de investigación han reportado que la mayoría de los documentos que se producen son elaborados a través de un esfuerzo colaborativo, y por otro lado, se ha demostrado que varios autores son susceptibles de producir documentos de mejor calidad gracias a la confrontación de sus experiencias, a la combinación de sus conocimientos, a sus niveles de especialización y a la contribución que cada uno puede ofrecer.

De manera natural, en las grandes industrias, como es el caso del IMP y PEMEX, los equipos de trabajo se encuentran distribuidos y muchas veces requieren trabajar juntos en la preparación de una misma documentación. Un ejemplo lo tenemos cuando los líderes de proyecto trabajan en conjunto con sus equipos de gente en la preparación y elaboración de la documentación relativa a los proyectos que desarrollan.

En este trabajo presentamos una contribución importante al diseño, desarrollo y evaluación de un software colaborativo, el cual permite, a un grupo de autores distribuidos, organizar, editar, mantener, revisar y compartir documentos estructurados de tipo **XML** (*eXtended Markup Language*). El prototipo llamado **ELXI** (*Editor coLaborativo de documentos Xml en Internet*), integra mecanismos que resuelven problemas de control (consistencia y concurrencia), coordinación, administración, visualización e integración de

información relativa a la documentación, elaborada por múltiples autores distribuidos, así como a su acceso compartido.

El editor colaborativo **ELXI** es el resultado del trabajo realizado por un equipo de desarrolladores dirigidos por el Dr. Manuel Romero Salcedo. La línea de investigación del Dr. Manuel Romero Salcedo y este trabajo se ubican en el proyecto *D.00006 Computación Distribuida Inteligente* del Programa de investigación en Matemáticas Aplicadas y Computación, bajo la dirección del Dr. Leonid Sheremetov en el Instituto Mexicano del Petróleo.

1.1. Planteamiento del problema

Actualmente, podemos constatar cada vez más en los autores, una necesidad de evolución de un modo de trabajo autónomo e individual (esencialmente de consumo de información) hacia un modo de trabajo más productivo (caracterizado por la colaboración en torno a una actividad común). Esta colaboración es posible ahora gracias al desempeño de los sistemas distribuidos; a la evolución de las redes de comunicación; y a la posibilidad de hacer interoperar computadoras localizadas en sitios físicamente distantes.

Es así como a finales de los 80's nace un área de investigación conocida como Trabajo Colaborativo Asistido por Computadora (*Computer Supported Cooperative Work*). Dicha área se interesa principalmente en el diseño e implementación de modelos, métodos, protocolos y sistemas colaborativos [1] que permiten apoyar al trabajo en grupo con el fin de cumplir un objetivo común.

Objetivo

El objetivo de este trabajo es diseñar, implementar y evaluar mecanismos y algoritmos, que permitan, la edición y visualización de documentos técnicos y científicos. Como caso de estudio para esta tesis se definió la elaboración de dos documentos: un documento tipo **Norma** *y un documento tipo* **General**. *El documento* **Norma** *permitirá editar, de forma distribuida, documentos relacionados a la Normatividad de Referencia usada en PEMEX. El documento* **General** *permitirá editar documentos genéricos para diferentes áreas temáticas.*

En particular, el documento **General** definirá un modelo para distribuir contenidos a un grupo de autores. Estos autores podrán editar, visualizar, mantener, revisar y compartir documentos estructurados de tipo XML (*eXtended Markup Language*) [41] de diferentes áreas temáticas. Este modelo será implementado en el editor colaborativo **ELXI** con el fin de implementar de forma tangible esta propuesta, robusteciendo a su vez el funcionamiento del editor.

Organización de la tesis

Esta tesis se estructuró de la forma siguiente. A manera de contexto teórico, en el capítulo 2 se presenta un panorama de las herramientas y tecnologías vinculadas directamente con este trabajo, así como algunos proyectos de edición colaborativa que actualmente se desarrollan. Se muestran también las características del editor de documentos de Microsoft, el cual, hoy en día, se presenta como el editor de mayor uso en las áreas donde es necesario redactar documentos.

Se muestra, enseguida, en el capítulo 3 la arquitectura y características del sistema **ELXI**, en su versión 1.0, con el fin de plantear los requerimientos para este trabajo. Se presenta también, en este capitulo, la metodología usada a lo largo de este trabajo, la cual permitió llegar a resultados satisfactorios.

En los capítulos 4 y 5 se expone, el análisis, las estrategias, el diseño y la implementación de los casos de estudio definidos en esta tesis. La estructura de estos capítulos consta de tres partes principales: distribución de fragmentos, visualización de los documentos, interfaces para el usuario, y documento **General**. Esta división se concibe con esta estructura, por ser la forma en la que se llevó a cabo el proceso de desarrollo de este trabajo. Finalmente, los últimos capítulos presentan la evaluación, resultados, contribuciones y conclusiones de esta tesis.

Capítulo 2

Estado del arte

Este capítulo muestra las herramientas y conceptos usados en el análisis, diseño e implementación de este trabajo. No es la intención ahondar demasiado en ellos, sino ofrecer los conocimientos necesarios para lograr un mejor entendimiento de los puntos en las que fueron aplicados. Se muestran brevemente conceptos de modelos de objetos distribuidos con **Java** [42], en específico **RMI** [26], la tecnología ofrecida por **Java** para el llamado de métodos remotos. Se explican después conceptos relacionados con la presentación y producción de documentos. En particular, se muestran los conceptos de **metadatos** y su relación con formatos para el manejo de la documentación.

Uno de estos formatos es **XML** [41], el cual fue usado para la implementación de este trabajo. XML marca elementos semánticos en los documentos, permitiendo estructurar su contenido. XML promueve la creación de documentos estructurados a través de marcas, dando de esta forma una estructura funcional al documento. Es decir, cada parte del documento se encuentra definida por alguna marca, permitiendo de esta forma su fácil ubicación. Un documento estructurado es un conjunto de elementos clasificados que están lógicamente relacionados unos con otros. [21]

2.1. Trabajos relacionados

Las actividades de investigación, diseño e implementación aquí descritas se enmarcan en la línea de investigación *del trabajo colaborativo asistido por computadora* y en particular en el área conocida como *Autoría colaborativa.*

El área de autoría colaborativa ha sido objeto de diseño y desarrollo de varios editores de texto, en los que diferentes usuários colaboran con el fin de crear un mismo documento.

Editores, tales como *Sepia* [10] y *Reduce* [12], han sido desarrollados con la intención de generar documentos creados en forma conjunta. Estos editores se enfocan en la creación de documentos orientados al trabajo colaborativo asistido por computadora; pero su diseño no se centra en el concepto de documentos estructurados.

Otros editores, como por ejemplo *Alliance* [13], sí permiten el manejo de documentos estructurados; pero definen estructuras estáticas para el orden de sus tipos de fragmentos. Es decir, la estructura de los documentos en fragmentos —cada fragmento con características definidas— no puede ser cambiada o redefinida, lo cual limita su uso para la edición de documentos enfocados a otras áreas. De esta forma, en estos editores, no se pueden editar documentos fuera del orden de su estructura.

Existen otros editores, como *FAVORITE* [15], el cual define un lenguaje de modelado flexible, con el fin de lograr la reutilización de otros documentos estructurados. Éste inserta partes del documento estructurado en los fragmentos del nuevo documento que se quiere producir conjuntamente. El lenguaje permite definir, de forma dinámica y colaborativa, la estructura del documento a través de restricciones (*constrains*[1]). Estas *restricciones* permiten estructurar el documento, entre varios autores, permitiendo decidir, en primera instancia, la estructura del documento y luego su contenido.

El lenguaje de modelado de *FAVORITE* aún no se ha terminado de definir y los problemas a los que se enfrenta el proyecto son: a) Manejo de fragmentos dinámicos; b) funciones y mecanismos de un fragmento en el servidor; c) intercambio de fragmentos dinámicos entre el servidor y las aplicaciones que procesan los documentos.

ELXI (*Editor coLaborativo de documentos XML en Internet*) es un producto de software colaborativo, el cual permite promover y aumentar la colaboración durante la elaboración de documentos técnicos y científicos, para

[1]Del término en inglés *constrains*: restringir, impedir, detener.

los cuales, en la gran mayoría de las ocasiones, diversos especialistas participan y contribuyen trabajando desde diferentes computadoras unidas por una red corporativa (Intranet) o incluso Internet. Con este producto de software colaborativo, diferentes autores pueden trabajar juntos y de manera simultánea en la edición, el mantenimiento y la revisión de proyectos documentales comunes.

ELXI permite la edición de documentos estructurados de forma colaborativa. Cada documento está constituido por fragmentos, los cuales son parte del documento completo. El presente trabajo se basa en este editor de documentos.

Se propone la definición e implementación de fragmentos con una cantidad finita de *elementos de edición* contenidos en éste. Los elementos de edición son objetos necesarios para conformar un documento —listas, tablas, parrafos, texto, imágenes, etc.—

Estos elementos de edición serán embebidos en los fragmentos, y el contenido de cada fragmento se considerará como un pequeño documento estructurado. Así, el sistema será capaz de editar partes de un documento específico; en razón a la cantidad y capacidad de los elementos de edición que contenga la estructura del documento de **ELXI** para cada fragmento. De esta forma, la estructura del documento podría crecer, si se deseara, para editar documentos más complejos.

En los párrafos anteriores se mencionaron algunos trabajos existentes, pero directamente en el mercado el producto más fuerte, como editor de texto, es Microsoft **Word 2003** [47] trabajando en conjunto con **SharePoint** [27]. Para mostrar la viabildad de **ELXI** como proyecto —dentro del proceso de desarrollo de nuevos productos del IMP— se realizó un cuadro comparativo entre Word 2003-SharePoint y ELXI. Se muestran, pues, las características, pros, y contras de los editores, delimitando los alcances y enfoques de estos.

Los parámetros tomados para realizar la comparación se basan en la literatura existente [20]. Cada uno de los parámetros representan características típicas de los sistemas de autoría colaborativa.

Comenzaremos con el cuadro comparativo, seguiremos con las tecnologías implicadas, y por último, mostraremos los conceptos de metadatos y documentación estructurada.

2.2. Evaluación de ELXI y SharePoint–Word 2003

Word 2003 perfila sus nuevas características hacia la edición de documentos de forma conjunta por grupos de trabajo. En esta sección se presenta un análisis comparativo entre Word 2003–SharePoint y el sistema de edición colaborativa **ELXI**. Este análisis fue realizado para comparar a **ELXI** con respecto a otros editores en el mercado, con el objeto de mantener, como se mencionó, a **ELXI** como un proyecto viable dentro del proceso de *desarrollo de nuevos productos* dentro del **IMP**.

Se muestran características específicas de Word 2003 y SharePoint para después mostrar una tabla comparativa de éstas contra **ELXI**. En esta sección no se presentan las características de **ELXI** debido a que, más adelante, en la sección 3.1 se tratarán éstas con todo detalle.

Información General de SharePoint Portal Server 2003

SharePoint es un portal que facilita la colaboración mediante la activación de las capacidades de agregación, organización y búsqueda de personas, equipos e información. Los usuários pueden encontrar información relevante de una forma rápida mediante la personalización del contenido y el diseño del portal. Las organizaciones pueden dirigir información, programas y actualizaciones a comunidades específicas según sea su función dentro de la organización, el equipo al que pertenezcan, sus intereses, grupo de seguridad o cualquier otro criterio de pertenencia que se pueda definir.

El portal ofrece, para los *sitios*[2] creados en él, herramientas de organización y administración, y permite que los equipos publiquen información útil para toda la organización.

[2]Conjunto de páginas **HTML** y archivos electrónicos que ofrecen algún servicio o información en particular. Se incluye una página inicial de bienvenida con un nombre de dominio y dirección de Internet específicos.

Conexión de espacios y personas que colaboran

SharePoint Portal Server 2003 proporciona un entorno potente de colaboración en equipo que permite a las organizaciones agregar, organizar, encontrar y proporcionar sitios de SharePoint a toda la empresa. Los sitios de SharePoint para equipos, documentos y reuniones también se pueden hacer extensibles a clientes y socios, aumentando el alcance y la eficacia de los métodos de colaboración existentes. De esta forma, el portal permite a los profesionales de la información encontrar fácilmente a las personas, los equipos y las mejores prácticas existentes.

Acceso compartido a un mismo documento por múltiples autores

En SharePoint la notificación de cambios en documentos se lleva a cabo por medio de *alertas*. Las alertas son correos electrónicos enviados a petición del usuario que comparte un documento. Si existe algún cambio por otro usuario en el documento, se envía un correo electrónico notificando el cambio.

Para editar un documento, el cual probablemente este siendo usado por otro usuario, se solicita una copia de éste. Si el documento no esta siendo usado por otro usuario se permite *bajarlo* del sitio; de lo contrario se advierte que es utilizado por alguien más y el documento se bloquea. En SharePoint no existen bloqueos por periodos de tiempo, ni bloqueos a partes específicas de un documento. En SharePoint todo el proceso de colaboración se hace de forma asíncrona [3].

Envío de mensajes-conversaciones entre autores

La comunicación entre autores se lleva acabo a través de herramientas externas a SharePoint. Si el usuario necesita comunicarse con otros autores puede hacer uso de otras herramientas como el mensajero de Microsoft *Messenger* [30], permitiendo establecer una comunicación con la cual se pueden tomar decisiones acerca del documento que se esta editando.

[3]Cuando un documento es editado en tiempos diferentes, se dice que la interacción con el documento se dá de forma asíncrona. Cuando el documento se edita en un mismo tiempo se dice que se interactúa de forma síncrona.

Muchas herramientas de Microsoft permiten que los usuários se comuniquen entre si redireccionando (exportar) la salida de sus monitores a otros usuários por medio de *NetMeeting* [40]. Con *NetMeeting* los demás usuários pueden observar y controlar (si tiene los permisos necesarios) la máquina del usuario que *exportó* su salida. Word 2003, en uno de sus menús de opciones, permite la llamada a *NetMeting*, logrando con esto comunicar a los autores de un documento.

Manejo de notas electrónicas-anotaciones-comentarios

SharePoint permite hacer anotaciones a partes de una página del sitio. Estas anotaciones, relacionan textos con secciones de una página dentro del sitio.

El mecanismo de funcionamiento se realiza a través de la agregación de textos de forma dinámica. Se relacionan partes específicas de la página, en la parte que fue marcada, a un texto y esta relación se toma como una anotación en la página *Web*. Las anotaciones permiten comunicar ideas sobre alguna sección de la página Web a otros usuários. Las anotaciones en documentos Word son independientes a la característica de anotaciones de SharePoint y se realizan directamente en el editor de textos Word 2003.

Administración de autores y Administración de grupos de autores

Los permisos a documentos y grupos se definen desde la creación y acceso al *sitio* con el fin de compartir los documentos de las *bibliotecas de documentos*[4].

Los tipos que se manejan son:

1. `Word`
2. `Power Point`
3. `Front Page`
4. `Excel`

[4]Las *bibliotecas de documentos* son conjuntos de documentos que se desean compartir.

5. `Página Básica HTML`

6. `Página elementos Web`

También, en la creación del sitio, se puede especificar, si se desea, un manejo de versiones y el tipo de documento que podrá manejar determinada biblioteca.

Una vez creada la biblioteca de documentos, se pueden administrar documentos del tipo especificado para esa biblioteca. Las opciones de administración ofrecen las siguientes acciones:

- `Crear nuevos documentos`. Crea nuevos documentos interactuando con la aplicación en que se crea (ej. Word, Excel, etc.)
- `Agregar documentos`. *Sube* documentos al sitio desde un una ruta definida por el usuario.
- `Filtrar documentos`. Con políticas definidas permite, o no, al sitio contener documentos específicos.
- `Editar documentos`. Interactúa con la aplicación (ej. Word, Excel, etc.) para editar documentos.

Manejo de roles de edición

Los roles posibles, que un usuario del sistema puede tener, se configuran por medio de una página, invocada desde el menú general, la cual permite la *configuración del sitio*. Desde esta página se administran los permisos de los usuários, sobre el sitio completo, es decir, los permisos asignados a un usuario, serán válidos para todos los documentos del sitio.

Los roles específicos para los usuários, siguiendo la forma de esta administración, son:

- `Lector`. Sólo puede leer documentos, pero no puede hacer modificaciones.
- `Colaborador`. Puede leer y modificar documentos, pero no tiene restricciones en los permisos de administración.

- `Diseñador Web`. Tiene capacidades de modificación de la presentación del sitio.

- `Administrador`. Puede dar o quitar permisos a otros usuários y tener ingreso a cualquier documento o parte del sitio para su modificación o administración.

Formatos de documentos y exportación a otros formatos

Los formatos para documentos que soporta SharePoint están enfocados a la tecnología de Microsoft. Estos formatos son: Word, Power Point, FrontPage, Excel y página **HTML** básica. Debido a que SharePoint es un sitio Web, se permite *subir* cualquier tipo de archivo, pero el enfoque directo del portal esta dirigido hacia archivos de Microsoft.

Cabe mencionar que, como caso específico, el formato XML se usa para un tipo de documento particular contenido en *bibliotecas de formulario*. Para hacer uso de esta biblioteca, debe estar instalado un editor XML compatible con SharePoint.

Plataforma S.O. soportada (Windows, Unix, MAC, etc.)

La plataforma, invariablemente, es la ofrecida por *Microsoft*, aunque por su diseño como sitio *Web*, SharePoint puede ser usado y administrado desde cualquier otro navegador diferente al *navegador* de Windows. Cabe señalar que si se usa otro navegador, el enfoque hacia la plataforma y herramientas de *Microsoft* se pierde.

Manejo de control de versiones-revisiones

El control de versiones se realiza haciendo copias de los documentos modificados y almacenados. Cada uno de estas copias se guarda en una ruta determinada en el servidor y se accede posteriormente a estas copias por medio de una liga desde una página *HTML*. Por cada cambio, se envía un correo notificando la modificación y se guarda una copia del documento correspondiente modificado.

Manejo de la conciencia de grupo

El manejo de la conciencia de grupo se realiza por medio de ligas a páginas donde se dan informes de tareas, contactos, eventos y anuncios en el sitio. Toda esta información se dá en tiempos diferentes (forma asíncrona).

El usuario puede solicitar el envío de *alertas* si ocurre algún cambio en el documento. Estas alertas son configuradas en relación a los tipos de cambio realizados en el documento:

- `Todos los cambios`. Cualquier cambio en el sitio.
- `Elementos cambiados`. Cambios a elementos específicos (ej. documentos)
- `Actualizaciones de discusiones Web`. Cambios en los foros de discusión del sitio[5].

La configuración es realizada a través de página en el portal, donde se pueden seleccionar opciones de envío de alertas. Se tiene también, en esta página, la capacidad de especificar la frecuencia en que las alertas son enviadas. La posibles configuraciones de frecuencia de envío de alertas es la siguiente:

- `Enviar correo electrónico inmediatamente`. Una vez realizado algún cambio se envía un correo electrónico.
- `Enviar un resumen diario`. Se envía un correo diariamente con los cambios totales de un día.
- `Enviar un resumen semanal`. De forma similar al anterior pero por semana.

[5]El manejo de discusiones es otra forma de interacción entre los usuários. En éste se definen grupos de discusiones *Web*, lo que permite por medio de una página enviar propuestas y respuestas a un tópico determinado del sitio donde se está trabajando.

Interacción con un documento (síncrono - asíncrono)

Cuando un documento es editado en tiempos diferentes, como ya se mencionó, se dice que la interacción con el documento se dá de forma asíncrona. Cuando el documento se edita en un mismo tiempo se dice que se interactúa de forma síncrona.

La interacción del documento con el usuario en SharePoint se dá de forma asíncrona. La comunicación y acceso a un documento se realiza en la medida en que opera el mismo protocolo de comunicación —http— del portal.

Manejo de seguridad (control de acceso al sistema)

El manejo de seguridad es soportado por el sistema operativo donde el servidor SharePoint esté instalado. Éste es el encargado de la políticas que dán, o no, acceso a las páginas que son solicitadas. Existe un administrador general del sistema operativo —Windows 2000—, el cual a su vez agrega usuários que pueden hacer uso de SharePoint.

Integración de Word 2003 con SharePoint

La integración de **Word 2003** con **SharePoint** permite tener un ambiente de colaboración de documentos. En este ambiente se pueden crear espacios de trabajo directamente desde Word.

La unión de *Word 2003*, *SharePoint* y *MSN Messenger* [30] dá como resultado un ambiente colaborativo para la organización y administración de grupos de trabajo.

Espacios de trabajo para documentos

Un sitio creado por SharePoint es definido como un espacio de trabajo. Un documento Word es asociado a el sitio, logrando con esto compartir el documento. Las características de SharePoint explicadas anteriormente se unen a la interfaz de Word 2003 para tener el llamado espacio de trabajo. En Word los *espacios de trabajo*, permiten tener objetivos, agendas, asistentes, etc. sobre el documento que se esté editando.

Lo que en realidad sucede es la creación de un sitio en SharePoint relacionado con el documento que se esta editando en Word 2003.

Panel de tareas de trabajo compartido

Las relaciones entre el documento editado en Word 2003 y el sitio generado en SharePoint se muestran a través del **panel de tareas de trabajo compartido** en la interfaz de Word 2003. Agregando el Messenger a este panel de tareas se da, en conjunto, el ambiente de colaboración que ofrece la plataforma *Microsoft*.

Desde el menú principal en Word 2003 se puede seleccionar la opción *creación de una área de trabajo compartida*, la cual dará acceso al sitio en SharePoint para el documento que se está editando.

El sitio en SharePoint pedirá la clave y contraseña para poder ingresar. Word presenta entonces un panel para el área de trabajo compartida, el cual permite la administración de algunos elementos del sitio en SharePoint. Los elementos de administración están enfocados a compartir el documento que se este editado en ese momento. De esta forma, el *área de trabajo* es creada en SharePoint y puede ser accesada, ahora, desde el panel de área de trabajo compartida de Word.

Al compartir documentos, Word presenta opciones de actualización de documentos. Si existen diferencias entre la copia en el servidor de SharePoint y el documento actualmente en edición, se informa al usuario y se solicitan opciones. Las posibles opciones son: reemplazar el documento por la copia en el servidor, abrir ambas copias, realizar cambios manualmente o combinar las copias.

La selección -*combinar*- muestra en diferentes colores las diferencias entre los documentos y permite la edición de estos. Después de los cambios necesarios, el documento modificado se guarda en el servidor.

La integración de SharePoint y Word 2003 permite la administración, a través de interfaces en Word, del sitio donde se comparten los documentos compartidos por varios usuários.

De este modo, los roles, bloqueos y otras características mencionadas anteriormente de SharePoint pueden ser usadas desde Word, por medio de interfaces manejadas desde el panel de administración de áreas de trabajo compartida de Word 2003.

Cuadro comparativo

SharePoint permite colaborar subiendo y bajando documentos con permisos en grupos y documentos. Al compararlo con **ELXI**, en lo referente a la edición colaborativa de documentos, coinciden en el manejo de grupos y permisos, y en la interacción de un documento de forma asíncrona.

ELXI es un editor de documentos y *SharePoint* es un portal *Web*, por lo cual cada uno se enfocan a los problemas para los que fueron diseñados. Con *ELXI* se editan documentos en forma colaborativa y *SharePoint* comparte diferentes documentos a través de un portal.

Cuando se integran las herramientas *Word 2003, SharePoint y MSN Messenger* se logra un ambiente colaborativo de forma asíncrona, el cual necesita de licencias para cada producto y recursos para hacer funcionar todo el ambiente. De esta forma, como se observa en el cuadro comparativo 2.2, *ELXI* proporciona un ambiente edición de documentos más rico debido a su enfoque, además de un menor uso de necesidades de computo para su uso.

Cuadro 2.1: Cuadro comparativo

Característica	Herramienta	
	ELXI	**Word 2003 + SharePoint**
Edición de texto básico	Permite la edición fluida y necesaria par editar cualquier tipo de documento	La edición es muy robusta teniendo todas las características ya conocidas de Word de Microsoft.

Sigue ...

Cuadro 2.1: Cuadro comparativo (continuación)

Característica	Herramienta (continuación)	
	ELXI	**Word 2003 + SharePoint**
Estrategia de colaboración	Permite la colaboración síncrona y asíncrona. Se comparten documentos completos o fragmentos de estos.	Se basa principalmente en SharePoint, el cual permite colaborar de forma asíncrona subiendo y bajando documentos de un sitio Web.
Acceso compartido a un mismo documento por múltiples autores	Se puede compartir el documento por medio de fragmentos, los cuales pueden visualizar los cambios del documento en el momento de adición de forma síncrona.	Un documento se puede *bajar* del sitio Web, y ser usado por algún usuario; en este periodo de tiempo nadie más puede hacer uso del mismo.
Administración de autores	Los autores juegan diferentes roles tanto en grupos, documentos y fragmentos. Se crean grupos de trabajo donde el creador del grupo puede asignar permisos.	Los autores juegan diferentes roles en todo el sitio Web, se generan grupos de administración de documentos y foros de discusión para estos grupos en SharePoint.

Sigue

Cuadro 2.1: Cuadro comparativo (continuación)

Característica	Herramienta (continuación)	
	ELXI	**Word 2003 + SharePoint**
Administración de grupos de autores	Se permite el manejo de autores, grupos de trabajo y documentos con diferentes roles de edición asignados a cada uno de ellos	Se lleva a cabo en Share Point, se administra el sitio Web desde interfaces en Word, permitiendo el manejo de grupos de trabajo, autores, foros de discusión, etc.
División de un documento en diferentes partes (fragmentos, secciones)	Cada documento se divide en fragmentos, los cuales pueden o no ser compartidos con diferentes permisos. Además, se pueden asignar estos mismos roles y permisos a documentos completos.	Se pueden compartir sólo documentos completos.
Manejo de roles de edición	Se asignan tres tipos de roles permitiendo la edición independiente por fragmento, documento o grupo de trabajo.	Se asignan cuatro roles de edición para documentos en el sitio completo.

Sigue ...

Cuadro 2.1: Cuadro comparativo (continuación)

Característica	Herramienta (continuación)	
	ELXI	**Word 2003 + SharePoint**
Manejo de la conciencia de grupo	De forma síncrona se pueden conocer los usuários conectados en el momento de la colaboración. Se tiene un Chat básico para la comunicación entre usuários.	Se tiene un control de versiones robusto, se pueden saber de forma asíncrona los cambios en documentos, por correos electrónicos se informa de cambios en los documentos.
Manejo de control de versiones/revisiones	Se guarda sólo el documento actual en un directorio definido.	Dá información por medio de una página Web de todos los archivos modificados, por autor y por fecha, además de enviar correos electrónicos en el momento en que son modificados a los usuários que lo desean saber.
Interacción con un documento (síncrono (tiempo real)/asíncrono (en tiempos diferidos)	Se realiza de forma síncrona y asíncrona gracias a un servidor de documentos que se encarga de esta administración.	Se realiza en forma asíncrona, debido a que se comparten documentos desde un sitio Web.

Sigue

Cuadro 2.1: Cuadro comparativo (continuación)

Característica	Herramienta (continuación)	
	ELXI	**Word 2003 + SharePoint**
Envío de mensajes - conversaciones entre autores	Se tiene un Chat de comunicación básico integrado a ELXI	Se sugiere utilizar el Chat de Microsoft, el cual se puede integrar a la propuesta de colaboración.(Word - SharePoint - Messenger).
Formato de documentos (XML, HTML, Word, etc.)	La estructura básica es XML, lo cual permite la visualización, con la ayuda de XSL Y XSL-FO, en formatos **HTML**, **RTF**, y **PDF**.	Maneja los tipos de documentos que se puedan abrir con Word.
Exportación a otros formatos	Se permite exportar a formatos **HTML**, **RTF** y **PDF**	Se permite guardar documentos en diferentes tipos de formatos. Si se necesitara exportar a **PDF** es necesario bajar un *plug-in* extra.

Sigue . . .

Cuadro 2.1: Cuadro comparativo (continuación)

Característica	Herramienta (continuación)	
	ELXI	**Word 2003 + SharePoint**
Manejo de notas electrónicas - anotaciones - comentarios	Existe una estructura especial para el manejo de notas, donde al compartir fragmentos o documentos de forma síncrona se puede compartir notas tanto de forma síncrona como asíncrona.	Se permite el mismo tipo de notas electrónicas conocidas e Word, al compartir un documento completo se comparten también las notas.
Manejo de seguridad (control de acceso al sistema)	Se realiza por medio del sistema operativo donde está instalado el servidor ELXI (Linux Red Hat), la seguridad se realiza a través de la seguridad que proporciona el sistema Linux.	El control de acceso se lleva a cabo por medio del servidor (Windows 2003 Server) donde está instalado el servidor de SharePoint, los privilegios que se tengan en el servidor es la seguridad que se ofrece al sistema.
Herramienta de pizarrón blanco	No existe	No existe.

Sigue

Cuadro 2.1: Cuadro comparativo (continuación)

Característica	Herramienta (continuación)	
	ELXI	**Word 2003 + SharePoint**
Plataforma S.O. soportada (Windows, Unix, MAC, etc.)	Las soportadas por Java; probadas: Windows y Linux.	Windows 2003 Server.
Red (Internet, Intranet)	Se permite el trabajo en redes de área local —Intranet— así como en Internet	Se permite el trabajo en redes de área local así como en Internet.
Dependencia con otros productos	Se debe tener el *Java Runtime Environment* de Sun, y el sistema Linux para su funcionamiento, ninguno de los productos mencionados necesita licencias.	Están sumamente relacionados los tres productos para que el ambiente colaborativo funcione. Por cada producto se deben adquirir licencias.

En esta tabla se observan las diferencias entre **ELXI** y Word 2003. Cada uno de los productos, como se mencionó, ofrecen características enfocadas a la resolución de necesidades específicas. **ELXI**, de este modo, es un editor que ofrece muchas características interesantes que lo hacen un producto factible como editor colaborativo desde su diseño.

Continuaremos ahora con las tecnologías usadas en el desarrollo de este trabajo. Cada una de ellas fue usada en el diseño e implementación, por lo cual se hablará de ellas en las siguientes secciones.

2.3. Modelos de objetos distribuidos

Un sistema distribuido requiere que las partes que lo componen se ejecuten en diferentes espacios de direcciones o máquinas y que estos componentes tengan la capacidad de comunicarse entre sí.

Una de las primeras tecnologías para implementar sistemas distribuidos fueron los *sockets*[6]. Los *sockets* tienen la capacidad de comunicar dos procesos, ya sea mediante datagramas o flujos de datos. Sin embargo, los sockets requieren que las aplicaciones implanten sus propios protocolos para codificar y decodificar los mensajes que intercambian, lo que introduce una problemática diferente a la naturaleza del problema que se pretende resolver y aumenta la posibilidad de errores durante la ejecución.

La alternativa que surgió al empleo de *sockets*, y que se implementa con base en ellos, es la tecnología **RPC** [7], donde la comunicación, entre los elementos que componen el sistema distribuido, se realiza mediante la invocación de funciones que se encuentran en espacios de direcciones diferentes. En este caso, el programador tiene la *impresión* de trabajar con procedimientos locales, mientras que en realidad **RPC** se encarga de empaquetar los argumentos y enviarlos al proceso que contiene el código que implementa a la rutina remota.

2.3.1. Sistemas distribuidos con Java

Como es de esperarse, en los lenguajes cuyo paradigma de programación no es el procedimental, sino el orientado a objetos, no se requiere ya invocar procedimientos remotos, sino métodos de objetos remotos. El empleo de objetos distribuidos Java, en lugar de procedimientos remotos, implica varias ventajas como la orientación a objetos misma, movilidad de las aplicaciones

[6]Del término en inglés *socket*: enchufe, encaje, cuenca

[7]Del acrónimo en inglés *Remote Process Call*: Llamadas a Procedimientos Remotos

Java, los patrones de diseño, la seguridad, la recolección de basura distribuida, etc.

La principal desventaja de los objetos distribuidos de Java, con respecto a las llamadas a **procedimientos remotos** y **sockets**, es definitivamente el rendimiento. Esta desventaja es inherente al mismo lenguaje Java, pero así mismo, todas las características positivas del modelo de objetos distribuidos de Java, lo hacen una alternativa bastante interesante con respecto a sus contrapartes procedimentales de más bajo nivel.

Aplicaciones con Objetos Distribuidos

Un programa orientado a objetos consta de una colección de objetos que interactúan entre sí, solicitando y proporcionando servicios mediante el intercambio de mensajes. Este paradigma se ajusta bastante bien al diseño de sistemas distribuidos, debido a que estos mensajes enviados y recibidos por los objetos, pueden ser extrapolados a mensajes enviados a objetos en diferentes máquinas a través de la red. Los objetos distribuidos radican en aplicaciones que pueden actuar como clientes, servidores o ambos, dependiendo si contienen objetos remotos, referencias de ellos o ambos.

2.3.2. Java RMI: Remote Method Invocation

RMI [26] es un modelo de objetos distribuidos, diseñado específicamente para el lenguaje de programación Java. Por lo que mantiene la semántica del modelo de objetos locales de Java, facilitando de esta manera la implantación y el uso de objetos distribuidos. En el modelo de objetos distribuidos de Java, un objeto remoto es aquel cuyos métodos pueden ser invocados por objetos que se encuentran en una *máquina virtual (MV)* diferente. Los objetos de este tipo manifiestan su comportamiento por una o más interfaces remotas.

Una aplicación servidora que utiliza RMI, generalmente crea cierto número de objetos remotos, permite que se creen referencias a dichos objetos, y espera a que algún cliente invoque de manera remota los métodos de dichos objetos.

Una aplicación cliente típica obtiene referencias de uno o más objetos remotos e invoca sus métodos por medio del sistema RMI de Java. RMI se encarga de proporcionar los mecanismos mediante los cuales, tanto clientes como servidores, pueden comunicarse.

Interfaces e implementación

Uno de los primeros detalles en la construcción de aplicaciones *RMI* involucra el concepto utilizado en los lenguajes orientados a objetos: la separación entre *interfaz* e *implementación*. La interfaz es únicamente una declaración de él o los métodos definidos en la implementación. Esto es, en la implementación se encuentra definida la lógica mientras que la interfaz funciona como una declaración.

Skeletons y *hubs*

RMI utiliza el mismo mecanismo —esqueletos (*skeletons*[8]) y puntos de conexión (*hubs*[9])— que los sistemas *RPC* para implementar la comunicación con los objetos remotos. Los *puntos de conexión* forman parte de las referencias y actúan como representantes de los objetos remotos ante sus clientes. En el cliente se invocan los métodos del *cabo*, quien es el responsable de invocar de manera remota al código que implementa al objeto remoto. En *RMI*, un *cabo* de un objeto remoto, implementa el mismo conjunto de interfaces remotas que el objeto remoto al cual representa.

Cuando se invoca algún método de un *cabo*, éste realiza las siguientes acciones:

- Inicia una conexión con la *maquina virtual (MV)* que contiene al objeto remoto.
- Ordena y transmite los parámetros (*marshall*)[10] de la invocación a la MV remota.
- Espera por el resultado de la invocación.

[8]Del término en inglés *skeleton*: esqueleto
[9]Del término en inglés *hubs*: centro, maza, cubo
[10]Del término en inglés *marshall*: dirigir, ordenar, organizar

- Retoma el orden de los parámetros *(unmarshall)* [11] y devuelve el resultado de retorno o la excepción.
- Devuelve el valor a quien lo llamó.

Los *puntos de conexión* se encargan de ocultar el ordenamiento de los parámetros, así como los mecanismos de comunicación empleados. En la *MV* remota, cada objeto debe poseer su esqueleto correspondiente. El esqueleto es responsable de dar salida a la invocación al objeto remoto.

Cuando un esqueleto recibe una invocación, realiza las siguientes acciones:

- Ordena los parámetros necesarios para la ejecución del método remoto.
- Invoca la implementación del método.
- Toma el orden de los resultados y los envía de vuelta al cliente.

Registro RMI

La conectividad entre las funciones remotas requiere definir lo que se denomina **registro RMI**. El **registro RMI** se establece en un puerto *TCP/IP*, que por defecto es el 1099 y mantiene un mapa de las funciones remotas que serán utilizadas. Es decir, cuando llega la solicitud para una función remota en el puerto conocido, el registro RMI será el encargado de redireccionar a la implementación apropiada.

La utilización de un puerto *TCP/IP* implica que de alguna manera la función remota debe ser ubicada a través de un mecanismo. Esto generalmente se hace mediante un RMI URL, por ejemplo:

rmi://elxi.imp.mx:4000/ServidorElxi

Se indica que se va a buscar el servidor *elxi.imp.mx*, que está corriendo la aplicación servidor de **ELXI**, en el puerto 4000.

[11] Del término en inglés *unmarshall*: opuesto a *marshall*

Bajo *TCP/IP*, RMI emplea el protocolo llamado **JRMP** *Java Remote Method Protocol*, para ejecutar métodos en los diversos objetos del sistema.

Las aplicaciones pueden utilizar dos mecanismos para obtener referencias a objetos remotos:

- Registrar sus objetos remotos con la facilidad de nombrado de *RMI rmiregistry*.
- Pasar y devolver referencias de objetos remotos como parte de su operación normal.

Marshall y unmarshall

Si se ejecuta un programa en una sola computadora, se tiene la seguridad de que la representación de datos está conforme a la plataforma empleada, sea *SunSolaris, Linux, Windows*, etcétera. Sin embargo, ¿Qué ocurre si se intentan pasar parámetros a un procedimiento en un servidor *Linux* de una computadora utilizando *Windows*?

Por los procesos de *marshall* y *unmarshal* debe pasar toda información para que ésta sea utilizable en ambientes heterogéneos. Como RMI permite al invocador pasar objetos Java a objetos remotos, RMI proporciona el mecanismo necesario para cargar el código del objeto, así como la transmisión de sus datos.

Carga dinámica de clases

RMI utiliza el mecanismo de *serialización de objetos de Java* para transmitir datos entre máquinas y además agrega la información de localización necesaria para permitir que las definiciones de las clases se puedan cargar en la máquina que recibe los objetos.

Cuando se realiza la recepción, para los valores de retorno o para los parámetros de una invocación, es necesario poseer las definiciones de las clases de todos estos objetos, para convertirlos en objetos activos dentro de la MV que los recibe.

En primera instancia, el proceso de recepción intenta resolver las clases por su nombre en el contexto del cargador de clases local. En caso de no encontrar las definiciones de manera local, *RMI* proporciona carga dinámicamente la definición de las clases de los objetos recibidos, desde las direcciones especificadas debida a la información contenida en la invocación. Esto incluye cargar dinámicamente las clases de los *puntos de conexión*, así como cualquier tipo pasado por valor en la llamada RMI.

Para soportar la carga dinámica de clases, RMI utiliza subclases especiales de *java.io.ObjectOutputStream y java.io.ObjectInputStream*, para el manejo de flujos de objetos. Estas subclases especializan al método *annotateClass* de la clase *ObjectOutputStream* y al método *resolveClass* de la clase *ObjectInputStream*, para incluir información sobre la localización de los archivos de clase, que contienen la definición de los objetos contenidos en el flujo.

Sincronización de métodos en Java

La palabra reservada *synchronized* de Java, se usa para indicar que ciertas partes del código, habitualmente métodos de la clase, están *sincronizadas*, es decir, que solamente un subproceso puede acceder a dicho método a la vez.

Cada método sincronizado posee una bandera que puede cerrar o abrir la puerta de acceso. Cuando un subproceso intenta acceder al método sincronizado, mirará si la puerta está cerrada, en cuyo caso no podrá accederlo. Si el método no tiene puesta la llave entonces el subproceso puede acceder a dicho código sincronizado.

Se debe evitar la sincronización de bloques de código y sustituirlos siempre que sea posible por la sincronización de métodos, lo que está más de acuerdo con el espíritu de la programación orientada a objetos.

Es importante destacar que la sincronización disminuye el rendimiento de una aplicación, por tanto, debe emplearse solamente donde sea estrictamente necesario.

Callbacks

Son simplemente invocaciones *RMI* utilizadas para establecer comunicación desde una máquina virtual que normalmente está ejecutando un proceso servidor hacia una máquina virtual que normalmente ejecuta un proceso cliente, de tal forma que el servidor actúa como cliente y el cliente como servidor.

Si vemos las máquinas virtuales como iguales o pares, el proceso de callback[12] se puede describir así:

Una máquina virtual A deja su interfaz en la máquina virtual B a través de un método remoto, para informar a la máquina B que existe. A partir de ese momento, la máquina virtual B puede usar esta referencia para invocar métodos en A.

Para finalizar este capítulo hablaremos de los metadatos y de la documentación estructurada. La finalidad de la siguiente sección es explicar estos conceptos y presentar el lenguaje de marcas extendido XML, el cual fue usado como base para la estructura de los documentos en **ELXI**.

2.4. Metadatos y documentos estructurados

En el desarrollo de las técnicas de documentación, se han ido aplicando una serie de procesos, tecnologías, metodologías, etc., que han sido descubiertas en el propio campo, o bien traídas de la mano de otras disciplinas.

A pesar de esa diversidad de técnicas y procedimientos, siempre se ha observado un fenómeno uniforme: la tendencia a incrementar la eficacia en la organización y recuperación de información.

2.4.1. Metadatos

Una de las líneas emergentes para la descripción de recursos, su localización o recuperación, en documentos electrónicos en la red son los **metadatos**.

[12] Del término en ingles callback: llamar atrás, mandar volver.

El **metadato**, desde la perspectiva documental, se define como *la información sobre una publicación en oposición a su contenido. No sólo incluye descripción bibliográfica, sino que también contiene información relevante como materias, precio, condiciones de uso, etc.* [22]. Es decir, los metadatos son datos acerca de los datos.

Se dice que un **metadato** describe los atributos de un recurso, teniendo en cuenta que el recurso puede ser un objeto bibliográfico, registros e inventarios archivísticos, objetos geoespaciales, recursos visuales y de museos, o implementaciones de software [3].

Existen, hoy en día, con el fin de cubrir las necesidades específicas que requiere el tratamiento de la información en diferentes ámbitos, diferentes formatos de **metadatos**.

Históricamente los formatos más conocidos para representar documentos de cualquier tipo son **TEI** [31] y **DocBook** [23], ambos basados en **SGML** [32].

Hablemos, pues, de **TEI** y **DocBook**, y de los formatos en lenguajes de marcas.

2.4.2. Formatos en lenguajes de marcas

El Lenguaje de Marcado Generalizado Estándar **SGML** es, un metalenguaje con el cual se puede definir la sintaxis de otros lenguajes de marcas específicos.

SGML fue desarrollado por Charles Goldfarb con el nombre de *General Markup Language* **GML**, y tenía como objetivo facilitar el intercambio de documentos en IBM. **GML** permite definir estructuras tipificadas de documentos a partir de las reglas que los rigen, las cuales son expresadas físicamente en los documentos por medio de marcas. Fue adoptado rápidamente para la creación de documentos por el Departamento de Defensa de EUA y por la Oficina de Publicaciones Oficiales de la Comunidad Europea, ambos clientes de IBM [5]. ISO lanzó, luego de un tiempo (1986), la norma ISO 8879 con el nombre de *Standard Generalized Markup Language* **SGML**.

SGML se ubicó como una de las tecnologías más usadas para el intercambio de datos documentales con una arquitectura lógica muy bien determinada. Sin embargo los estándares de descripción de lenguajes y las necesidades del mercado se convirtieron en un problema, debido a la diversidad de lenguajes de marcas que se generaban por cada área específica.

Un documento SGML hace referencia a un conjunto de declaraciones. Estas declaraciones se almacenan en un archivo llamado **DTD** *Document Type Definition*, el cual contiene una estructura de nombres de marcas y especifica las reglas de combinación para las características estructurales y de semántica con las cuales se construye el documento.

SGML es un metalenguaje para la definición de otros lenguajes de marcas. No es la intención adentrarnos demasiado en SGML; pero podemos decir que este lenguaje a dado estructura a muchos otros lenguajes de marcas. Se mostrarán, a continuación otros formatos que son utilizados en diferentes áreas para organizar datos. Estos formatos han sido utilizados ya varios años atrás y precisamente SGML ha servido como base para estructurar algunos de ellos.

DocBook es un lenguaje de marcas para describir libros, artículos, y otros documentos, particularmente documentos técnicos. **DocBook** está definido por la sintaxis de una DTD de SGML. Al igual que **HTML**, **DocBook** es un ejemplo de un lenguaje de marcas definido por SGML. **DocBook** se encuentra actualmente en la versión 4.4 bajo los auspicios de **OASIS** *Organization for the Advancement of Structured Information Standards* [33].

TEI apareció debido a que la mayoría de los esquemas de codificación de lenguajes reflejaban los intereses *científicos* de sus creadores y no de documentos de otras áreas específicas. Es decir, ninguno esquema era lo suficientemente flexible o generalizable para aplicarlo a la codificación de materiales de un amplio espectro de aplicaciones. **TEI** es un formato orientado a la codificación de textos electrónicos, el cual intenta abarcar una amplia variedad de tipos de texto —en general científicos— considerando un número mínimo de características que compartan todos los textos.

En el ámbito documental hallamos dos formatos significativos, **DC** [35] y **EAD** [36]. **EAD** *Encoding Archival Description*, es un formato que cae en el

campo archivístico, en concreto destinado a la descripción de instrumentos de descripción archivística, está también basado en SGML.

DC *Dublin Core* es un formato simple con la intención de aplicación genérica, pero se a consolidado en el ámbito bibliotecario, y por tanto su principal aplicación la tiene en este campo.

El **DC** es un formato simple de descripción de recursos, que ha generado considerable atención, porque se ha situado como una solución potencial para tres requerimientos principales. El primero es tener un formato simple de descripción de recursos. El segundo uso al que se dirige es suministrar una base semántica para metadatos embebidos o adjuntados a documentos **HTML**. El tercer uso al que se dirige es suministrar una base para la interoperabilidad semántica de metadatos más ricos entre dominios.

MARC es otro formato importante que se emplea fundamentalmente en catálogos de bibliotecas o repertorio bibliográfico y define reglas de catalogación y formatos para transmitir información.

Por último, otro formato, es **FGDC** [34], el cual es usado para información geográfica. Este formato es usado desde tiempo atrás.

Así, observamos que para cubrir las necesidades específicas de metadatos, que vienen impuestas por el tipo de información que se utiliza en una organización o en un campo o dominio específico, hay que recurrir a la definición de formatos que se adecúen a esas necesidades. Existe en la actualidad, una gran cantidad de formatos para estructurar documentos de diferentes tipos (como los mencionados anteriormente), pero se están realizando esfuerzos para tratar de estandarizar todos estos lenguajes. De estos esfuerzos, uno de los más importantes como aglutinador es XML el cual, ahora, se utiliza en una gran cantidad de proyectos de diferente índole. Para continuar hablaremos, debido su importancia actual, en la siguiente sección de XML.

2.4.3. Documentación estructurada en el Web y XML

Los avances tecnológicos afectan a los cimientos de la documentación desde su materia prima, el documento, ha visto multiplicar de un modo exponencial sus potencialidades informativas en una doble vertiente. En primer

lugar, siendo mucho más asequible a cualquier persona al superar barreras espaciales y temporales y en segundo integrándose de un modo reticular en las nuevas estructuras derivadas de la consolidación de la informática en nuestros días.

Un documento electrónico tiene un carácter mucho más abierto que un documento bibliográfico. A la luz de estos hechos se redefinen conceptos básicos relacionados con el documento, su tipología y estructuras.

El documento *hipermedia* —en el cual se mezcla lo textual con lo sonoro y visual— abre las puertas a un nuevo escenario documental generando nuevos problemas y demandando nuevas soluciones.

La relación cognitiva del *hypertexto* con el contenido textual a que pertenece permite una representación mucho más precisa y exhaustiva a través de los metadatos que, a modo de etiqueta, se incluye en el documento electrónico.

Buscadores de documentos electrónicos en el Web

Fue reportado que en el año 2003 [4] más de 500 billiones de páginas Web estuvieron disponibles en Internet y el número de peticiones a máquinas de búsqueda hubieron alcanzado el número de 550 millones por día.

Los documentos electrónicos ofrecen buenas perspectivas, pero también muestran limitaciones de recuperación de información *hipertextual* importantes. Se han propuesto, diferentes mecanismos para recuperar información en este tipo de documentos. Estos mecanismos de recuperación se pueden agrupar en tres grupos: índices compilados manualmente, bases de datos creados por robots o arañas, y métodos de *indexación* distribuida.

Los **índices compilados manualmente** consisten en grandes bases de datos, donde la información de ubicación de los documentos es sugerida y categorizada por los dueños del documento con base en un formulario o es manejada por personas expertas en la organización de esta información.

Un gestor de páginas Web se encarga de hacer las búsquedas, realizadas por grupos conceptuales, las cuales a su vez contienen otros grupos conceptuales y así sucesivamente.

Para las **bases de datos creadas por robots o arañas** un robot se utiliza para obtener información de los documentos y alimentar luego con esta información la base de datos.

La información extraida de los documentos se basa principalmente en *URLs* y en palabras significativas contenidas en los *tags* de mayor contenido textual o en las palabra con mayor frecuencia en el documento. Una vez indexado el documento, el robot busca enlaces con otros documentos usando la información obtenida del primer documento. Así, de forma recursiva el robot indexa la información y de nuevo busca enlaces a otros documentos.

Los métodos de *indexación* distribuida se basan en una arquitectura de software y de hardware repartida en los servidores Web. Un software instalado en el servidor Web (*gatherers*) [13] extrae información relativa a los documentos disponibles en el servidor, otro software recupera la información de los *gatherers* y la integra en índices donde se podrá luego hacer búsquedas.

Con la cantidad de documentos en Internet se dificulta encontrar objetos, a pesar de que los servicios de localización, antes mencionados, indexan los recursos disponibles en el Web y actualizan las localizaciones de las BD. Los índices resultantes son útiles en pequeñas colecciones pero conforme se expande su amplitud los índices sucumben ante problemas de recuperación de grandes conjuntos y de generalidades semánticas de disciplinas cruzadas. Para mejorar la búsqueda y recuperación se necesitarán registros más ricos, creados por expertos de contenido. Esta riqueza la pueden suministrar normas formales tales como **TEI** y **MARC**, pero debido a la inversión de tiempo en su creación y mantenimiento, sólo se se podrían aplicar a los recursos más importantes.

Una solución alternativa sería la creación de un registro que sea más informativo que un índice pero menos completo que un registro de catalogación

[13] Del inglés *gather*: acumular, cosechar, poner en orden, deducir

formal. Se describirían más objetos si sólo se requiriera un pequeño esfuerzo humano para crear los registros, especialmente si se animara al autor a crear la descripción. Sí la descripción del registro siguiera una norma establecida, sólo la creación del registro requeriría intervención humana. Las herramientas automatizadas descubrirían estas descripciones y las recogerían. Podría definirse un registro simple de metadatos que describa un amplio rango de objetos electrónicos.

2.4.4. XML y la Web semántica

La consistencia es algo de lo que la actual Web carece. Creando una estructura que facilite la ubicación de los documentos, convertirá a Internet en un gran repositorio de información.

Con la *Web semántica* [39] se busca un significado bien definido a la información, permitiendo un mejor trabajo de colaboración entre personas.

Los metadatos y las ontologías son conceptos relacionados con la *Web Semantica*. Una ontología, dentro del contexto informático, es la especificación de una conceptualización, es decir, una descripción formal de los conceptos y las relaciones entre conceptos.

La Web Semántica se basa en dos conceptos fundamentales:

- La descripción del significado que tienen los contenidos en la Web.
- La manipulación automática de estos significados.

La descripción del significado requiere conceptos ligados a:

- La semántica, entendida como significado procesable por máquinas.
- Los metadatos, entendidos como contenedores de información semántica sobre los datos.
- Las ontologías, entendidas como el conjunto de términos y relaciones entre ellos que describen un dominio de aplicación concreto.

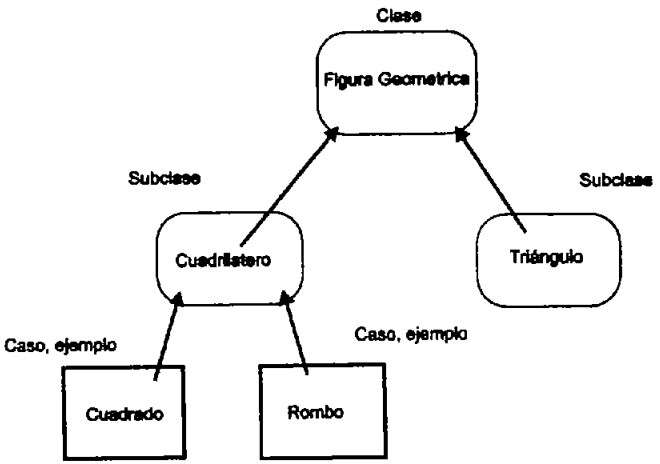


Figura 2.1: Ejemplo de una *ontología*

La Web Semántica se basa principalmente en ontologías formales, las cuales estructuran metadatos, permitiendo así una mejor transferencia y comprensión de los datos para una máquina de búsqueda [6] (ver fig 2.1).

En términos prácticos, el desarrollo de una ontología incluye:

- Definir clases en la ontología
- Colocar las clases en una jerarquía de taxonomías (subclase-superclase)
- Definir atributos y describir los valores permitidos para esos atributos.
- Rellenar los valores de los atributos con ejemplos.

Las ontologías se pueden expresar mediante lenguajes definidos. Uno de ellos es **RDF**.

El lenguaje de descripción de recursos *RDF*

RDF *Resource Description Framework*, desarrollado bajo los auspicios de W3C, es una infraestructura que posibilita la codificación, el intercambio, y la reutilización de los **metadatos** estructurados. Esta infraestructura

permite la interoperabilidad de los **metadatos** a través del diseño de mecanismos que soportan convenciones comunes de la semántica, la sintaxis y la estructura. **RDF** no estipula la semántica para cada comunidad de descripción de recursos, sino que suministra la capacidad de definir tantos elementos de **metadatos** como necesiten estas comunidades.

RDF suministra un modelo para describir recursos, en el que los recursos tienen *propiedades* (atributos o características) y se define el recurso como cualquier objeto que es unívocamente identificable por un URI (Identificador Uniforme de Recursos).

RDF usa XML como sintaxis común para el intercambio y procesamiento de **metadatos** e impone una estructura formal sobre XML para soportar la representación consistente de la semántica.

RDF tiene en cuenta la descripción de recursos de otras áreas para definir su semántica. Un problema implícito es cuando diversas áreas pueden usar un mismo nombre para representar diferentes cosas. Para prevenir esto, **RDF** identifica únicamente nombres mediante el mecanismo *namespace* [14]. *Namespace XML* suministra un método para identificar sin ambigüedad la semántica y las convenciones que gobiernan el uso particular de las *propiedades*.

Los esquemas **RDF** se usan para declarar vocabularios y conjuntos de propiedades de la semántica definida para una área de conocimiento en particular. Los esquemas **RDF** definen las propiedades válidas en una descripción **RDF** dada, además de cualquier característica o restricción de los valores mismos de la propiedad. El mecanismo *namespace XML* sirve para identificar Esquemas **RDF**.

En la siguiente subsección hablaremos más del lenguaje de marcas XML, para entender mejor todos estos conceptos.

2.4.5. *XML* Lenguaje de marcas extendido

Como se mencionó, **XML** es el acrónimo de *eXtensible Markup Language* y tiene sus raíces en SGML. SGML se ha ganado la reputación de complejo;

[14] Del inglés *namespace*: espacio de nombres

a pesar de esto ha tenido una sólida aceptación en el manejo de diferentes documentos (por ejemplo **DocBook**). Para lograr llevar SGML al universo de la Web, se tomó la decisión por parte del **W3C** de simplificar y aplicar una cantidad definida de reglas, manteniendo la capacidad de especificar y crear marcas, con el fin de crear un nuevo lenguaje. En febrero de 1988, el **W3C** publicó la especificación de **XML 1.0**. Desde ese momento, muchas de las tecnologías relacionadas con el Web se han enfocado hacia XML.

XML conserva el concepto de DTD de SGML, mecanismo que define el conjunto de marcas o vocabulario para propósitos específicos. Se dice que un documento está *bien formado* cuando cumple con la sintaxis correcta de XML. Se dice que un documento es *válido* cuando se siguen las reglas de construcción definidas en una DTD.

XML es continuamente descrito como la tecnología sucesora de SGML y no es de sorprender que muchas aplicaciones en SGML estén siendo transportadas a XML [?].

Como se mencionó, **RDF** esta definido sobre XML y se está involucrando en diferentes proyectos como *Warwick Framework* [8], el cual se basa en el conjunto de datos de **DC** *Dublin Core*.

En este rumbo también se encuentra **MARC**, *Lane Medical Library at Stanford* ha iniciado un proyecto para crear una DTD para registros **MARC** el cual es llamado **XMLMARC** [43].

Así, observamos que la estructura XML, como conjunto, junto con **RDF** y DTD permiten la sintaxis exposición, y validación de los metadatos de un documento respectivamente. Es decir, la DTD de un documento permite validar su contenido y con **RDF** se expone este mismo contenido a otras aplicaciones.

2.4.6. Normalización de estilo en XML

Hasta ahora, sólo hemos considerado el contenido de los documentos, pero otro punto importante es la presentación de estos. **DSSSL** y **XSL** son las tecnologías que permiten asignar un estilo de presentación a los documentos, esto es, el contenido dentro de un documento XML está perfectamente

estructurado, pero para presentar esta información se debe poder contar con herramientas específicas.

DSSSL es una norma internacional nominada como ISO/IEC 10179:1996 *Lenguaje de especificación y Semántica de Estilo de Documento.* Esta norma internacional define el Lenguaje de Especificación y la Semántica de Estilo que se usa con el fin de especificar el formateo y la transformación de documentos SGML.

El principal objetivo de esta Norma Internacional es suministrar un lenguaje para expresar el formato, y otras especificaciones de procesamiento de documentos, de una manera formal y rigurosa. Esto con el fin de que las especificaciones puedan ser procesadas por un rango amplio de *formateadores* (programas que dan formato a un documento) de forma natural o usando mecanismos de traducción.

La normalización de la semántica de formateo se suministra en DSSSL a través de un conjunto de estructuras básicas conocidas como objetos de flujo y por un conjunto asociado de características que se aplican a estos objetos. DSSSL suministra mecanismos para definir y extender las construcciones semánticas de forma que los diseñadores de aplicaciones DSSSL puedan construir las aplicaciones DSSSL que mejor se adapten a sus entornos de aplicación.

SGML normaliza la representación de la estructura del documento, dejando a los usuários el desarrollo de sus propias técnicas para interactuar con los formateadores y otros procesadores. Precisamente DSSSL está concebido para apoyar a las aplicaciones de presentación, suministrando una arquitectura normalizada para formateo y otras especificaciones de procesamiento. Con estos se permite también a los usuários poder intercambiar tales especificaciones en un marco entendible y normalizado.

Una especificación DSSSL es normalmente externa al documento SGML. Así múltiples especificaciones DSSSL pueden ser aplicadas a un documento SGML dado, para producir varias presentaciones del mismo documento.

SGML proporcionan la capacidad de distinguir entre el contenido intrínseco y la estructura de un documento, por una parte, y por otra, las especificaciones para procesarlo. Con DSSSL, el formateo y otras especificaciones de

procesamiento pueden ser intercambiadas entre varios documentos SGML, con el fin de suministrar una especificación común de la visualización de estos. Como se observa, de esta forma, se preserva la distinción esencial entre contenido y formato.

De esta manera, una característica clave del *marcado generalizado* es que el formateo, y cualquier otra información de procesamiento asociada con el documento, está separado de las etiquetas de marcado (etiquetas que definen el contenido del documento).

XSL es el acrónimo de *eXtensible Stylesheet Language* y es la parte análoga de DSSSL en XML. Lo componen dos estándares abiertos del **W3C**: **XSL-FO** [15] y **XSLT** [16]. Mientras **XSL-FO** presenta una forma de transformar documentos XML en otros documentos XML o no XML —**PDF**, por ejemplo—, **XSLT** se compone de un conjunto integral de marcado que funciona de forma muy similar a **CSS** [37] y permite entregar documentos XML —**XHTML** [38], por ejemplo— para su presentación visual y/o impresa.

XPath es el lenguaje de expresiones que se usa para referirse a los elementos de un documento XML dentro un archivo **XSL**. Las expresiones **XPath** guardan cierta analogía a la especificación de rutas de directorios en un sistema de archivos.

Es necesario un motor de **XSL** para realizar la transformación. El motor **XSLT** empezará a leer nuestro documento XML desde el nodo raíz, y por cada identificador, de algún elemento que encuentre, intentará ubicar una plantilla (objeto de flujo) en el archivo **XSL** que le indique lo que debe mostrar. Si no la encontrara, mostraría el contenido del elemento de forma igual.

Hoy en día, los navegadores Web más conocidos traen su propio motor **XSLT** incorporado. Otros motores que también destacan son:

- MSXML: Motor de Microsoft que usa Internet Explorer. Está catalogado como uno de los más rápidos del mercado. El principal problema

[15] Acrónimo de *Formatting Objects*
[16] Acrónimo de *XSL Transformations*

que tiene es que es de Microsoft (nunca estará para Linux); no implementa completamente el estandar XSLT hasta la fecha, y no se puede usar desde la línea de consola.

- SABLOTRON: Motor Open-Source que viene con la mayoría de las distribuciones de linux. Hasta la versión 0.90 podemos comprobar que no implementaba completamente la importación y exportación de XSL a través del protocolo http, además de que no tenía soporte para las instrucciones XSLT de formateo numérico. Con documentos XML de 3MB en LaTeX, era extremadamente lento [9].
- XALAN: Motor Open-Source implementado dentro del proyecto Apache. Existen dos implementaciones: Una en Java y otra en C++. Implementa toda la funcionalidad del estándard XSLT.
- Saxon: implementado por Michael Kay, coeditor de XSLT.

2.5. Síntesis y discusión

Observamos que existen, hoy en día, una gran cantidad de formatos para estructurar diferentes tipos de documentos. XML apunta como un importante aglutinador de formatos, debido a que varios formatos, anteriormente implementados con su propio lenguajes de marcas, se están ahora estructurando sobre XML. Con esto se logra una estandarización y una implementación más sencilla con respecto a los formatos anteriores definidos con SGML.

En este trabajo se pretende resolver la edición de un documento **General** estructurado en XML. Este documento permitirá editar y visualizar cualquier tipo de documento en **ELXI**, usando las características de XML para mejorar su estructura colaborativa. La idea principal es embeber pequeños documentos estructurados en los fragmentos del editor **ELXI** y al final obtener un solo documento estructurado compuesto por cada uno de los documentos contenidos en cada fragmento. De este modo, se logrará que la implementación de mecanismos y funciones sean más sencillos para el manejo distribuido de los documentos.

Los mecanismos y funciones serán implementados con JAVA-RMI, el manejo de los documentos se logrará con las bibliotecas **DOM** de Apache, la

visualización se resolverá con XSL y XSL-FO, y la estructuración de los documentos embebidos se hará con XML. Con estas herramientas y el enfoque hacia la estructuración en XML, se pretende implementar un sistema capaz de editar documentos colaborativos, los cuales permitirán un mejor flujo de participacion en grupos de trabajo y por tanto documentos más útiles para las necesidades de intercambio de información que en la actualidad se demandan.

Capítulo 3

Arquitectura del sistema de edición colaborativa ELXI, requerimientos y metodología

3.1. Descripción funcional y arquitectura del sistema ELXI

ELXI es un producto de software colaborativo (groupware), el cual permite promover y aumentar la colaboración durante la elaboración de documentos técnicos y científicos, para los cuales, en la gran mayoría de las ocasiones, diversos especialistas participan y contribuyen trabajando desde diferentes computadoras unidas por una red corporativa (Intranet) o incluso Internet. Con **ELXI** diferentes autores pueden trabajar juntos y de manera simultánea en la edición, el mantenimiento y la revisión de proyectos documentales comunes.

Con **ELXI**, diferentes autores pueden realizar contribuciones sobre un documento compartido, al mismo tiempo (de forma síncrona) o en tiempos diferidos (de forma asíncrona), sin que ello represente problemas de inconsistencia. El sistema **ELXI**, en su version 1.0, permite a varios autores editar un mismo documento de tipo XML y permite *visualizar* el documento elaborado en diferentes formatos, tales como *HTML y PDF*.

La arquitectura de **ELXI** es de tipo cliente-servidor. El servidor **ELXI** está compuesto por una base de datos, un servidor **JAVA-RMI** y el sistema de archivos. La conexión del servidor **ELXI** con la base de datos se realiza a través de JDBC[1]. El análisis sintáctico de los documentos XML en los clientes **ELXI** se realiza por medio **DOM**[2] (ver figura 3.1).

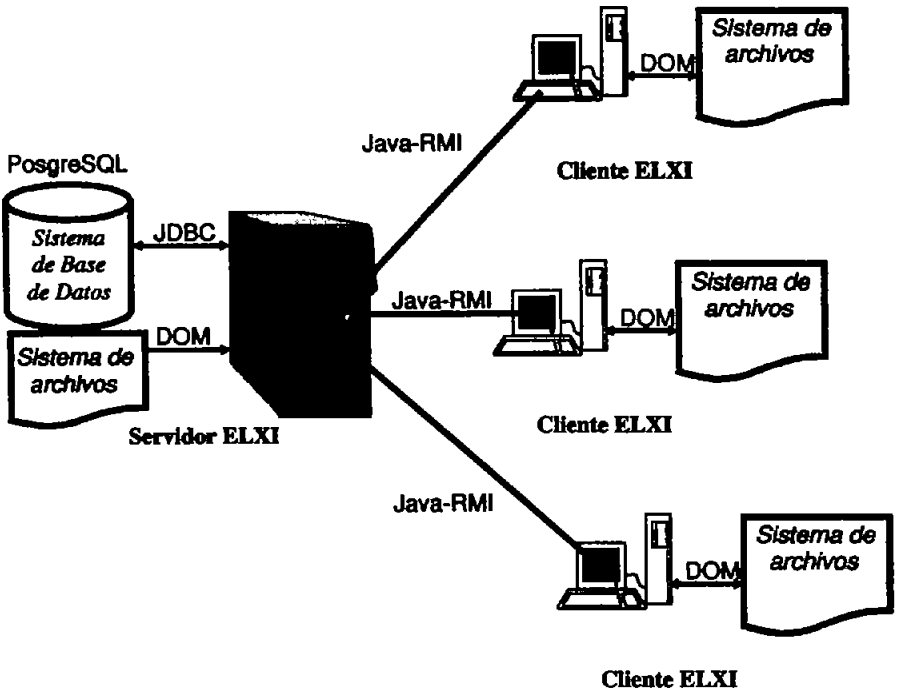


Figura 3.1: Arquitectura cliente-servidor del sistema ELXI

[1]Del acrónimo en inglés *Java Data Base Connectivity*. JDBC es una biblioteca de clases para la conexión de bases de datos con Java.

[2]Del acrónimo *Document Object Model*. DOM es una estructura jerárquica en forma de árbol, donde cada nodo contiene componentes de una estructura XML. El uso de las bibliotecas de XALAN permite crear nodos, remover nodos, cambiar su contenido, y navegar a través de la estructura de árbol.

La base de datos mantiene los registros relacionados con los autores, contraseñas de acceso, grupos de trabajo, documentos, estado de los fragmentos y autores conectados. **ELXI** utiliza *PostgreSQL 7.0* como base de datos.

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 y se logra un acceso más rápido a los documentos.

El servidor **ELXI** es el corazón de la administración de los documentos. Está implementado como un servidor **JAVA-RMI** que exporta su interfaz con más de treinta métodos para la administración y la utilización del sistema de edición. Tiene a su cargo varias tareas, de éstas las más importantes son:

- 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
- 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

El cliente **ELXI** consta de más de veinte clases escritas en lenguaje de programación Java. Las funciones de estas clases son:

- **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.
- **Ofrecer los servicios de conexión**, con el fin de que el cliente pueda contactar al servidor a través de llamadas a procedimientos remotos.

- **Manejar y administrar los documentos locales y la comunicación con el sistema de archivos local**, con el fin de mantener documentos almacenados y actualizados en la máquina que contiene el cliente **ELXI**.

- **Exportar su interfaz al servidor** para que pueda ser contactado a través de *callbacks*[3] JAVA-RMI.

- **Crear y destruir los hilos necesarios** para la administración local y distribuida de los documentos.

En la figura 3.2 se puede observar la interfaz del cliente de **ELXI**, la cual contiene el editor de documentos local, el árbol de documentos, la consola de información, el escritorio de trabajo, y el menú desplegable.

3.1.1. Características

Las características de **ELXI** se pueden describir de la siguiente forma:

- **Acceso compartido a un mismo documento por múltiples autores dispersos.** Un documento puede ser consultado y/o editado simultáneamente por un conjunto de autores.

- **Administración de autores y grupos de autores.** Un administrador define una lista de autores y puede formar grupos de trabajo. La colaboración se lleva a cabo únicamente entre los autores que sean miembros de un grupo de trabajo. Todo autor, siendo miembro de algún grupo de trabajo, puede ser el administrador de sus propios documentos y puede invitar a nuevos autores a colaborar sobre sus documentos.

- **Control de acceso al sistema colaborativo a través de contraseña.** Únicamente los autores autorizados pueden participar en el trabajo colaborativo. Por lo que, asociada a la definición de autores, un administrador puede también establecer a cada autor una contraseña para proteger el acceso al sistema colaborativo.

[3] Ver subsección 2.3.2

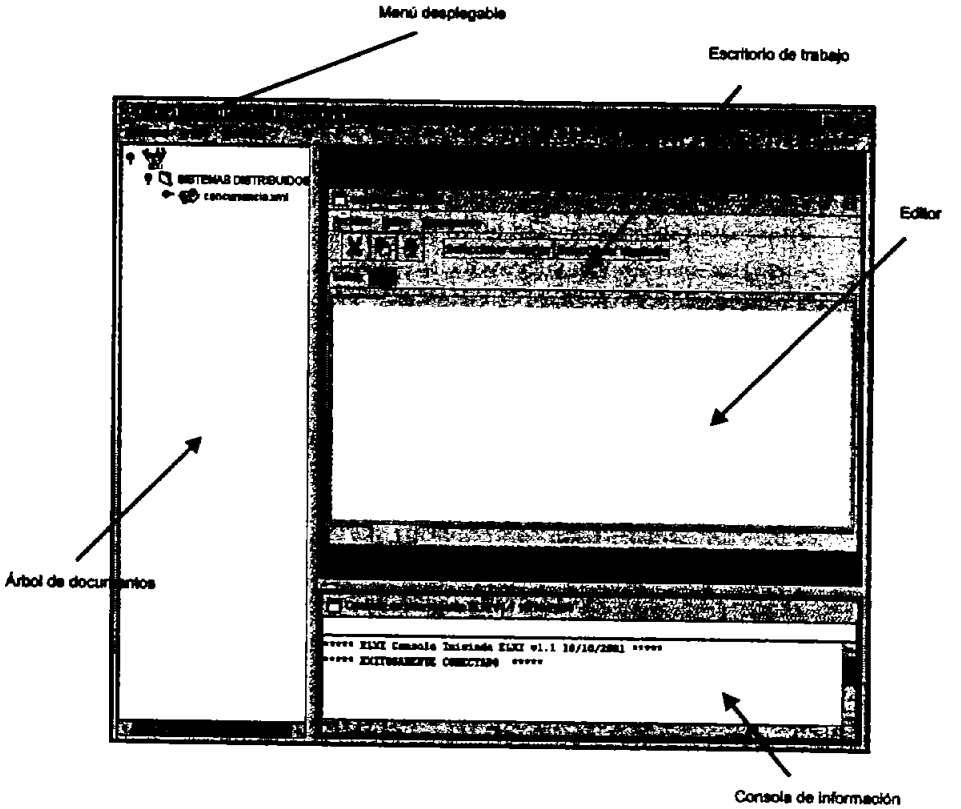


Figura 3.2: Interfaz del cliente ELXI

- **Compartición de un documento a través de su división en diferentes partes (fragmentos).** Siguiendo la estructura inherente de un documento, éste puede dividirse en diferentes partes o fragmentos. Un fragmento es la unidad mínima de colaboración y puede ser cualquier elemento dentro de la estructura de un documento, es decir, una sección, una subsección, un párrafo, una lista, etc.

- **Control de acceso a un documento a través de roles de edición.** Una vez definidos los autores y los grupos de trabajo, para colaborar sobre un documento, el administrador es el encargado de atribuir los roles de edición que jugarán cada uno de los autores. Los roles de edición son

los siguientes: lector, escritor y el propio administrador. Los primeros dos roles, como su nombre lo indica, permiten a un autor leer y escribir sobre los diferentes fragmentos de un documento. El administrador es quien orquesta y dá seguimiento a los pasos específicos y evolución de un proyecto documental, por lo que, además de poder modificar los roles de los autores, puede también agregar nuevos fragmentos a un documento o destruirlos. De igual modo, puede crear nuevos documentos o destruirlos. De ser necesario, un administrador puede delegar su rol de administrador a otro(s) autor(es). Tomando en cuenta la evolución de un documento, el cambio de roles de los autores se realiza de manera dinámica.

- **Interacción con un documento al mismo tiempo y en tiempos diferidos.** Sin importar el momento en que los autores participen en la elaboración de un documento, ni desde el lugar en donde lo hagan, cualquier autor conectado a la red puede tener acceso al documento completo.

- **Transmisión de mensajes instantáneos entre autores y de un autor a los miembros del grupo de trabajo.** Los autores pueden enviar y recibir mensajes entre ellos (invitaciones para colaborar, comentarios, etc.) y también lo pueden hacer a todos los miembros de un grupo de trabajo específico.

- **Producción (creación y visualización) de documentos XML.** El núcleo de edición permite generar documentos estructurados en el formato industrial y universal XML. Dicho núcleo integra las funcionalidades mínimas de edición necesarias para producir documentación técnica y/o científica. Asimismo, se utilizan plantillas de estilo con el fin de poder visualizar y dar presentación a los documentos. De esta manera, los autores no tienen que concentrarse en el formateo y presentación de los documentos, sino que únicamente en la producción del mismo.

- **Exportación de documentos a formatos PDF y HTML.** Los documentos generados pueden ser exportados, en todo momento, a formatos de documentos estándares y comerciales con el fin de poder ser visualizados, impresos o importados por otras aplicaciones. De esta

manera, se mantiene la compatibilidad total con los demás formatos de documentos de uso común por una gran cantidad de usuários.

- **Control de la conciencia de grupo.** A través del sistema colaborativo, todos los autores mantienen un amplio conocimiento de las acciones relativas al trabajo de edición de sus demás colegas. Saben, en todo momento, si alguien está conectado y en actividad con el sistema. Además, están actualizados sobre cada una de las acciones (modificaciones, cambios de roles, nuevos documentos, nuevos fragmentos, etc) de todos los participantes. La conciencia de grupo se maneja a través de un protocolo especializado y apoyado por un sistema de notificaciones en tiempo real.

- **Garantía de la consistencia del documento.** Cualquier acción de edición (escritura, modificación, actualización, etc.) en un documento es consistente con el trabajo de edición de todos los demás. Para ello, se establece una política que garantiza que sólo un autor puede ejercer el rol de escritor en un fragmento a la vez.

- **Implementación de la arquitectura de cómputo distribuido basada en el modelo cliente-servidor.** El software colaborativo se implementa siguiendo un modelo cliente-servidor, con lo cual se manejan eficientemente los diferentes eventos que se suscitan durante la colaboración.

- **Compatibilidad total con Windows y/o UNIX (ej. Linux).** Los clientes pueden ejecutarse en cualquier infraestructura de cómputo disponible, operando con los sistemas operativos Windows (en cualquiera de sus versiones) y/o UNIX (o compatibles).

- **Colaboración a través de Internet y/o Intranet.** Los autores, conectados a Internet y/o a una red corporativa (Intranet), pueden mantenerse en comunicación, colaborar y coordinarse, vía el documento compartido, desde distintas localidades que pueden estar apartados unos de otros en sitios geográficamente distantes en México y/o en el mundo.

El motor de edición está conformado por un editor de documentos estructurados basado en el lenguaje XML[4]. El lenguaje XML constituye hoy en día

[4]Del acrónimo en inglés *eXtended Markup Language*

un estándar industrial y un formato universal de documentación electrónica. Con el fin de poder visualizar los documentos XML y poder convertirlos a otros formatos, en la aplicación de trabajo colaborativo **ELXI** se utilizaron otras tecnologías asociadas a XML, tales como: XSL[5], XSLT[6] y XSL-FO[7].

Para explicar mejor la arquitectura de **ELXI** podemos dividir al sistema en tres grandes partes basadas en su implementación.

- Mecanismos de control de concurrencia en **ELXI**
- Mecanismos para el control de la consistencia en **ELXI**
- Estructura de documentos XML en **ELXI**

3.1.2. Mecanismos de control de concurrencia en ELXI

El mecanismo de control de concurrencia que se ha implementado en **ELXI** consiste de tres partes: 1) Estructuración del documento; 2) Asignación de responsabilidades a los autores del sistema; y 3) Control del proceso de escritura sobre la unidad de concurrencia.

1. Estructuración del documento

En **ELXI** se considera que los autores deben tener la posibilidad de trabajar en diferentes partes del documento y además cada una de esas partes deben ser almacenadas y actualizadas, tanto localmente como a distancia sin interrumpir el trabajo de cada autor. Esta forma de trabajo conduce al hecho de dividir el documento en unidades que llamaremos **fragmentos**. En la figura 3.3 se puede apreciar, enmarcado en líneas, un fragmento dentro de un documento en el cliente **ELXI**.

En el sistema **ELXI** son los propios autores quienes definen los parámetros de concurrencia y granularidad. Para ello, dividen el documento en fragmentos. Dichos fragmentos son de granularidad variable (líneas, párrafos, secciones, capítulos, etc), y constituyen **la estructura del documento**.

[5]Del acrónimo en inglés *eXtensible Stylesheet Language*
[6]Del acrónimo en inglés *eXtensible Stylesheet Language Tranformations*
[7]Del acrónimo en inglés *XSL Formatting Objects*

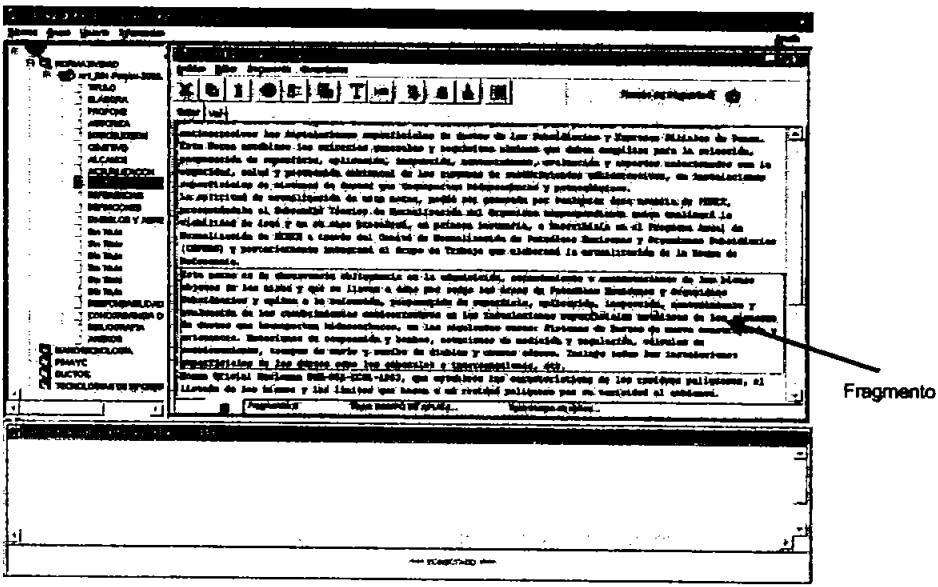


Figura 3.3: Un fragmento en el editor de ELXI

Una vez definida la estructura del documento, cada autor puede editar simultáneamente sobre cada uno los diferentes fragmentos. Sin embargo, con el fin de evitar que dos o más autores editen al mismo tiempo el mismo fragmento, en **ELXI** se definió una asignación de responsabilidades a los autores del sistema.

2. Asignación de responsabilidades a los autores del sistema

La asignación de responsabilidades a los autores del sistema es propuesta para su administración a través de permisos de acceso (roles) atribuidos a cada autor. Si todos los autores tuvieran acceso a todos los documentos, sin ninguna restricción, entonces cualquier autor podría obtener una copia del mismo, aunque realmente no hubiera interés de su parte o incluso el compromiso de trabajar en ellos. La asignación de responsabilidades condujo a la implementación de grupos de trabajo [2], a través de los cuales se invita a participar a los autores que se desea estén involucrados en algún proyecto de

edición. De esta manera, los documentos que se desarrollan en un grupo de trabajo sólo podrán ser accesados por los miembros de ese grupo, limitando así el acceso a otros autores de otros grupos. Asimismo, existe la posibilidad de que un autor con permiso especial en el grupo pueda restringir que otros miembros del mismo grupo de trabajo tengan acceso a uno o varios de los documentos del grupo.

De esta manera, un autor puede accesar a un documento siempre y cuando sea miembro del grupo al que pertenece el documento y además cuente con el permiso correspondiente para visualizarlo o editarlo. Los permisos que puede tener un autor sobre un documento son implementados a través de roles.

Los roles son asignados de acuerdo al tipo de trabajo que realizarán los autores a) en el documento y b) en los fragmentos.

Para a), se han propuesto dos roles de autores en el documento:

1. **Administrador de documento.** Con este rol, un autor puede recuperar el documento para su consulta y/o edición; agregar fragmentos al documento; así como asignar y/o cambiar roles a los autores en el documento.

2. **Autor del documento.** Con este rol, un autor del documento puede recuperar el documento para su consulta y/o edición; agregar fragmentos al documento; o incluso quedarse sin ningún permiso para accesarlo.

Para b) se han propuesto tres roles de autores en los fragmentos:

1. **Administrador de fragmento.** Con este rol, un autor puede asignar roles, escribir y leer en uno o más fragmentos.

2. **Escritor de fragmento.** Con este rol, un autor puede escribir y leer en uno o más fragmentos.

3. **Lector de fragmento.** Con este rol, un autor solo puede consultar uno o más fragmentos.

En la figura 3.4 se observa el menú que permite asignar permisos a un fragmento de un documento en **ELXI**. A través de botones de opción, se asignan o se quitan permisos a usuários del fragmento.

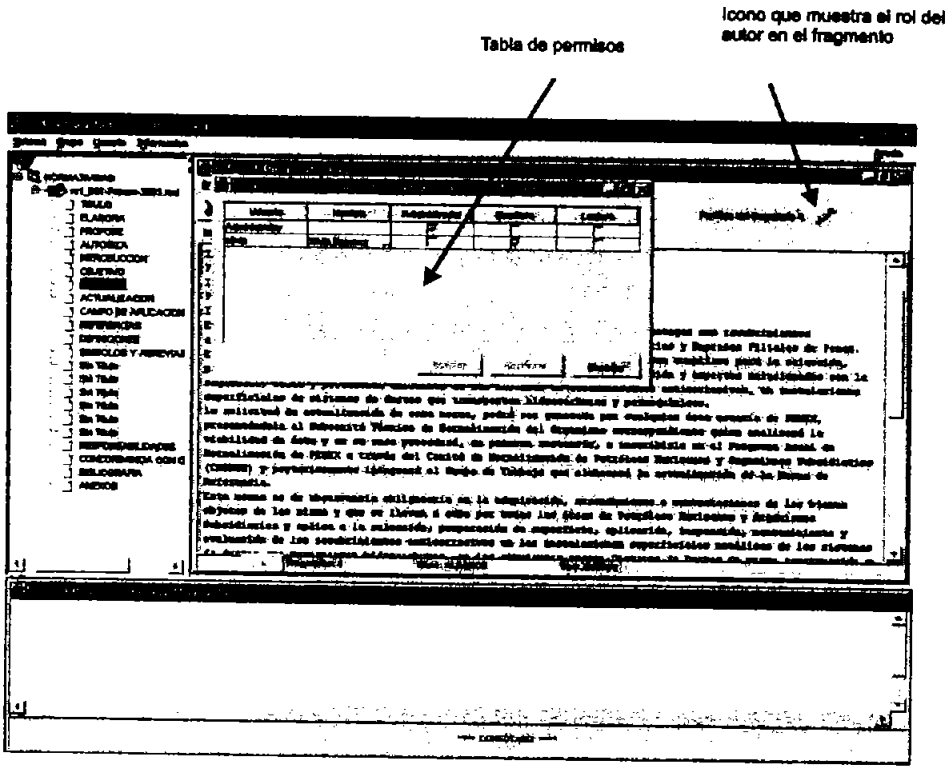


Figura 3.4: Tabla de visualización de permisos de un fragmento

Con estos roles de autores en los fragmentos, básicamente se permite o se niega el acceso en escritura a un fragmento. Sin embargo, dado que dos o más autores pueden contar con el rol de escritor o administrador sobre el mismo fragmento, aún persiste el problema de que dos o más autores podrían desear escribir sobre el mismo fragmento al mismo tiempo. Para resolver este problema, se implementó un mecanismo para el control de la escritura sobre un fragmento, el cual es considerado como una unidad de concurrencia.

3. Control de escritura sobre una unidad de concurrencia

El control de escritura sobre un fragmento se implementa a través de un mecanismo de bloqueo, el cual permite controlar el acceso (en escritura) a cada uno de los fragmentos. De esta manera, diferentes autores con el rol de escritor o administrador, pueden en todo momento, solicitar el bloqueo de un fragmento.

Aquella solicitud que haya llegado primero al servidor **ELXI** será atendida. Las demás se formarán en una cola y seguirán una política FIFO. Por tanto, este mecanismo garantiza que, en un instante dado, sólo un autor pueda bloquear un fragmento en escritura y los demás sólo puedan leerlo. Se propuso un mecanismo de bloqueo que dura sólo un intervalo de tiempo para cada autor. De esta manera, el escritor o administrador son los que definen el tiempo de bloqueo y no hay restricción alguna de volver a solicitar el bloqueo cuantas veces desee.

En las figuras 3.5 y 3.6 se puede observar la edición de dos fragmentos de forma síncrona. En la primera un usuario bloquea un fragmento para su edición. En la segunda figura otro usuario comparte el mismo documento y edita otro fragmento. Para el primer usuario, el fragmento que edita el segundo estará bloqueado.

Mientras un fragmento está siendo editado, este fragmento aparece, en el árbol de navegación, en color verde, mientras los fragmentos que están siendo editados por otros autores se muestran en color rojo.

Así, como se observa, el sistema de concurrencia, define una estructura que permite el manejo correcto de documentos en **ELXI**.

3.1.3. Mecanismos para el control de la consistencia en ELXI

Para lograr mantener la consistencia de sus documentos, **ELXI** se basa principalmente en tres puntos:

1. Control de acceso a los documentos.

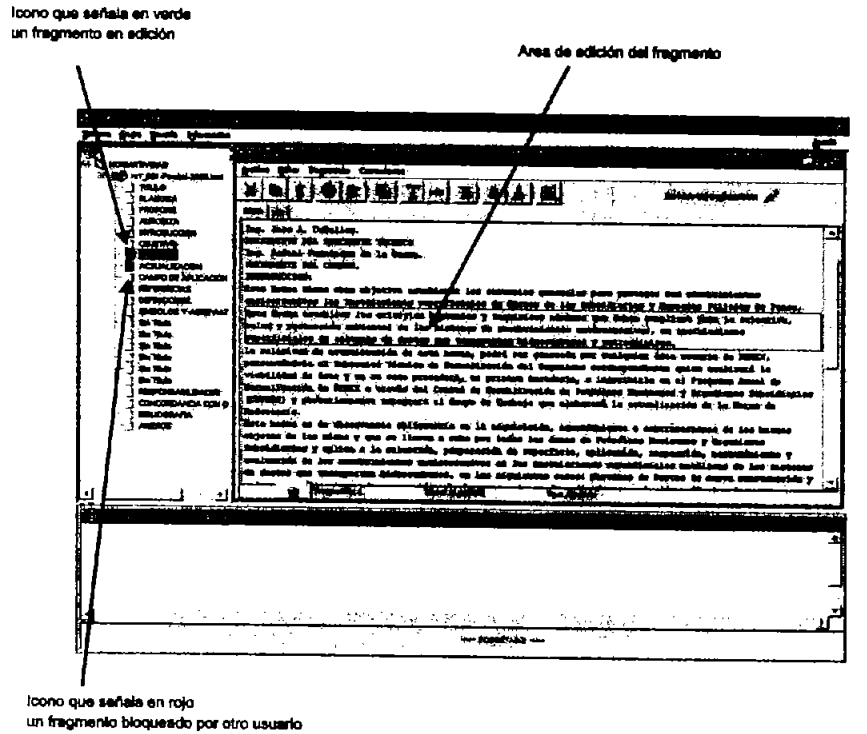


Figura 3.5: Edición y bloqueo de un fragmento por un usuario

2. Manejo de réplicas.

3. Vistas de usuários.

1. Control de acceso a los documentos

Como ya se mencionó, los documentos en el sistema **ELXI** están divididos en fragmentos para permitir el trabajo concurrente de varios usuários. El diseño de **ELXI** está interrelacionado en cada una de sus partes, por lo cual, muchos de sus aspectos resuelven otros a causa de haber sido analizados y diseñados de manera conjunta desde un inicio.

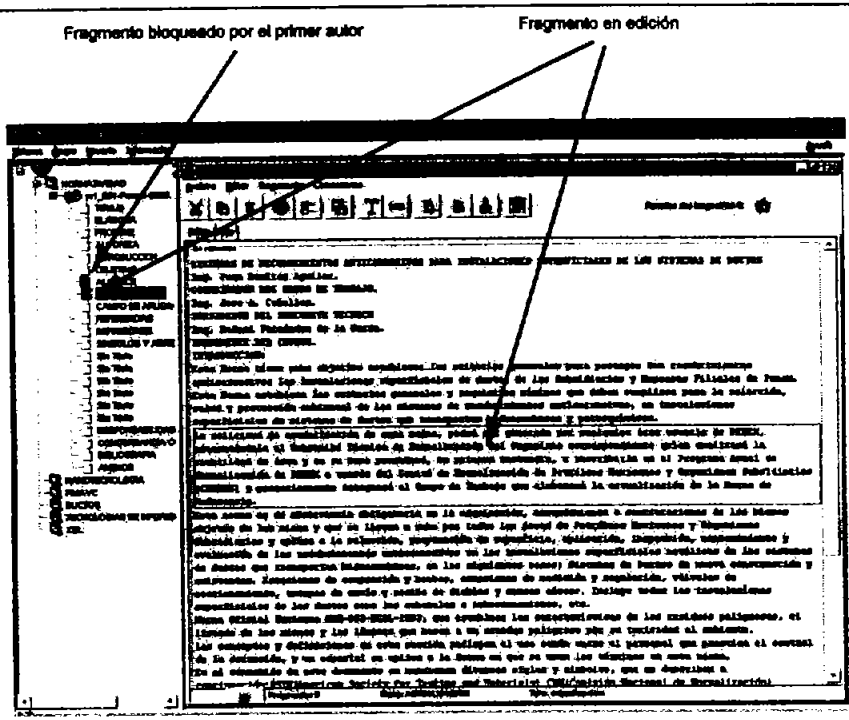


Figura 3.6: Segundo usuario compartiendo el mismo documento

Por ejemplo, el uso de bloqueos de escritura, antes mencionado, sobre los fragmentos permite a **ELXI** resolver la relación de consistencia de los documentos que se están editando. En la figura 3.7 se muestra el menú con el que se solicita la edición de un fragmento. Esta solicitud se puede hacer en horas o minutos.

El mecanismo de control de concurrencia que se implementó en el sistema **ELXI** consiste de tres partes: i) Estructuración del documento; ii) Asignación de responsabilidades a los autores del sistema; y iii) Control del proceso de escritura sobre la unidad de concurrencia.

Para la *estructuración del documento* en **ELXI**, los autores deben tener la posibilidad de trabajar en diferentes partes del documento y además cada

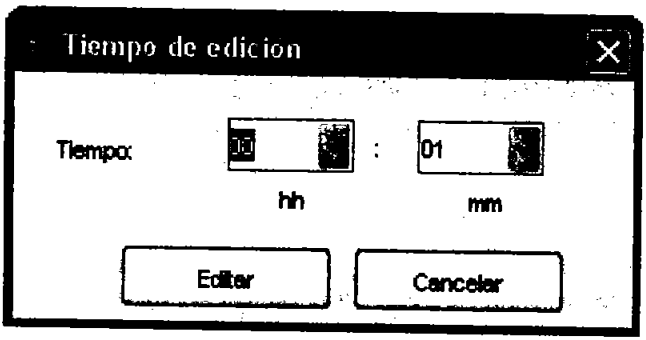


Figura 3.7: Menú para petición de tiempo de edición

una de esas partes deben ser almacenadas y actualizadas, tanto localmente como remotamente sin interrumpir el trabajo de cada autor. Esta forma de trabajo condujo al hecho de dividir el documento en **ELXI** en unidades llamadas fragmentos.

La *asignación de responsabilidades a los autores del sistema* se administra en **ELXI** a través de permisos de acceso (roles) atribuidos a cada autor por medio de de grupos de trabajo [23]. A través de estos grupos, se invita a participar a los autores que se desea estén involucrados en el proyecto de edición. De esta manera, los documentos que se desarrollan en un grupo de trabajo sólo podrán ser accesados por los miembros de ese grupo, limitando así el acceso a otros autores de otros grupos.

El *control de escritura sobre una unidad de concurrencia* (fragmento) se implementa en **ELXI** a través de un mecanismo de bloqueo, el cual permite controlar el acceso (en escritura) a cada uno de los fragmentos. Aquella solicitud que haya llegado primero al servidor **ELXI** será atendida. Las demás se formarán en una cola y seguirán una política *FIFO* [8]. Por tanto, este mecanismo garantiza que, en un instante dado, sólo un autor pueda bloquear un fragmento en escritura y los demás sólo lo podrán leer. El mecanismo de bloqueo dura sólo un intervalo de tiempo para cada autor. El autor mismo es quien define este tiempo de bloqueo y no hay restricción alguna de volver a solicitar el bloqueo cuantas veces desee.

[8]Del acrónimo en inglés *First Input First Output*: Primero en entrar primero en salir

Así, el control de acceso garantiza, que no existan dos escritores activos sobre un mismo fragmento, con lo que se logra que los documentos en **ELXI** sigan siendo consistentes.

2. Manejo de réplicas

ELXI mantiene una versión actualizada de cada uno de los documentos en el servidor y réplicas en los clientes que solicitan el documento del servidor.

La replicación de datos en **ELXI** persigue tres objetivos principales: eficiencia, disponibilidad y/o tolerancia a fallas. Para **ELXI**, se implementó la replicación como una manera de aumentar la eficiencia. Cada uno de los usuários mantiene réplicas de los documentos en su máquina local, de tal manera que el acceso al documento no es a distancia, sino local. Lo cual hace más eficiente el proceso; aunque este aumento en la eficiencia tiene un costo, el cual está relacionado con la consistencia mencionada anteriormente. Es preciso garantizar que las réplicas mantenidas en las diferentes máquinas sean consistentes entre sí.

En **ELXI** las actualizaciones de los documentos son difundidas a los demás clientes en fragmentos y no en documentos completos. Lo anterior permite reducir el tráfico en la red, es decir, envíando los fragmentos actualizados.

Ahora, estos fragmentos y documentos se transfieren de un sitio a otro usando *métodos remotos*[9] lo que lleva a una estructura de datos. Esta estructura consiste de objetos de tipo *fragmentos*, *grupos* y *archivos* los cuales definen la estructura de cada uno de los documentos manejados en **ELXI**. En la figura 3.8 se observa la estructura mencionada en forma de árbol: en la raíz se encuentra la información del usuario; la estructura continua con los grupos de usuários; los archivos dentro de los grupos son mostrados como ramas de los grupos; y por último se muestran los fragmentos contenidos en los archivos.

Por tanto, **ELXI** permite replicar documentos en diferentes clientes a través de su estructura de documentos definida, viajando ésta por medio de llamadas a *métodos remotos*.

[9]Ver sección 2.3.2 página 24. Java RMI: Remote Method Invocation.

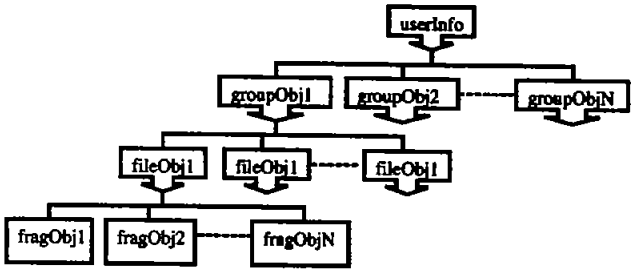


Figura 3.8: Estructura de datos en los documentos de ELXI

Cabe mencionar, que las rutas para el almacenamiento de los documentos en el servidor, esta dada por directorios definidos. De igual forma los documentos replicados en los clientes también tienen asignados directorios específicos. La ruta para los documentos en el cliente es: *dirInstalacionElxi/datos/grupodeTrabajo* y en el servidor: *dirInstalacionElxi/datos/usuario/grupodeTrabajo.*

3. Vistas de usuários

El sistema mantiene las vistas de usuario consistentes, es decir, para dos o más usuários diferentes que estén trabajando sobre un mismo documento o dentro de un mismo grupo, los eventos que se generen en sus respectivas interfaces gráficas, deben ser consistentes con la información real de los documentos y de los grupos en el servidor **ELXI** y también entre ellos.

En la figura 3.9 se puede observar la representación de un usuario M y un usuario F. El usuario M al conectarse al sistema, envía su interfaz IM y se registra en el servidor **ELXI** dentro del grupo G1. El servidor almacena en una estructura de datos, compuesta por tablas de relación anidadas, las interfaces de los usuários.

En la figura 3.10 se observa que cuando el usuario F envía una actualización al servidor, éste obtiene todas las interfaces de los usuários involucrados

con el cambio de la estructura de datos (interfaz de M) y los contacta a través de una llamada para enviarles la nueva actualización.

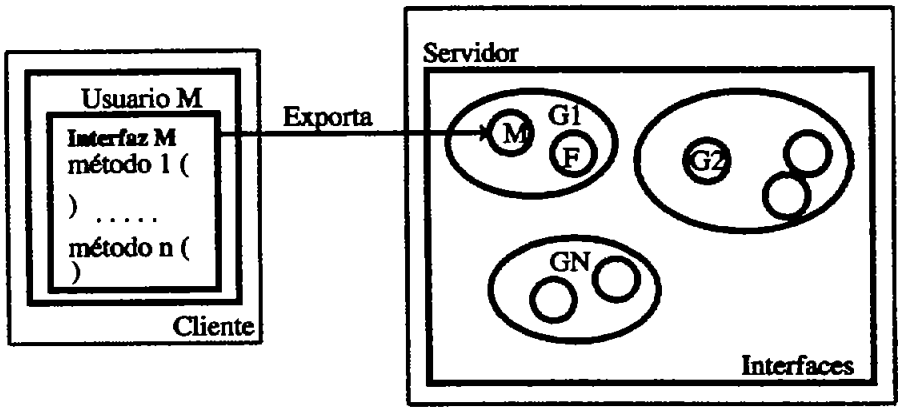


Figura 3.9: Exportación de interfaces

De esta manera, se logra que todos los usuários que se encuentren conectados al sistema tengan vistas consistentes en sus máquinas locales, además de que estén informados de la participación de sus colegas en el trabajo.

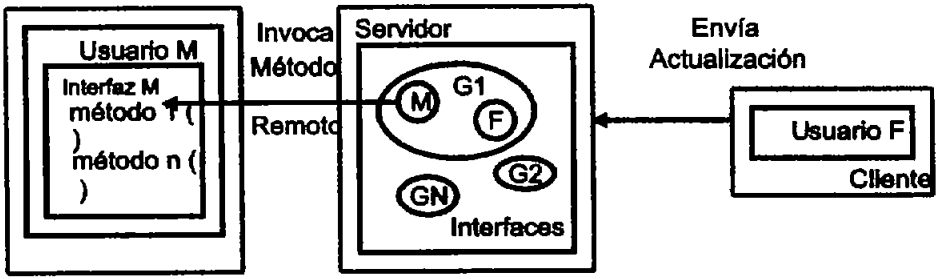


Figura 3.10: Invocacion de la interfaz de algún cliente por el servidor

Así, las vistas de usuario permiten mantener actualizadas las versiones de los clientes con respecto a la version ubicada en el servidor, manteniendo la consistencia de los documentos.

3.1.4. Edición de documentos XML en ELXI

En esta parte, **ELXI** cuenta con un área de edición, para generar documentos *estructurados*[10]. Esta área en **ELXI** permite la edición de documentos con formato XML, además de permitir su visualización desde la aplicación misma o su exportación a 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, etc.

El documento en **ELXI** se divide en varios fragmentos visuales, enmarcados por líneas rojas, donde cada uno tiene permisos y atributos que permiten su control de forma autónoma. Así, el uso del formato XML como medio de control para los fragmentos de los documentos de **ELXI** es evidente, ya que este formato permite tener una descripción detallada y estructurada del documento. Los fragmentos visuales son nodos XML, así cada nodo puede ser controlado como una entidad única que contiene información adicional que sirve para: definir permisos al documento; definir un *tipo* útil para su visualización; definir un estado para el fragmento, el cual informa si es de lectura, escritura o bloqueado; un identificador único para diferenciarlo de todos los demás nodos o fragmentos.

El tipo sirve para definir posteriormente como será visualizado ese fragmento. Existe un tipo finito de tipos y todos están predefinidos. Dependiendo del tipo asignado será la forma en que será visualizado el fragmento. Este tipo servirá para asignar atributos de color, fondo y posición.

El estado define si el documento puede ser editado o está bloqueado debido a que otro autor esta escribiendo sobre él. Si el autor observa que el estado permite editar el fragmento, hace una solicitud de bloqueo del mismo. Si es aceptada su petición el fragmento cambia su estado a bloqueado para los demás autores. El editor mantiene de manera visible, para el usuario, un control de todos los posibles estados del fragmento.

[10]Ver sección 2.4.1 pag. 29. Metadatos y documentos estructurados

El estado de lectura indica que el fragmento solo puede ser leído. El de escritura muestra que el fragmento puede ser modificado o actualizado y el estado de bloqueado indica que el fragmento está siendo editado por algún otro autor y por tanto no puede editarse ni modificarse hasta que cambie su estado.

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 o con otros nodos dentro de algún otro documento.

Los componentes que constituyen el editor son básicamente: a) la definición del tipo de documento o DTD[11], b) un documento XML[12] con la estructura básica mínima de un documento, c) plantillas XSL [13] para permitir la transformación de documentos a los diferentes formatos, d) clases que permiten la edición y visualización de los documentos XML creados.

La definición de tipos *DTD* del documento, en la versión 1.0 de **ELXI**, se presenta a continuación:

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

<!ELEMENT documento (fragmento+)> <!ATTLIST documento
          id_file CDATA #REQUIRED
          tipo_f CDATA "">

<!ELEMENT fragmento (#PCDATA)*> <!ATTLIST fragmento
          titulo CDATA #REQUIRED
          id_fragment CDATA #REQUIRED
          tipo CDATA "">
```

El editor contiene también los componentes que están involucrados con el procesamiento de los documentos XML y los procesos de transformación para la visualización. Para el procesamiento y transformación de los documentos se siguen procesos más específicos. De estos, el procesamiento del documento XML es el que permite la administración de los documentos.

[11] Ver subsección 2.4.5 pagina 37. *XML* Lenguaje de marcas extendido
[12] Ibidem
[13] Ver subsección 2.4.6 página 38. Normalización de estilo en XML

Procesamiento del documento XML

ELXI analiza sintácticamente el documento XML como primer paso a la hora de procesarlo. Si un documento no está bien formado, el analizador aborta marcando un error de construcción. En **ELXI** hay una DTD y el analizador valida según éste.

Para el editor, se necesita tener en todo momento el conocimiento de cuáles son los elementos de los documentos para poder modificarlos en un momento dado. **ELXI** hace uso de la interfaz DOM de Xerces [45] para procesar el documento.

Transformación del documento XML

El motor de transformación XSLT-XALAN [44] realiza la operación de transformación tomando el documento XML como entrada, junto con la plantilla de transformación XSLT, para crear el documento en formato **HTML**. La especificación XSLT es usada para transformar documentos XML al formato **HTML**. El editor permite hacer una previsualización del documento en formato **HTML** luego de la transformación (sin necesariamente tener que exportar el documento a un formato deseado).

En el lenguaje Java sólo existe un componente (la clase JTextPane) capaz de visualizar el formato **HTML**. El JTextPane sólo es capaz de visualizar documentos en **HTML** versión 3.2 y CSS versión 1.0. Esto limita las posibilidades de formato en **ELXI**, pero dado que una gran mayoría de programas interpretan el lenguaje **HTML**, se determinó que era lo suficientemente estándar para usarlo como base.

Una plantilla XSL-FO[14] es un archivo con código para manejar marcas (*tags*[15]) relacionadas a colores, tamaños y posiciones. La plantilla 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 lenguaje ayuda a crear una hoja de estilo, que después podrá ser interpretada por algún convertidor de estilo. El convertidor interpreta los objetos de formateo, para transformar el archivo XML a otro formato conocido (**PDF**, **PS**, **TXT**, etc.).

[14] Objetos de formateo, ver subsección 2.4.6 pag. 38
[15] Del inglés *tag*: marca, rótulo, marbete.

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 fácilmente. En **ELXI** se usa el procesador Java Apache FOP [46], por su fácil integración al analizador XALAN.

En **ELXI** existen varias plantillas XSL-FO que permiten generar formatos distintos a los documentos XML. Estas plantillas se usan exclusivamente para transformar el documento XML al formato **PDF**.

El editor

El editor en **ELXI** tiene 4 componentes principales: el árbol de navegación, la ventana de edición, los menús del editor y la consola de información (ver figura 3.11).

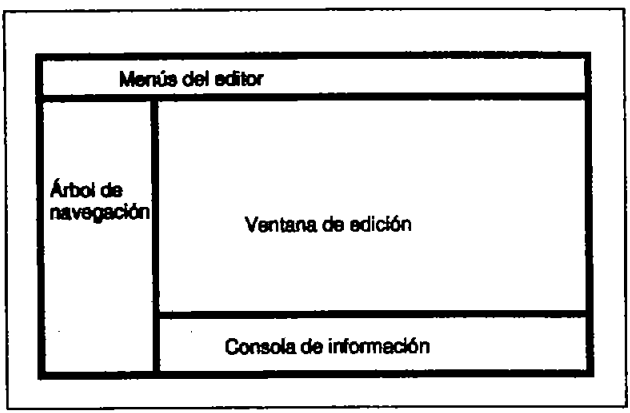


Figura 3.11: Interfaz del editor en ELXI

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 genera con la información que el servidor **ELXI** le envía (ver figura 3.12).

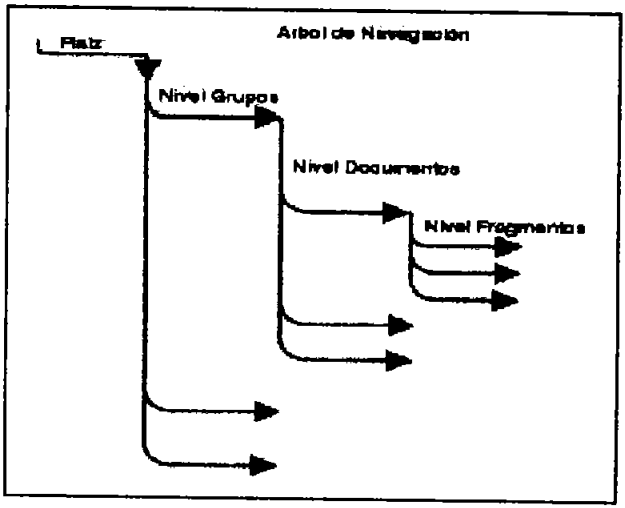


Figura 3.12: Estructura del árbol de navegación

En la ventana de edición se realizan las operaciones de edición y visualización de los documentos. Dicha ventana aparece del lado derecho del área de trabajo del editor. El usuario abre o crea un documento y éste aparece en la ventana de edición dentro de la pestaña *edición*. Una vez abierto el documento, el usuario puede visualizarlo dando un click en la pestaña de *visualización (ver)*.

La pestaña de visualización permite mostrar el documento que el usuario está editando. De esta manera, puede hacer una previsualización del documento. Dentro del panel de visualización, se puede observar cual será el formato del documento para su posterior exportación a **HTML** [16] o **PDF**. El panel en ese momento permite escoger el tipo de hoja de estilo para generar distintos tipos de salida y formato. Cuando se está dentro de la pestaña de visualización, el área de edición se convierte en el área de visualización —ver fig 3.13—.

[16]En los capítulos referentes al diseño e implementación, se observará que el presente trabajo agrega el formato RTF; RTF puede ser visualizado en MSWord [47] o en otros editores comerciales

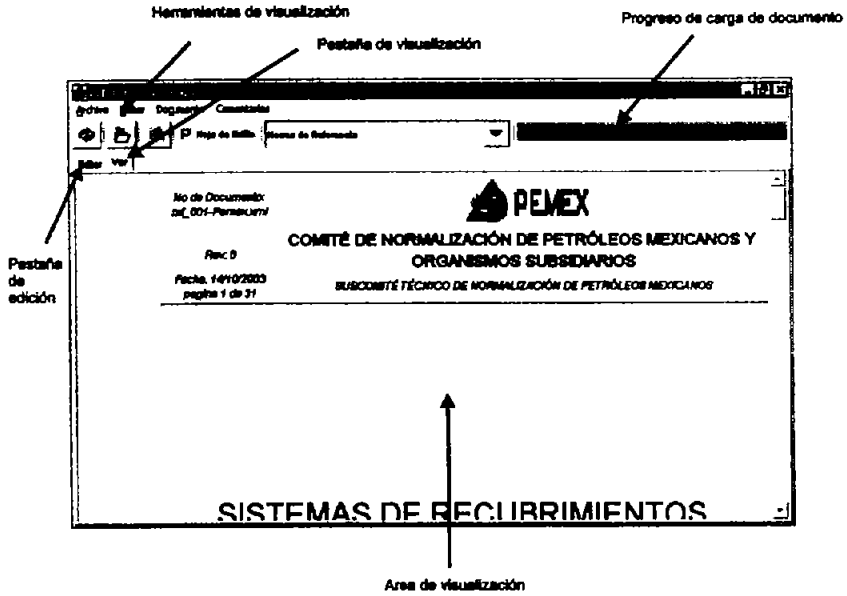


Figura 3.13: Área de visualización en ELXI

Así, el editor ofrece una interfaz de usuario cómoda y es una de la partes esenciales de **ELXI**.

3.2. Evaluación del estado del sistema y planteamiento de requerimientos

La presentación de documentos, mostrada en la sección anterior, es tomada como punto de partida para el análisis, diseño e implementación de módulos tratados en este trabajo. Esto con el fin de incrementar las características relativas a la visualización de documentos en **ELXI**.

Los documentos en **ELXI** están, como se mencionó, formados de fragmentos, los cuales son las unidades colaborativas del documento completo.

Los documentos logran esta estructura a través de una DTD específica, la cual permite definir el *modelo de contenido* de los documentos y los tipos de datos. La DTD que define los documentos, en la versión 1.0 de **ELXI**, presenta en general dos tipos importantes en la estructura del documento:

- Documento
- Fragmento

Con tal estructura definida, **ELXI** genera documentos en los cuales el contenido de los fragmentos es un bloque de texto sin párrafos.

Una de las características deseadas en **ELXI** es la capacidad de manejar una mayor cantidad de elementos, a diferencia del bloque de caracteres por fragmento que **ELXI** maneja. Esto es, hasta la definición anterior de la DTD mencionada, cada fragmento sólo puede contener uno y solo un bloque de texto por fragmento. Así, un fragmento debe contener más de un bloque de texto para poder representar diferentes párrafos, dando con esto una mayor capacidad de edición y comprensión de cualquier documento. Por tanto, un fragmento debe ser capaz de contener más de un párrafo dentro del mismo.

En la figura 3.14 se observa una **Norma** en formato **PDF**, este documento se editó en **ELXI** en las primeras fases de este trabajo. Se implementó la primera fase de agregación de imágenes y se obtuvo un buen formato, pero aún se tenía un solo bloque de texto por fragmento.

En la definición del *modelo de contenido*, definido por la DTD, se declara solo un tipo de elemento llamado fragmento. Es deseable, entonces, poder definir otros tipos que permitan un conjunto de edición más rico con relación a elementos de edición disponibles. Estos elementos de edición se deberán definir dentro de la DTD existente, haciendo crecer éste para lograr documentos mas completos.

Por tanto, en este trabajo, el análisis, diseño e implementación de otros elementos de edición como listas, tablas, imágenes, etc., son los elementos propuestos que permitirán un contenido más rico en los documentos generados por **ELXI**. El objetivo es lograr un conjunto de elementos de edición

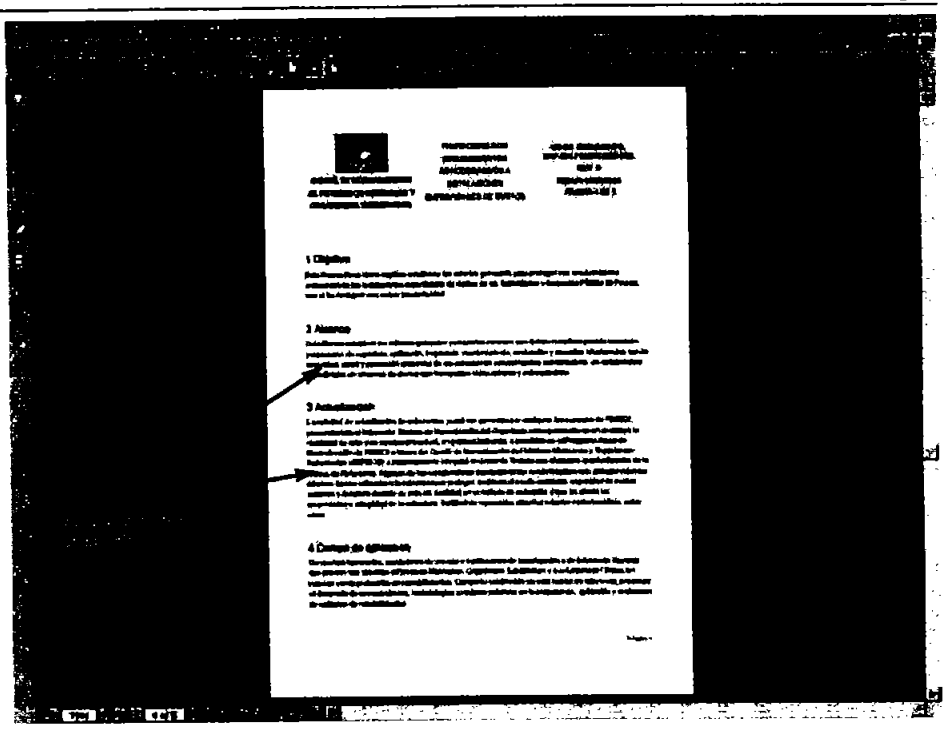


Figura 3.14: Documento con sólo un párrafo por fragmento

más completo, el cual a su vez permitirá definir estructuras de elementos complejas, logrando así que **ELXI** sea capaz de manejar documentos mucho más elaborados.

Con cada uno de estos elementos de edición definidos en el modelo de contenido, se puede tener un lenguaje de marcas capaz de implementar documentos específicos. Este trabajo se enfoca en permitir la edición y visualización de un documento de tipo **Norma**, el cual tiene un modelo específico dado por las mismas necesidades de edición de una *Norma de referencia de PEMEX*. Posteriormente se pretende lograr el diseño e implementación de un documento más complejo capaz de editar cualquier documento técnico o científico, contando para ese momento con el conocimiento obtenido de la implementación del primer documento.

El sentido y objetivo de una *Norma de Referencia de PEMEX* se basa en la magnitud y complejidad de las funciones que se realizan en Petróleos Mexicanos y sus Organismos Subsidiarios, las cuales requieren de disposiciones técnicas y administrativas, así como de la participación de multiples autores, para que la gestión institucional se dé en un marco de congruencia, orden, participación y corresponsabilidad. En este sentido, la Ley Federal sobre Metrología y Normalización, publicada en el Diario Oficial de la Federación el 1 de julio de 1992 y reformada en mayo de 1997, establece el objetivo de modernizar y adecuar el marco normativo vigente respecto a los procedimientos para la emisión de Normas Oficiales Mexicanas, Normas Mexicanas y Normas de Referencia, así como la comprobación de las mismas a fin de mejorar la calidad, seguridad y eficiencia de los procesos, bienes y servicios nacionales.

Por lo anterior, una **Norma** de referencia es un documento con una estructura bastante completa. La estructura de una **Norma** es la siguiente:

Objetivo Su redacción debe ser en infinitivo y señalar el propósito que persigue la aplicación y cumplimiento de la Norma de Referencia.

Alcance Se deben especificar el propósito de los temas contenidos en la norma y la profundidad con que los trata.

Campo de aplicación En esta sección se debe establecer los límites de aplicabilidad de la Norma, además de precisar que ésta es de aplicación general y observancia obligatoria en las adquisiciones, arrendamientos o contrataciones de los bienes o servicios objetos de la misma, que lleven a cabo las áreas de Petróleos Mexicanos y Organismos Subsidiarios.

Actualización En esta sección se deberá señalar el tiempo máximo (5 años) para efectuar las revisiones a la Norma de Referencia, las áreas facultadas para realizar las sugerencias de modificación o cancelación, así como el nombre del titular y domicilio del responsable del Subcomité Técnico de Normalización facultado para realizar los cambios sugeridos.

Referencias Se deben relacionar únicamente las Normas Oficiales Mexicanas, Normas Mexicanas, Normas Internacionales u otra Norma de Referencia que sea indispensable consultar para la aplicación de la Norma de Referencia.

Definiciones Se anotará el significado de los términos empleados dentro del contenido técnico de la Norma de Referencia y ordenarse en forma numérica y alfabética.

Símbolos y abreviaturas Se deben considerar aquellas abreviaturas que faciliten el manejo e interpretación de la Norma de Referencia.

Desarrollo En este capítulo se debe describir con el detalle suficiente las especificaciones, requisitos, condiciones, formatos y pruebas de aceptación requeridas por PEMEX y Organismos Subsidiarios para satisfacer la adquisición, compra o arrendamiento de bienes o servicios.

Responsabilidades En esta sección se deben indicar las obligaciones de cada una de las áreas de PEMEX, Organismos Subsidiarios, de organizaciones, proveedores, prestadores del servicio, etc., que estén involucradas en el cumplimiento de la Norma de Referencia.

Concordancia con otras Normas En su caso se debe indicar el grado de concordancia con otras normas mexicanas o regulaciones internacionales, señalando si la Norma de Referencia que se presenta es idéntica, equivalente o no equivalente.

Bibliografía En esta sección se deben de relacionar los documentos que se tomaron de base para la elaboración de la norma de referencia.

Anexos Se anotarán los números y títulos de los anexos generados en el apartado de desarrollo. En el caso de no presentar anexos, se anotará en este apartado *No aplica.*

Esta estructura fue implementada en **ELXI**, y se tomará como base para el desarrollo de este trabajo.

Toda la arquitectura de **ELXI** es útil para la edición del tipo de documento **Norma**. Sin embargo, varias características debieron ser tomadas en consideración para una correcta edición. Como se mencionó anteriormente, una **Norma** contiene varios párrafos, que si bien pueden ser implementados cada uno como un fragmento en **ELXI**, no se obtiene la estructura adecuada de una sección en una **Norma**. Una Norma está estructurada en partes especificas: objetivo, campo de aplicación, concordancia con otras normas, etc. Cada una de estas partes se definió como un fragmento con un tipo en

especifico, pero cada parte de la **Norma** está definido a su vez por varios párrafos. La anterior fue una de las necesidades por la cual se resolvió la implementación de varios párrafos dentro de un fragmento.

Además, una **Norma** de referencia en su edición necesita la inserción de otros elementos de edición, tales como: listas, tablas e imágenes. Cada uno de estos elementos fue necesario definirlo en el modelo de datos de un documento de **ELXI**. Así, para la edición de un documento **Norma** se necesitó redefinir los elementos posibles en **ELXI**.

Los elementos básicos propuestos son los siguientes:

- Párrafos
- Listas
- Tablas
- Imágenes
- Notas a pie de página
- Encabezados
- Pies de página
- Secciones y subsecciones
- Texto
- Texto sin formato
- Índice

ELXI permite distribuir los contenidos de los documentos en diferentes clientes, en lugares diferentes de forma síncrona. Si ahora se tienen además otros elementos de edición más complejos, la forma de distribuir el documento completo implicará varios cambios de funcionamiento, inherentes a estas modificaciones.

3.2.1. Estructura de la propuesta

De esta manera, surge la necesidad de poder distribuir junto con cada fragmento estos elementos de edición. Además, se deben poder manejar estos elementos de edición, tanto para el documento **Norma**, como para el documento **General**[17].

El desarrollo de esta propuesta se puede estructurar en la forma siguiente:

1. Problema de distribución de fragmentos: cada fragmento debe poder contener elementos de edición tales como listas, tablas, imágenes etc., los cuales a su vez, al estar contenidos en los fragmentos, deben poder ser distribuidos junto con el fragmento.
2. Visualización por medio de hojas de estilo *XSLT* y *XSLT-FO* para documentos de tipo **Norma** y documentos de tipo **General**.
3. Implementación de interfaces de usuario que permitan el manejo de la estructura de elementos de edición, esto de forma sencilla y trasparente para el usuario final.
4. Implementación y visualización de un documento de tipo **General**.

3.3. Metodología

La Programación extrema [49] y el **Proceso Unificado** son dos de las metodológias, en Ingeniería de Software, más empleadas para el desarrollo de nuevos productos. La primera es parte de las llamadas *metodologías ágiles*, cuyo principal objetivo es el software, a diferencia de las llamadas *metodologías monumentales*, donde la importancia recae tanto en el software como en la documentación y los pasos que define el proceso para llevar a buen fin el desarrollo del sistema.

Para la realización de este trabajo, debido al número de desarrolladores y al nivel de conocimiento en el diseño e implementación de los objetivos se decidió poner en marcha varias de la prácticas propuestas por **Extreme Programming**.

[17] Ver sección 2.5 pag. 41. Estado del arte; síntesis, discusión y análisis.

3.3.1. Programación Extrema

XP (*Programación Extrema*) es una metodología ágil que enfoca como primer objetivo la codificación.

La *Programación Extrema* se centra en la constante revisión de código, pruebas frecuentes, relación muy cercana con el cliente, rápida retroalimentación, redefinición de la arquitectura cuando sea necesario, integración continua, rediseño y constante planeación.

XP define **cuatro valores** principales:

Comunicación Debe existir la mayor comunicación posible entre los integrantes del equipo y en igual medida con el cliente.

Simplicidad Se propone no sobre-diseñar algoritmos. No es recomendable complicar sin necesidad el proyecto.

Retroalimentación La retroalimentación se logra con pruebas de código, iteraciones pequeñas de las etapas del proyecto, programación en parejas y otros métodos.

Determinación Se recomienda tener la determinación de hacer las cosas como se debe. No tener miedo de volver a escribir alguna parte del código a razón de no saber que efectos pueda tener este cambio en todo el código restante. Se debe hacer los cambios y tener el conocimiento y control de ellos.

XP define **cinco principios.**

Proveer una rápida retroalimentación Se enfoca en iteraciones pequeñas entregadas al cliente para asegurar que el sistema entregado será el que él necesita.

Asumir simplicidad. Cada problema deberá ser resuelto de la forma más sencilla posible. También esta simplicidad implica diseñar sólo para la iteración actual en la que se encuentre el proyecto.

Realizar cambios incrementales Los cambios deben ser etapa a etapa. Con lo cual se obtiene un sistema que etapa a etapa se conoce y se controla.

Esperar cambios Los cambios en el software son siempre esperados. Se debe tener el enfoque de realizar estos cambios sin miedo con el fin de llegar a lo que el cliente necesita.

Realizar el trabajo con calidad El trabajo debe realizarse con calidad posible evitando con esto muchos problemas.

Estos **valores y principios** son llevados a cabo por **doce reglas**.

Entregas pequeñas La idea **entregas pequeñas** es el mayor numero de reglas de negocio resueltas, con el menor esfuerzo de codificación posible, en tiempos definidos de entregas llamados **iteraciones**. Estas **iteraciones** integran todos los pasos del proceso para lograr entregas de módulos del sistema, en tiempos pequeños. Cada requerimiento debe ser implementado con la mayor funcionalidad posible. Las **entregas pequeñas** dan una rápida retroalimentación por parte del cliente y reducen el riesgo de codificar software que el cliente no desea.

Planeación El propósito de la **planeación** es determinar el alcance de la iteración actual. Esto se logra determinando la tarea que es más importante para el cliente y resolver ésta como primer objetivo. La **planeación** involucra directamente al cliente para determinar el alcance del proyecto, las tareas con prioridad, las tareas de cada entrega programada, y los días de entregas.

Diseño simple El **diseño simple** tiene como idea principal no tratar de resolver problemas futuros: problemas que no estén contenidos en la iteración actual que se está realizando. La mayoría de los clientes no saben en realidad lo que necesitan hasta que se les entrega algo tangible, con base en lo que ellos puedan decidir. Así, haciendo diseños entregables y tangibles, se dá al cliente una forma de saber qué es lo que requiere y al desarrollador le evita codificación no enfocada en las necesidades del cliente.

Pruebas Para las **pruebas** se pretende resolver errores, rediseñar o volver a codificar las clases o componentes que presenten errores. Esto con el fin de generar código útil y sistemas funcionales. Existen herramientas que automatizan estas pruebas, pero para sistemas específicos se tienen que aplicar mecanismos capaces de probar el sistema particular en desarrollo.

Integración continua La **integración continua** nos permite observar qué cambios pueden hacer fallar al sistema en periodos pequeños de integración, esto es, si se integran de forma continua las partes del sistema, se podrán realizar ajustes o modificaciones en ese momento a las partes específicas, en lugar de realizar estos cambios cuando las partes para integrar sean demasiadas y los detalles asociados a éstas hagan más complicado el trabajo. Además, la codificación realizada es aún reciente para los desarrolladores, lo que permite realizar cambios de forma más eficiente.

Mejoras al código realizado Las **mejoras al código** permiten realizar cambios en el código manteniendo la simplicidad. Las pruebas y las mejoras de código están sumamente relacionadas, si se realizan pruebas a cada componente junto con la integración continua de estos, se dará valor suficiente al programador para mejorar el código en cualquier momento.

Programación en parejas La **programación en parejas** es probablemente la practica más revolucionaria de **XP**. Esta práctica mejora considerablemente la comunicación entre los desarrolladores, evitando que se realize trabajo doble por parte de estos. Otra característica es que permite que el conocimiento se propague en cada uno de los desarrolladores en igual forma y que el conocimiento de cada uno de los programadores o el conocimiento generado sea compartido por el equipo. Dos cabezas piensan mejor que una y los errores generados por un programador pueden ser encontrados y señalados por el otro, evitando así la generación de más errores.

Código sin pertenencia Si no se define una pertenencia de código, cualquiera puede hacer cambios a éste sin tener que esperar. La **no pertenencia de código** contribuye para la mejora del mismo código, ya que si se encuentran mejoras a realizar o errores, estos se pueden solucionar

rápidamente. La programación en parejas contribuye a que cualquiera de los desarrolladores tenga el conocimiento para realizar los cambios. Además, evita tener que esperar por partes del código dependientes, retrasando la codificación.

Cuarenta horas de trabajo Para **XP** si el trabajo definido no puede realizarse en **40 horas a la semana** entonces algo está mal. Con más de este tiempo los programadores comienzan a cometer errores. Junto con el tiempo de trabajo definido, el lugar de trabajo deberá necesariamente ser en el lugar donde labore el cliente, esto para resolver cualquier inquietud y no hacer suposiciones incorrectas.

Trabajo en el lugar del cliente El trabajo debe ser realizado en el lugar donde el cliente se encuentra. Esto permite una mejor comunicación entre los desarrolladores y el cliente, evitando errores respecto a las necesidades del cliente.

Metáfora Una **metáfora** es un lenguaje común que todos los miembros del equipo y el cliente deberán conocer. La idea es que el cliente y desarrolladores sean capaces de tener la misma visión del sistema para referirse a él. Para los programadores en específico deberá de existir un **estándar de código** que será seguido por cada uno de ellos, para tener un lenguaje común de códificación evitando confusiones y desorden en la programación.

Estándar de codificación Se debe definir un estándar de codificación, donde el nombre de variables, clases, métodos, etc. se definan en un principio. El estándar uniforma los nombres usados en la codificación evitando así una gran cantidad de posibles errores.

3.3.2. Prácticas propuestas de la Programación Extrema para la implementación de nuevos módulos en ELXI

Al inicio del trabajo con **ELXI**, durante el levantamiento de requerimientos, se observaron errores en el código existente del sistema. Los errores que presentaba **ELXI** se podían clasificar, dependiendo de sus prioridades, en errores del funcionamiento básico del sistema o en errores con baja prioridad

en el funcionamiento del sistema (tales como, errores de presentación, que implicaban posibles mejoras de algunas características del sistema). Esto llevó a la decisión de asignar un periodo de tiempo (tres meses) para la corrección de estos errores.

Al tener ubicados los errores, cada uno de ellos fue asignado a una *estrategia de solución*, la cual definía en periodos cortos (una semana o más) el análisis, diseño e implementación de la solución, a realizar por un par de programadores.

Cabe mencionar que en un principio se trató de seguir los pasos marcados dentro del **Proceso Unificado**. Se asignó, por ejemplo, a cada uno de los integrantes del equipo un rol, dentro del equipo de desarrollo; lo cual fue problemático debido a que la cantidad de programadores era muy pequeña (tres programadores). Además, los riesgos y tiempos de entrega para el sistema se enfocaban mejor con prácticas de una metodología ágil.

Así, entonces, al final de periodos determinados, más pequeños que los definidos para la iteración (un día), se decidió que se deberían de realizar integraciones de las soluciones a los errores encontrados. El sentido de esto fue lograr versiones siempre estables y funcionales del sistema, las cuales pudieran definir un estado de avance en la planeación general programada.

La decisión de programar en parejas asignaba una determinada cantidad de puntos de errores a resolver a un par de programadores. Lo que permitió que luego de los tres meses definidos para la corrección de errores, cualquiera de los desarrolladores tuviera el conocimiento necesario, tanto del sistema como de las herramientas de la que se estaba haciendo uso —además, claro esta, de tener resueltos los errores encontrados—.

El sistema en este momento pasó a una nueva etapa. Se asignaron actividades específicas de la estrategia, que en ese momento se estuviera llevando a cabo, a cada uno de los integrantes del equipo de desarrollo. De esta forma se siguieron realizando las practicas propuestas en principio con la salvedad de aquella en la que se solicita la programación por parejas de desarrolladores.

Al terminar la etapa de corrección de errores la definición de puntos en la estrategia para cada iteración cambió. Uno de los programadores se enfocó

directamente en la solución de la *anotación de los documentos en ELXI*; otro a colaborar en la solución de problemas específicos definidos en cada una de las iteraciones, como instaladores del sistema, seguridad, o ayuda a puntos específicos de los otros programadores; y el último a la solución de la edición y visualización de documentos científicos y técnicos, que es lo que atañe a este trabajo.

3.3.3. Patrón de pruebas del sistema

Para la realización de pruebas unitarias, debido a la naturaleza de los módulos del sistema distribuido que se estaba implementando, no se podían implementar directamente clases de pruebas a los módulos programados, por lo que se buscó aplicar algún *patrón de pruebas* específico para el tipo de sistema que se estaba desarrollando. Se aplicó, así, el patrón de pruebas de integración de sistemas distribuidos propuesto por Buschmann [16].

Este patrón de pruebas define la intención de demostrar la estabilidad de interacción entre pares de nodos aislados para luego incrementar el alcance hasta el total de nodos del sistema. Este patrón es útil en el contexto en el que el sistema bajo prueba incluye muchos componentes que corren concurrentemente y no existe un control sencillo entre ellos. Cabe señalar que en este patrón no se cubren todos los posibles puntos de fallo, pero se concentra en aquellos con más importancia en ejecución.

Para este patrón existen varios modelos de pruebas: manejo de componentes de riesgo, manejo de componentes con menos riesgo, manejo de componentes con mayor dependencia.

De estos modelos, se decidió probar con el primero: *manejo de componentes de riesgo*, donde se comienza por probar las interfaces y componentes con más riesgo a fallar. Para poder hacer estas pruebas se definió una estrategia apegada a la que propone el patrón. Se desarrolló, para cada nodo una configuración con la cual se probó cada uno de estos; se verificaron las características de los equipos en los que se deberían de probar cada uno de los clientes; se ubicaron errores en el funcionamiento más sencillo del sistema; se agregaron más nodos y se volvió a probar el sistema.

Cabe aclarar que durante la implementación del sistema se realizaron además pruebas e integraciones de forma continua con, al menos, un cliente y el servidor *levantados* con el fin de lograr versiones siempre estables del sistema.

Así, con prácticas y pasos que la *Programación Extrema* propone y adaptando éstas a las necesidades específicas del sistema, se buscó, durante el tiempo de trabajo, el seguimiento de la metodología.

3.4. Síntesis y discusión

En este capítulo se mostraron las características existentes del editor, así como los requerimientos involucrados en la nueva propuesta para este trabajo de tesis. Estos requerimientos conforman los puntos a analizar, diseñar, e implementar en los siguientes capítulos. De esta manera para esta tesis, se tomó el trabajo realizado en la versión 1.0 del editor colaborativo **ELXI**. Esta versión consta de 20 clases en el cliente y un servidor RMI con 30 métodos para la administración distribuida de los documentos. Estas clases y métodos forman la estructura de funcionamiento del editor **ELXI**.

Los requerimientos se estructuraron en cuatro grupos: i) distribución de fragmentos, ii) visualización de documentos, iii) interfaces de usuario para el manejo de elementos de edición, y iv) la implementación y visualización de un documento **General**. Estos requerimientos serán implementados con el fin de hacer más rico y funcional al editor **ELXI**. Al final de este trabajo se pretende obtener un editor más completo, para permitir la generación de documentos más elaborados y complejos.

Se decidió que la metodología a usar sería *Extreme Programming*. Dicha metodología de Ingeniería de Software para el desarrollo de proyectos, ofrece un acoplamiento al desarrollo de sistemas con altos riesgos debidos a cambios. Esta metodología será aplicada durante este trabajo con el fin de llevar a buen fin todo el proceso de desarrollo del editor **ELXI** en esta nueva fase.

En los siguientes capítulos, continuaremos con el análisis, diseño e implementación de los requerimientos planteados en estas secciones.

Capítulo 4

Análisis y estrategias para la edición de documentación técnica en ELXI

Un problema importante en numerosas áreas científicas, técnicas y, en general, en todas las ramas del conocimiento humano es el desarrollar, producir y mantener una misma documentación, en la cual generalmente participan varios autores.

La aplicación de trabajo colaborativo **ELXI** presenta, como ya se mencionó, elementos innovadores, como un modelo conceptual de interacción y comunicación, así como un protocolo social de colaboración. Gracias a estos elementos, los diferentes colaboradores, por un lado, pueden contar con un acceso (en lectura y/o escritura) simultáneo y dinámico a las diferentes partes (fragmentos) que componen un documento, guiado por un conjunto de roles de edición; y, por otro lado, les es posible que en todo momento estén concientes tanto de la evolución del documento, como del trabajo de edición que realizan los demás colegas.

Asimismo, es un producto: a) parametrizable y flexible, gracias a la integración de la tecnología XML, lo cual permite elaborar cualquier tipo de documento y mantener una compatibilidad total con otros formatos y aplicaciones documentales actuales; b) escalable, gracias a su diseño basado en objetos, lo cual permite agregarle módulos con nuevas funcionalidades (de

edición y colaboración); y c) portable, gracias a su implementación en el lenguaje de programación Java, lo cual permite su ejecución en múltiples plataformas.

4.1. Caso de estudio: Documentos de Normatividad en el IMP y PEMEX

Tomando en cuenta que en las últimas décadas, muchas industrias se han transformado y han pasado de ser organizaciones monolíticas y centralizadas a organizaciones mucho más estructuradas y globalmente distribuidas (como es el caso también de las compañías petroleras), se tienen ahora nuevos desafíos en la forma de trabajar, comunicarse, generar conocimiento y unir los esfuerzos de los diferentes especialistas distribuidos.

Gran parte del conocimiento, que se genera colectivamente en una organización, se consolida en documentos. De esta manera, los documentos se vuelven los instrumentos portadores del conocimiento de la empresa y constituyen uno de los activos más valiosos.

Actualmente, las organizaciones y empresas tienen necesidad de producir y administrar proyectos documentales, los cuales, dependiendo del tipo de procesos e información que manejen, podrían ser voluminosos y complejos. Por otro lado, en muchas ocasiones existen guías, formatos y regulaciones que se necesitan aplicar y/o seguir al elaborar la documentación y además se cuentan con restricciones importantes de tiempo para producirlos.

En el caso concreto de PEMEX, al igual que en muchas otras grandes empresas, los equipos de especialistas se encuentran apartados unos de otros y en muchas de sus actividades productivas requieren participar conjuntamente en la realización de proyectos documentales comunes. Un ejemplo son los Comités de Normalización de PEMEX y Organismos Subsidiarios, en donde varios especialistas de PEMEX y otras instituciones, así como empresas y consultores técnicos, unen sus esfuerzos para trabajar en un proyecto documental de Normatividad Técnica.

Actualmente, estos Comités de Normalización ven perturbada su productividad debido a problemas tales como: la dificultad de poder integrar las diferentes revisiones que son generadas por los especialistas que elaboran una **Norma**; la necesidad de tener que desplazarse a un lugar común para asistir a reuniones y poder tomar decisiones; la dificultad de poder estar al día sobre la evolución que está teniendo la elaboración de las Normas; el seguimiento y cumplimiento de fechas de entrega; etc.

Como se mencionó, una **Norma** presenta una estructura determinada —ver sección 3.2— y es originada con el objetivo de adecuar el marco normativo de PEMEX vigente respecto a los procedimientos para la emisión de Normas Oficiales Mexicanas. Con este fin, existe un documento llamado *Guía para la emisión de normas de referencia de Petróleos Mexicanos y organismos subsidiarios* [19] donde se pretende que las disposiciones para la estructuración, redacción y presentación de las Normas de Referencia de Petróleos Mexicanos y Organismos Subsidiarios, establecidas en este documento guía, sean acordes a la normatividad oficial correspondiente.

Para este trabajo las **Normas** antes mencionadas serán tomadas como caso de estudio. Esta **Normas** son necesarias para la coordinación de trabajo en PEMEX y en el Instituto Mexicano de Petróleo, por lo cual se realizarán las implementaciones necesarias con el fin de que puedan ser generadas en **ELXI**. Más aún, el objetivo de este trabajo es poder generar documentos en **ELXI**, los cuales no estén relacionados a ningún tipo de documento específico. Por esta razón, como consecuencia de la implementación para la generación de **Normas** en **ELXI**, y de la evaluación de requerimientos de la primera versión de **ELXI**, se implementarán los módulos necesarios para la generación de un documento **General**. Este documento **General** deberá ser capaz de generar cualquier tipo de documento técnico y científico en **ELXI**.

Por las razones antes expuestas, las **Normas de referencia** y el documento **General** se establecen como casos de estudio para este trabajo.

4.2. Distribución de fragmentos

La implementación de concurrencia y distribución de documentos existente en **ELXI** permite la edición colaborativa de documentos de forma ágil

y ordenada. Los requerimientos mencionados, en la sección anterior, dejan abierta la posibilidad a cambios en esta implementación. Cabe la posibilidad, también, de usar la estructura definida sin cambios tratando de modificar sólo módulos específicos sin alterarar el trabajo ya realizado.

En esta etapa, se tomó la decisión de mantener la estructura de los documentos. Los fragmentos no cambiaron su posición en el modelo de contenido del documento, más aún, de forma contraria, se decidió hacer crecer el modelo con referencia al contenido de un fragmento. Un fragmento contiene texto y este texto, junto con los elementos de edición, se manejan y distribuyen sólo como una unidad de texto.

Puede entonces definirse como **unidad distribuida** al contenido que un fragmento maneja como bloque de texto. Es decir, se implementaron las clases necesarias para que un documento pudiera embeber, dentro de los bloques de texto de los fragmentos, código que definiera más elementos de edición. La implementación, en la version 1.0 de **ELXI**, toma sólo como texto el contenido de los fragmentos al momento de distribuirse. Para este trabajo, al momento de llegar a cada uno de los clientes los contenidos de los fragmentos, las clases implementadas deben de obtener estos contenidos —elementos de edición definidos en código— y crear un documento más grande y complejo, en el cual se pueda manejar cada uno de los elementos de edición, antes solo tomados como texto (ver figura 4.1).

Con esto, se lleva a cabo la reutilización de código existente y se evitan los posibles problemas de distribución de elementos de edición —los cuales probablemente hubieran llevado a problemas más complejos, por ejemplo, la distribución de elementos de una lista anidada—.

4.3. Visualización del documento

Cada elemento definido en el modelo de documento de **ELXI** está relacionado con una plantilla de patrones XSL[1] la cual define la presentación final del documento. Para cada elemento nuevo definido debe existir un patrón

[1] Ver sección 2.4.6.

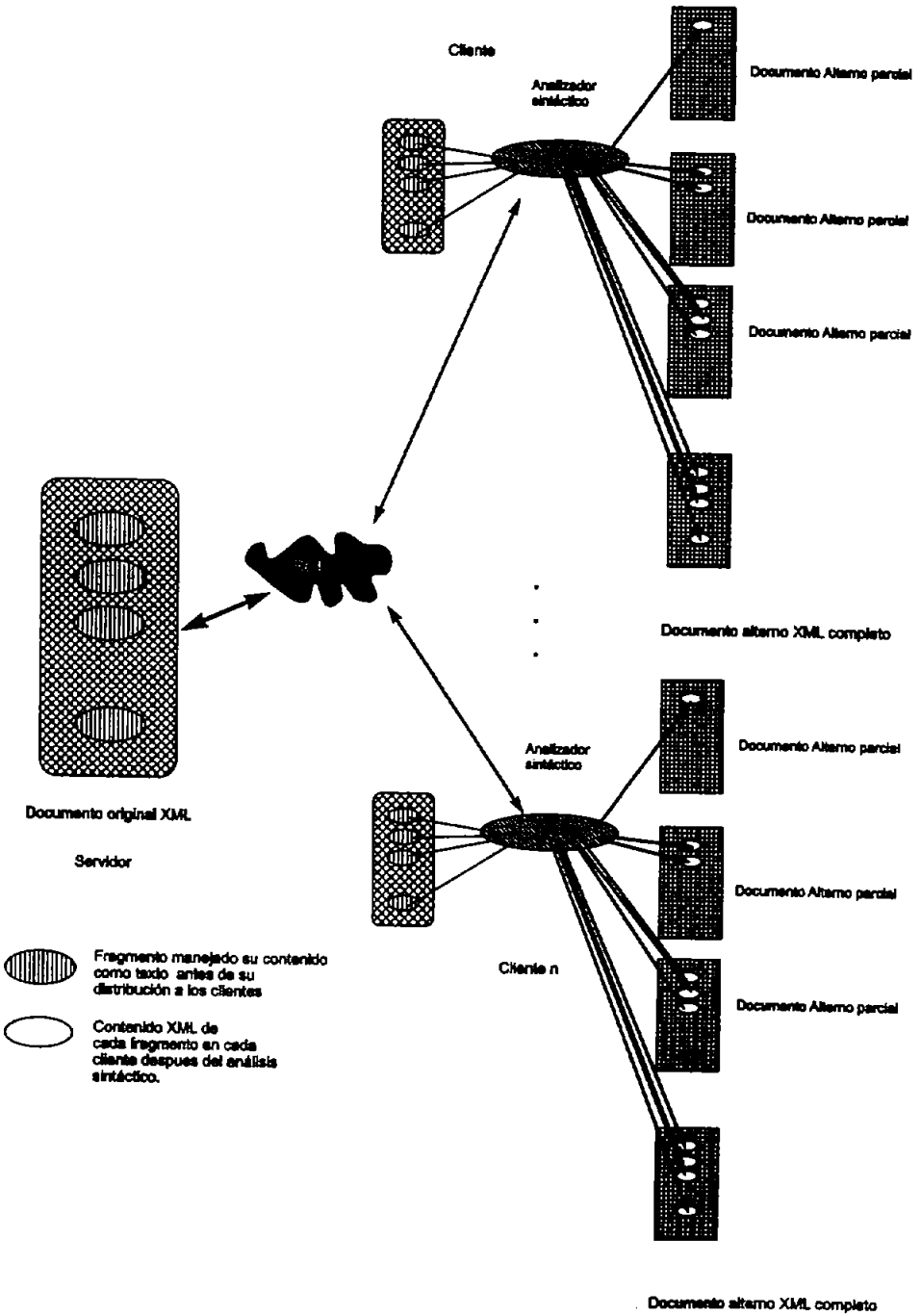


Figura 4.1: Diagrama para la distribución de fragmentos

de visualización. Así, para el documento de tipo **Norma** se deben de definir los comportamientos de visualización para cada *nuevo* elemento. El número de elementos se definió en términos de las necesidades generadas al editar una **Norma**, y de los requerimientos de la primera versión de **ELXI**. Estos elementos se consideraron como base para la edición de cualquier documento. Como se observará, en el siguiente capítulo, el número de elementos puede crecer si fuera necesario, es decir, la estructura del documento de **ELXI** permite agregar nuevos tipos de elementos sin demasiados problemas.

Los elementos nuevos a agregar en la estructura de **ELXI** son los siguientes:

4.3.1. Texto

Para la generación de párrafos, se definió un nuevo elemento llamado **texto**. Este elemento contiene caracteres, los cuales, al ser agregados dentro del contenido de un elemento *texto*, permiten la agrupación de estos en párrafos. Es decir, se puede definir más de un elemento de tipo texto dentro de un fragmento con el fin de generar párrafos dentro del fragmernto. Para separar el elemento *texto* de otros elementos, se asigna un retorno de carro y dos líneas de separación al final de su contenido.

En la versión 1.0 de **ELXI**, sólo un bloque de texto podía ser asignado a un fragmento. Así, con este tipo de elemento se asignan varios elementos *texto* dentro de un fragmento, lo cual es necesario para una correcta edición.

4.3.2. Listas

Una *lista* es un elemento básico en cualquier editor de texto y este elemento fue también necesario para la edición del documento de tipo **Norma**. Este elemento de edición se subdividió en tres elementos de tipo *lista*. Los elementos generados fueron: *listas con balas*, *listas numeradas* y *listas simples*. Así, para una *lista numerada* se definió el elemento **listaN**. Para la *lista con balas* se definió el elemento **listaD**. Finalmente, para una *lista sencilla* se definió **lista**. Para cada nueva aparición de un tipo de elemento, en la plantilla de patrones de transformación se implementó la visualización especifica y deseada para el documento de tipo **Norma**.

En cada elemento *lista* se agregó otro nuevo elemento dentro de ella que define el contenido de ésta. Así, cada elemento de tipo lista puede contener elementos de tipo *elementoX*: **elementoN** se definió para *elementos de la lista numerada*, **elementoD** para *elementos de la lista con balas*, y **elemento** para *elementos de la lista normal*. Cada uno de estos elementos está agregado como parte del elemento *lista*.

A su vez, cada *elemento de una lista* debe poder contener párrafos. Por tanto, cada *elemento de una lista* se relacionó con el elemento de edición *texto*, anteriormente definido, permitiendo su inserción dentro de la estructura de una lista[2]. De este modo, una *lista* puede contener *elementos* y a su vez estos elementos pueden contener *párrafos*. Esta estructura es muy parecida a una lista *description* en LaTeX[48].

En un documento **Norma**, además aparecen *listas* dentro de otras *listas*, por lo cual el requerimiento de una *lista* contenida dentro de otra *lista* fue necesario en la nueva estructura del documento **ELXI**. Esta necesidad planteó el problema de definir un nivel de profundidad deseado para los elementos de una *lista* o la implementación del anidamiento de *listas* hasta un número infinito. Como se explicará más adelante, debido a las herramientas utilizadas, la segunda opción fue la más viable.

4.3.3. Tablas

Para poder implementar una *tabla*, se debe tener la posibilidad de definir un número finito de *celdas* dentro de ésta. Cada *celda*, a su vez, puede contener otros elementos de edición: ya sean *listas*, *imágenes* o *tablas* mismas, además obviamente de *párrafos*. Así, para poder tener la posibilidad de agregar *tablas* al documento **Norma**, se definió el elemento **tabla**, el cual, a su vez, contiene elementos **celda**.

4.3.4. Notas a pie de página

Las *notas a pie de página* permiten hacer referencias a secciones de texto de las cuales interesa dar una referencia más específica. Para definir este tipo

[2]Más adelante, en esta sección, observaremos que todos los nuevos elementos están relacionados: una *lista* puede contener *párrafos*, *imágenes*, *listas* etc., una *tabla* puede contener *listas*, *imágenes*, *párrafos* etc.

de elemento dentro del lenguaje se marcó el lugar donde se necesitaba hacer la referencia, y luego al final de la página en la que se visualiza esa sección del documento, se describió a detalle la información deseada. El manejo y visualización de documentos en **ELXI** permite la exportación a los formatos **HTML** y **PDF**, de los cuales se partió para el diseño de *notas a pie de página*. Una página **HTML** permite manejar su contenido por medio de un visualizador, el cual muestra de forma continua y completa la información del documento. Así, para una página **HTML** las *notas a pie de página* fueron referenciadas al final de la página, con un número incremental igual al número donde se realizó la referencia dentro del cuerpo del documento.

A diferencia de una página **HTML**, un documento **PDF** permite su estructura en tres partes: encabezado, cuerpo y pie del documento por cada página del documento. Por tanto, dentro de la sección de pie del documento se decidió imprimir la información deseada a la referencia en esa página, con un número consecutivo referido a la sección deseada.

4.3.5. Encabezados y pies de página

El encabezado y pies de página no existen como tal dentro de la estructura de un documento **HTML**, por tal motivo no se agregó en la visualización del documento **Norma** en este formato. Para un documento **PDF**, como se mencionó anteriormente, existen secciones encabezado y pie de página específicas, las cuales fueron tomadas para agregar en esas partes la información relacionada con el encabezado y pies de páginas del documento **Norma**.

Una **Norma** presenta, en su primera página, un encabezado con información referente a la **Norma**, y en las siguiente páginas esa misma información de encabezado se muestra con otra presentación. Los pies de página de una **Norma** contienen una leyenda relacionada con la autenticidad de ésta, la cual no cambia su contenido a lo largo del documento. Veremos en la sección siguiente que estas características se involucraron con el diseño del documento de tipo **General**.

4.3.6. Índices

La forma de poder ubicar rápidamente información en un documento es a través de un índice. Un índice paginado para un documento **HTML** no

existe como tal, ya que la información se presenta de forma continua y en una sola página a la vez, en cambio, para el documento en formato **PDF** se permite la estructura de un índice debido a que cada página del documento se puede numerar.

Por tanto, se mostró útil poder desplegar un índice de referencias de cada una de las partes importantes de un documento **Norma**. Cada una de estas parte ya estaba definida como fragmento de un documento en **ELXI** [17]. Por lo cual, el índice se formó ahora en referencia a cada uno de estos fragmentos. El índice se ubicó al inicio de la visualización generada.

4.3.7. Imágenes

Las imágenes dentro de un documento **Norma** pueden estar en cualquier sección del documento. Para esto, se definió el elemento *imagen* el cual debe permitir la inserción de imágenes dentro de un fragmento. Con respecto a este elemento, la definición dentro del modelo de contenidos no fue demasiado complicada. El mayor problema fue en el momento de la distribución de documentos a los clientes, ya que cada una de las imágenes insertadas por algún cliente se debió administrar en su distribución para que los demás usuários pudieran ver sin inconsistencias los cambios o agregaciones en las imágenes.

4.3.8. Código embebido en fragmentos

Al existir ya un editor, el cual permitía agregar texto dentro de cada uno de los fragmentos, y habiendo tomado la decisión de reusar el código ya existente, se decidió que el contenido que antes era sólo texto como contenido de un fragmento, ahora sería otro documento XML el cual contendría todos los elementos de edición necesarios para editar ese fragmento. Con esto, se logran crear las secciones definidas dentro del documento **Norma**. Es decir, si en un documento **Norma** en la parte *Concordancia con otras Normas* era necesario agregar una tabla o una lista o una imagen además de sus párrafos, entonces ahora se podrían usar completamente, dentro de un fragmento de tipo *Concordancia con otras Normas*, todos los elementos de edición necesarios para esa parte de la **Norma**. Además, ese contenido sería tomado por el sistema sólo como texto al ser distribuido, pero como un documento pequeño de edición de elementos al llegar a cada cliente.

Para que el sistema pudiera ubicar los contenidos de cada uno de los fragmentos, como pequeños documentos de modelo de contenido de una parte de un documento **Norma**, se implementaron las clases necesarias para armar un documento completo de lado del cliente. Este documento muestra el modelo de contenido del documento **Norma** completo. A este documento completo se le llamó **Documento Alterno**.

Las clases implementaron un analizador sintáctico, el cual permite revisar el contenido del documento compartido por varios usuários. Esto es, el analizador toma el contenido de texto de cada uno de los fragmentos y los transforma en pequeños documentos XML. Con cada uno de estos fragmentos transformados se crea un documento completo, el cual, al ser creado en el cliente, permite la visualización del documento **Norma** completo. La DTD manejada en la versión 1.0 de **ELXI** [18][17], se utiliza para toda la problemática de concurrencia y colaboración, y el **Documento Alterno**, junto con la nueva DTD, permite la visualización del documento en el cliente.

Para cada uno de los fragmentos, en la implementación del documento de la versión 1.0 de **ELXI** (**Documento Base**), se realizan operaciones de cambios, altas, bajas y modificaciones por cada fragmento. Toda la implementación se realiza por medio de un módulo de acceso a distancia (remoto) en el servidor, de tal forma que la última versión actualizada del documento se encuentra en el servidor. Estas operaciones se realizan por cada fragmento y también por sus atributos. Es decir, puede haber cambios en los títulos o tipos de un fragmento del **Documento Base** —los títulos o tipos son manejados como atributos para un fragmento—. De esta forma, cada operación de modificación hecha en el **Documento Base** implicó su implementación también en el **Documento Alterno**. Esto con la idea de permitir una sincronización entre los dos documentos y permitir la visualización del documento en cualquier momento con las actualizaciones requeridas. Si se realiza algún cambio en el **Documento Base**, este cambio se refleja también en el **Documento Alterno**.

El **Documento Alterno** es necesario en el momento de la visualización, por lo cual, éste es generado sólo hasta el momento en que se visualiza el documento en edición. Esto lleva a la idea de que todas las modificaciones, anteriormente mencionadas, podían ser solo llevadas a cabo en el **Documento Base** y la generación del **Documento Alterno** llevarse a cabo con los

cambios ya realizados en el **Documento Base**. Al proponer esto, se reducen las operaciones necesarias de actualización en el **Documento Alterno**, permitiendo de este modo un mejor desempeño de la aplicación.

La implementación se realizó desde ambos puntos de vista. Se implementaron las operaciones necesarias de actualización en los dos documentos, pero se permitió que la generación del **Documento Alterno** fuera solo con las actualizaciones realizadas en el **Documento Base** y hasta el momento en que la visualización del documento fuera necesaria. Cabe señalar que parte de este último punto, dentro del análisis, fue llevado a cabo en colaboración con el grupo de desarrollo de **ELXI** en el IMP para lograr un mejor desempeño de la aplicación.

4.4. Interfaces de usuario

Cada uno de los elementos de edición creados son parte del modelo de datos del **Documento Alterno**. Desde este punto, la edición de un documento, para la vista del usuario, es un documento dividido por líneas que marcan las partes que conforman el documento. En cada una de estas partes se pueden editar los elementos creados. Estos al escribirse se representan como etiquetas de marcado dentro del lenguaje definido en el mismo modelo de contenido. Cada una de las etiquetas hacen referencia a los elementos de edición creados, los cuales se escriben dentro de las áreas marcadas entre líneas, que representan los fragmentos en la interfaz de edición de **ELXI**.

Este lenguaje es suficiente para usar los elementos creados directamente en el editor, pero implica un conocimiento de la sintaxis de definición usada. El lenguaje creado debe ser usado con la sintaxis exacta y ser escrito en los fragmentos que en **ELXI** eran usados para la edición de texto. Por tanto, es muy complicado para el usuario final la escritura de algún documento, ya que además de saber la sintaxis del lenguaje debería tener el manejo y conocimiento básico del funcionamiento modificado de **ELXI**. De este modo, surgió la necesidad de contar con interfaces de usuario que permitieran la generación de elementos de edición. Esto también debido a la idea de que el editor estuviera dirigido a cualquier tipo de usuario, con o sin conocimientos previos de la definición de sintaxis de un lenguaje o del funcionamiento interno del editor.

Imágenes

La interfaz para manejar las imágenes debe permitir la agregación de una imagen, así como la modificación de ésta por cualquier otro autor que comparta el documento. Dando de esta manera la posibilidad de agregar al documento imágenes establecidas por los diferentes autores.

Listas

Para cada interfaz se buscó una estructura que permitiera cambios en posibles elementos anidados contenidos en el elemento de edición. La interfaz de lista permite la agregación, borrado y modificación de cada uno de los elementos de la lista, permitiendo, de este modo, manejar adecuadamente su contenido. Como se mencionó anteriormente, para la edición de un documento tipo **Norma** se necesita la implementación de un mecanismo, el cual permita el anidamiento de listas, por lo que se consideró este punto en el diseño de la interfaz.

Una lista se estructuró con *elementos de lista*, además de párrafos dentro de cada uno de estos *elementos de lista*. Lo anterior para permitir un contenido más rico y robusto en los contenidos posibles de la lista.

La interfaz de lista se definió con un campo para el manejo de los *elementos de lista* y un área de texto. Esta área de texto tiene el mismo comportamiento que un fragmento del editor, con el fin de poder agregar allí otra lista anidada en el contenido de la primera.

Tablas

De forma semejante, se llevo a cabo el análisis y diseño de la interfaz para el manejo de tablas. Esta interfaz permitió las modificaciones y agregaciones necesarias para el manejo de tablas en el documento de tipo **Norma**. Cada tabla tiene áreas de texto en forma de matriz, las cuales representan celdas de la tabla.

Encabezados y pies de página

Para el caso de los elementos encabezado y pie de página, no se necesitó ninguna interfaz, ya que por la misma estructura de un documento **Norma**,

ésta ya estaba definida desde la creación hecha por la misma plantilla del documento [17]. Cabe mencionar que esta parte aunque no presentó implementación alguna en un documento de tipo **Norma**, si tuvo que tener el diseño e implementación para el caso de un documento **General**.

4.5. Documento General

Para la definición de todo el modelo de contenido del lenguaje de marcas, se tomó como base un documento de tipo **Norma** y la evaluación realizada a **ELXI** en su primera versión. El documento **Norma** es uno de los casos de estudio del presente trabajo, el cual dio una gran ayuda para el análisis de cada uno de los elementos de edición. Una **Norma** es un documento con doce elementos a considerar para su edición completa y correcta (ver sección 3.2). Luego de definir todos los elementos, se continuó el trabajo con un documento, el cual pudiera editar cualquier tipo de estructura dentro de él, desde una carta o un informe de actividades hasta documentos tan complicados como es el caso de una **Norma**.

Un documento de tipo **General** se construye con algunas modificaciones hechas en el diseño de las plantillas XSL hasta ahora analizadas. Se puede pensar en tipos de fragmentos no tan específicos como un documento de tipo **Norma**. En lugar de un tipo de fragmento *autores* —el cual en la plantilla de diseño solo le es permitido mostrar párrafos centrados con los nombres de los autores de la **Norma**— o el tipo *título* —al cual también solo le es permitido editar nombres en letras de tamaño grande centradas— se puede permitir la edición de un tipo de fragmento *libre*, el cual puede tener todos los elementos de edición diseñados. De esta forma se permite al usuario tomar aquellos elementos de edición útiles al tipo de fragmento y documento que quisiera editar.

Así, con fragmentos de tipo *libre* se permite la edición de casi cualquier documento. Si se necesitara una carta, podría definirse el primer fragmento como tipo libre para escribir en el párrafo la información relacionada a la persona a quien va dirigida la carta. Otro fragmento de tipo *libre* serviría para redactar el cuerpo de la carta y por último otro fragmento de este mismo tipo para la firma y el remitente. Si fuera necesario que en el cuerpo de la carta se agregara una lista no habría ningún inconveniente ya que para el tipo de fragmento *libre* le es posible el manejo de cualquier elemento de edición.

Sin embargo, la necesidad de tipos específicos es también útil en el diseño del documento que llamaremos de ahora en adelante **General**. En el caso de necesitar un párrafo orientado hacia la izquierda o a la derecha o centrado, un fragmento de tipo libre no estaría enfocado para esta necesidad. Se definieron tipos específicos a cualquier documento. Estos tipos permiten, entre otras cosas, la orientación de párrafos a la derecha o izquierda, la inserción de caracteres de tamaños diferentes a los definidos como base, y la generación de sangrias en los párrafos dependientes del número de secciones y subsecciones definidas en esa parte del documento. De esta manera, los posibles nuevos tipos para un documento **General** son:

Título1 Permite la visualización del texto contenido en el fragmento con un tamaño de 24 pt para representar un título.

Título2 Permite la visualización del texto contenido en el fragmento con un tamaño de 18 pt, un poco menor al de *Título1*.

Título3 Permite la visualización del texto contenido en el fragmento con un tamaño de 14 pt, un poco menor al de *Título2*.

Sección Permite la visualización del contenido de un fragmento para representar una sección del documento en edición. El nombre asignado en el árbol de navegación aparecerá en la parte superior del texto visualizado. Por ejemplo, para una sección con el atributo *Elaboración de trabajos* asignado como título del fragmento en el árbol de navegación, aparecerá *Elaboración de trabajos* en letra remarcada en la parte superior de la sección, y el contenido del fragmento abajo de este título.

Subsección Permite la visualización del contenido del fragmento, con una indentación de 1 cm más en relación a la indentación de *Sección*. Por ejemplo, si se asignara una subsección a la sección *Elaboración de trabajos*, ésta subsección agregará el contenido del fragmento con una indentación de 1 cm más que la de *Sección*.

Subsección2 Permite la visualización del fragmento, con una indentación de 1.5 cm mayor a la de *Subsección*.

Subseccion3 Permite la visualización del fragmento, con una indentación de 2 cm a la de *Subsección2*.

Párrafo_derecha Permite la visualización del contenido del fragmento orientado hacia la derecha.

Párrafo_centrado Permite la visualización del contenido del fragmento centrado.

Párrafo_Indexado Permite la visualización del contenido de un fragmento con indentación. Esta indentación es usada en el momento en que un fragmento no desea ser definido como subsección.

Libre Permite la visualización del contenido de un fragmento representando una parte del documento en edición. Con este tipo, el nombre asignado al fragmento en el árbol de navegación, no aparecerá en el índice del documento ni en la parte superior del texto visualizado. Por ejemplo, si se deseara agregar más elementos o compartir parte del contenido del fragmento *Elaboración de trabajos*, se podría agregar un tipo libre, el cual no agregará ningún título en la parte superior de su contenido.

Párrafo Permite agregar texto, en forma de parrafos, en alguna sección que deseara ser compartida entre más colaboradores. Su funcionamiento es muy semejante a *libre*.

Itálica Permite la visualización del contenido del fragmento con un estilo de letra itálica.

Bold Permite la visualización del contenido del fragmento marcado con un estilo de letra fuerte.

Bold-Itálica Permite la visualización del contenido del fragmento marcado con un estilo de letra itálica y marcada.

Encabezado Permite la visualización del contenido del fragmento en la parte superior de cada una de las hojas del documento.

Pie de página Permite la visualización del contenido del fragmento en la parte inferior de cada una de las hojas del documento.

Los tipos sección, subsección, subsección2 y subsección3 fueron definidos al necesitar fragmentos de tipo libre que permitieran indentar el texto, además de agregar un título al inicio del mismo. Este título está relacionado con el árbol de documentos de **ELXI** [18] y con la generación del índice del documento.

Del mismo modo, se definieron tipos de fragmento para el tipo de letra: bold, itálica y sus combinaciónes, con el fin de poder marcar párrafos, en los cuales los que se quieren denotar partes en específico. Estos tipos fueron asignados en esta forma debido a que cualquier documento generado por los lenguajes de marcas mostrados en el capítulo 2, muestran texto remarcado o indexado con el fin de hacer más entendible el contenido del documento.

Al definir los tipos Título-*n* se tiene la posibilidad de marcar el título del documento en diferentes tamaños. Esto se realizó tomando en cuenta la forma en que lo hacen varios procesadores de texto como Word o simplemente como lo definen algunos lenguajes de marcas tal como **HTML**.

Las tablas, imágenes, listas etc. tendrán el mismo comportamiento que en los fragmentos de un documento tipo **Norma**. Cada uno de estos elementos están agregados al contenido de los fragmentos definidos para el documento **General** con excepción de los tipos *Titulo* donde no son necesarios.

Un punto importante a notar en el documento de tipo **General**, a diferencia del documento tipo **Norma**, es que un encabezado y pie de página pueden ser definidos por el usuario. En una **Norma**, se definen en la misma plantilla del documento, debido a que la estructura del documento **Norma** ya esta predefinida, como se vio en la sección 3.2. En los requerimientos planteados, el documento **General** debe poder generar encabezados y pies de página. Por tanto, estos fueron implementadas como un tipo de fragmento. Este tipo de fragmento, al igual que *libre* o *párrafo*, permiten agregar cualquier elemento de edición —tablas, listas, imágenes, etc.— y son siempre visualizados en la parte superior o inferior, según sea el caso, del documento **PDF**. En el formato **HTML** o **RTF**, debido a que el documento es plano y de una sola página, no es necesario visualizar estos fragmentos.

4.6. Síntesis y discusión

En este capítulo se mostró el análisis y las estrategias tomadas para la elaboración de este trabajo de tesis. Se definió el caso de estudio y se analizaron los puntos más importantes para llevar a cabo la implementación de los módulos de este trabajo.

Estos puntos se dividieron en cuatro propuestas: aquella que permite la **distribución de los fragmentos** del documento; la que permiten la **visualización del documento**; la que define las **interfaces de usuários para los elementos de edición**; y la propuesta de un **Documento General**.

La propuesta relacionada con **distribución de los fragmentos** aborda el problema del manejo de los fragmentos, y de los documentos, de forma distribuida a cada uno de los clientes del editor. Cada fragmento contiene un pequeño documento estructurado, el cual contiene los elementos de edición necesarios para una correcta visualización.

La **visualización del documento** se relaciona con los algoritmos necesarios para la visualización de los documentos, así como en los formatos en los que serán presentados. La propuesta de las **interfaces de usuários** permite la administración, por parte de los clientes, de los elementos de edición de un documento (tablas, imágenes, listas, pies de página, etc.) de un modo gráfico.

Por último, se propone un **Documento General** el cual permite editar cualquier documento científico o técnico de forma distribuida en el editor **ELXI**. El **Documento General** contendrá tipos de fragmentos que permitirán agregar en su contenido cualquier elemento de edición. Con esto se logra que los documentos editados en **ELXI**, gracias a que el contenido de los fragmentos es más abierto, permitan generar cualquier tipo de texto científico o técnico de forma distribuida.

Cada una de estas propuestas se diseñaron e implementaron en las diferentes etapas del proyecto, siendo por tanto, los temas a abordar en los capítulos siguientes.

Capítulo 5

Diseño e implementación para la edición de documentación técnica y científica en ELXI

5.1. Diseño e implementación del caso de estudio

Para el diseño y la implementación de las clases que componen los nuevos módulos, se tomaron como base las clases ya desarrolladas en la version 1.0 de **ELXI**. Las herramientas de uso fueron el lenguaje de programación **Java** y el lenguaje de visualización de estilos XSLT.

La implementación de las hojas de estilo para el documento tipo **Norma** se desarrolló en función a una plantilla base ya implementada [17], la cual define cada tipo de fragmento perteneciente a una **Norma**. Estos tipos permiten definir la estructura básica de una **Norma**. Uno de los requerimientos fue, como ya se explicó en el capítulo de análisis, diseñar e implementar mecanismos que permitieran agregar más elementos de edición a cada uno de estos tipos, tales como listas o imágenes.

Uno de los problemas principales fue la distribución de cada uno de estos elementos y el reuso del código existente. La propuesta fue en principio tener la capacidad de escribir código XML en cada fragmento del documento,

el cual contendría la definición de los nuevos tipos de edición. Con esto se logró integrar como texto, en cada uno de los fragmentos, el código XML de los elementos de edición a visualizar. A cada fragmento con el código XML embebido se le llamó **unidad distribuida de edición**.

Uno de los primeros puntos a resolver fue el diseño de un analizador sintáctico que pudiera reconocer cada uno de los fragmentos existentes para tomar su contenido. Lo anterior con el fin de que mientras el contenido es distribuido a cada cliente, el sistema **ELXI** sólo tomará como texto este contenido (como sucede en la version 1.0 de **ELXI**), pero una vez entregado a cada cliente, el contenido del fragmento se traducirá a código XML, representando así en realidad su contenido a visualizar.

Una vez traducido, se decidió que cada uno de los fragmentos debían de generar, en el cliente, documentos XML pequeños para luego ser unidos y generar otro documento completo, enteramente útil para su visualización. Por tanto, se diseño e implementó la clase que debía ser capaz de construir tal documento. Esta clase se llamó **Documento Alterno**.

Para la visualización, ya con el **Documento Alterno** armado en el cliente, se implementaron las hojas de estilo XSL y XSL-FO correspondientes a cada uno de los dos documentos previstos para **ELXI**. Para la **Norma**, se agregó al diseño que se planteó en la version 1.0 de **ELXI** más tipos y *templates*[1], con el fin de poder manejar el conjunto de nuevos elementos de edición. Además, se redefinió el orden en que debería de aparecer cada sección de una **Norma** tratando de mejorar la presentación del documento, en lo referente a las secciones y subsecciones que el documento puede manejar.

5.1.1. Diseño e implementación para la distribución de fragmentos

La herramienta utilizada para crear el analizador fue DOM [24]. DOM es una especificación para el manejo de documentos XML [24], la cual fue utilizada en **ELXI** para el manejo del **Documento Base** implementado en

[1]Del inglés *template*: patrón, plantilla, modelo.

la versión 1.0 de **ELXI** [18][17]. DOM provee los métodos para el recorrido del árbol de la estructura XML, lo cual permitió la definición de los métodos necesarios, para poder resolver las necesidades de distribución de fragmentos. Se manejaron los documentos **Base original** y **Documento Alterno** con estas bibliotecas con el fin de lograr la visualización completa del documento **Norma** y posteriormente del documento **General**.

Los métodos implementados se pueden dividir en tres grupos: i) aquellos que permiten el borrado sobre el **Documento Alterno**; ii) los que realizan agregaciones sobre el **Documento Alterno**; y iii) aquellos que realizan modificaciones sobre éste.

Métodos que realizan borrado sobre el documento

boolean borrarTemp(fragObj fragBorrar) Borra el archivo temporal del fragmento almacenado en el sistema de archivos local. Al cerrar o borrar el documento original, los archivos por fragmento deben ser también borrados.

boolean borrarTempCompleto() Borra el **Documento Alterno** que permite la visualización. Además de borrar los archivos temporales, el **Documento Alterno** completo puede también ser borrado.

boolean borraTodo() Borra todos los documentos generados. Este método manda llamar a los dos últimos métodos explicados, logrando con esto dejar vacío el directorio donde se generan todos los temporales y el **Documento Alterno**.

void eliminaNodo(int idFrag_t, fragObj actFr) Elimina un nodo o fragmento del **Documento Alterno**, borrando también el archivo temporal relacionado con el fragmento.

Métodos que realizan agregaciones

Documento Alterno(XmlEdit xe) Inicializa las variables de instancia xe, pathRegresivo y pathT, las cuales ubican rutas de almacenamiento y el objeto XmlEdit con el cual se maneja el editor de documentos XML de ELXI [17].

void agregarTempaCompl(Node d, String id_fra) Agrega un nodo al archivo **Documento Alterno** que permite la visualización del documento. En la estructura del documento XML cada nodo representa un fragmento del **Documento Base** [17] [18].

void poneNombreArchivoInicio(String NomArchivo) Dá un valor a la variable de instacia *nombreArchivoOriginal*, la cual es usada para realizar el análisis sintáctico del documento original de ELXI.

String daNombreArchivoOriginal() Devuelve un valor a la variable de instacia nombreArchivoOriginal, la cual es usada para realizar el análisis sintáctico del documento original de ELXI.

void escribeTipoDocumento(String tipoDocuP) Escribe el tipo del fragmento del documento original ELXI en el fragmento correspondiente del **Documento Alterno**. Esto es escrito en cada operación *updateFragment* de XmlEdit o en el armado completo del documento alterno al pedirse la visualización. Este método se manda llamar desde *agregaTempaComp()*, el cual a su vez lo manda llamar *actualizaFragmento()* desde *updateFragment()* en XMLEdit o desde el constructor de *viewPanel()* de la clase ViewPanel en el momento de la visualización.

creaTempCompleto() Genera el **Documento Alterno** completo, el cual permite la visualización.

String obtenContenidoFragmentoOrig(String identificador) Devuelve el contenido del fragmento, del documento original, especificado por el identificador recibido como parámetro.

Vector obtenNumerodeFragmentos() Devuelve un vector con los identificadores de los fragmentos del documento original ELXI. Cuando se pide la visualización del documento, todos y cada uno de los fragmentos del documento original deben ser analizados para obtener su contenido y así formar el **Documento Alterno** completo. Este método permite saber el número de fragmentos total a ser analizados.

String obtenTipFragOrig(String idF) Obtiene el tipo de fragmento del documento original ELXI, para ser escrito en cada operación *updateFragment* de XmlEdit o en el armado completo del documento alterno

al pedirse la visualización, este método se manda llamar desde *agregaTempaComp()*, el cual a su vez lo manda llamar *actualizaFragmento()* desde *updateFragment()* en XMLEdit o desde el constructor de viewPanel() de la clase ViewPanel en el momento de la visualización.

String obtenTitActual(String idF) Obtiene el título del fragmento del **Documento Base** con el fin de ser escrito en cada operación *updateFragment* de XmlEdit o en el armado completo del **Documento Alterno** al pedirse la visualización. Este método se manda llamar desde *agregaTempaComp()*, el cual a su vez lo manda llamar *actualizaFragmento()* desde *updateFragment()* en XMLEdit o desde el constructor de viewPanel() de la clase ViewPanel en el momento de la visualización.

void crearTemp(String id) Crea un archivo temporal de un fragmento. Por cada uno de los fragmentos del documento original se crea un archivo XML que representa el contenido del fragmento. Cuando se necesita armar el documento completo se debe tomar este archivo y se une junto con los demás archivos de fragmentos para formar el documento completo.

Métodos que realizan modificaciones

void cambiaTituloFrag(String id, String tituloParam) Cambia el título del fragmento del documento Alterno en cada operación de actualización *updateFragment* de XmlEdit.

void cambiaTipoFrag(String id, String tipoParam) Cambia el tipo del fragmento del documento Alterno en cada operación de actualización *updateFragment* de XmlEdit.

actualizaTemp(String id, int idFrag_t) Si algún cambio se realiza en el contenido del documento original, este método refleja el cambio en el archivo temporal del fragmento.

Cada una de estas operaciones puede tomarse como acciones paralelas relacionadas a cambios en el **Documento Base** original. En la figura 5.1 se pueden observar las llamadas a métodos para la creación del **Documento**

Alterno. Si se deseara más adelante, por ejemplo, agregar un nuevo fragmento al **Documento Base**, se hacen de nuevo llamadas a estos métodos, con el objeto de reflejar los cambios en el **Documento Alterno**.

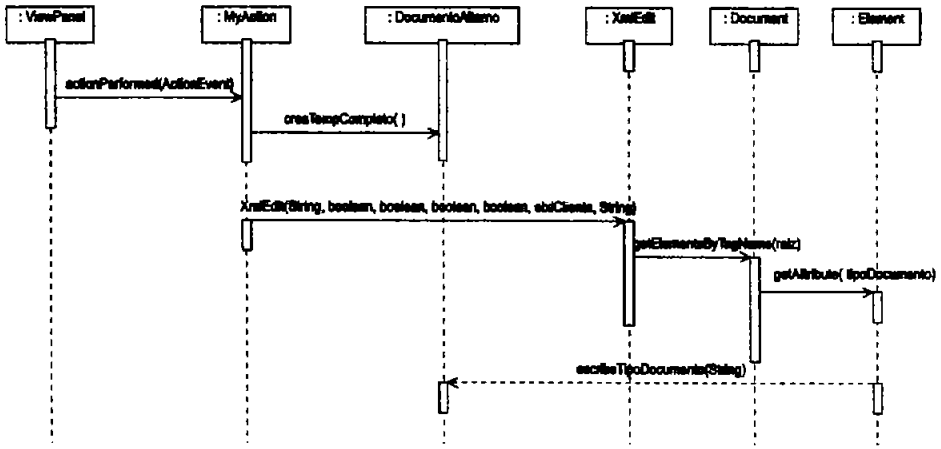


Figura 5.1: Creación del **Documento Alterno**

El la figura 5.2, se pueden observar las relaciones existentes de la clase **DocumentoAlterno** con otras clases importantes. En **ViewPanel** se encuentran los botones que realizan la exportación del documento XML a otros formatos *HTML*, *PDF* y *RTF*. En la clase **Editor**, como se mencionó, están los métodos que realizan actualizaciones al **Documento Base**. En estos métodos de la clase **Editor** se hicieron cambios con el objeto de que cuando ocurriera un cambio en el **Documento Base**, este cambio provocará la llamada a los métodos de la clase **DocumentoAlterno**. Con la llamada a los métodos de **DocumentoAlterno**, enlistados anteriormente, se reflejan los cambios del **Documento Base** en el **Documento Alterno**.

En el diagrama de la figura 5.3, se puede observar también el llamado a métodos de la clase **DocumentoAlterno** para el armado del **Documento Alterno** en relación al **Documento Base** original.

Otro de los requerimientos fue el manejo de imágenes dentro de los documentos. Para ello, se implementaron métodos dentro de la clase *Servidor*

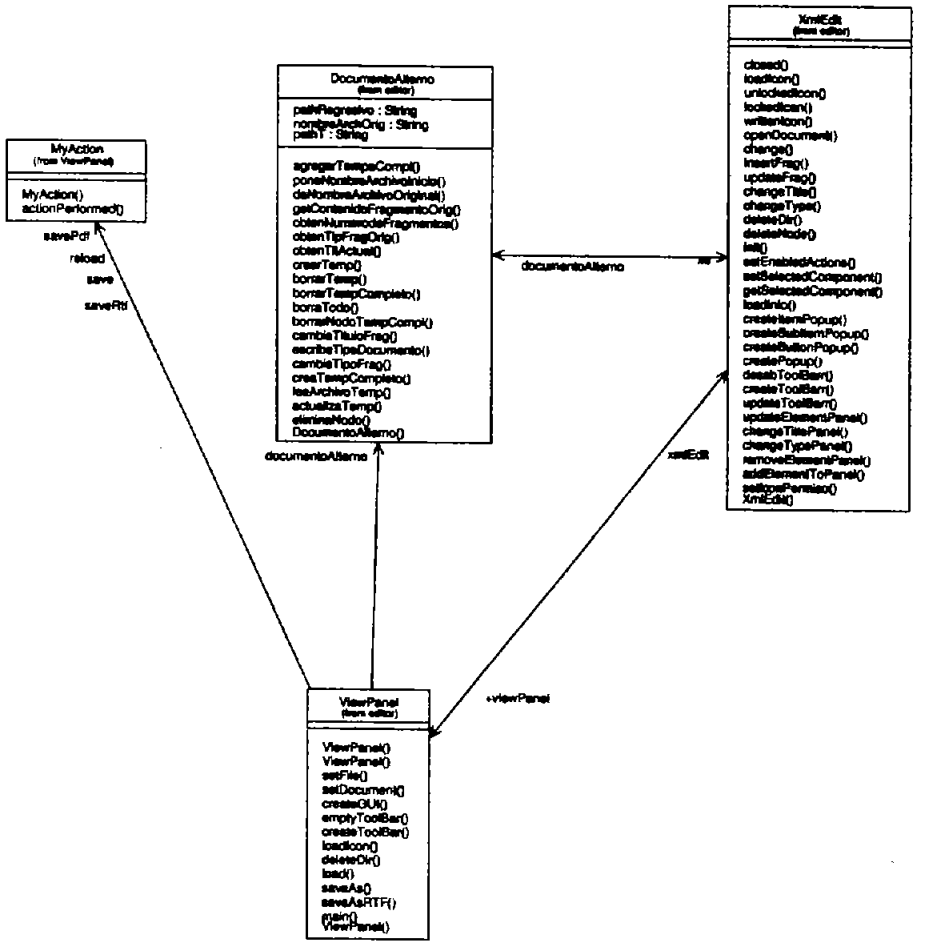


Figura 5.2: Relaciones de la clase **DocumentoAlterno** con otras clases

los cuales permiten la distribución de imágenes a través de los clientes por medio del servidor.

Cuando una imagen es insertada en el documento, ésta es primero copiada en directorios al mismo nivel donde se guarda el documento XML [17]. Esos directorios guardan una estructura acorde a los fragmentos y al documento en

edición. Por ejemplo, si una imagen *figura.jpg* es insertada en el fragmento 2 y esta misma figura a su vez es insertada en el fragmento 4 entonces se crearán los directorios: *dirInstalacionElxi/datos/usuario/grupodeTrabajo/ImagDoc_NombreDoc/2/figura.jpg*; y *dirInstalacionElxi/datos/usuario/grupodeTrabajo/ImagDoc_NombreDoc/4/figura.jpg* en los cuales se guardará una copia de estas imágenes. Con esto, se pueden tener referencias separadas para cada una de las imágenes contenidas en los fragmentos. Así, se logra administrar el manejo de estas imágenes cuando se borran o modifican.

Los métodos para guardar las imágenes, dentro de directorios, se implementaron en la clase **ViewPanel** dentro del botón de acción *ver* que permite la visualización del documento [17][18]. Es en esta acción del *escuchador*[2] donde se piden las imágenes que el servidor en ese momento tenga del documento. De tal forma que el servidor será el lugar donde se tenga la última actualización de las imágenes insertadas en el documento.

[2]Clase que se encarga de realizar una acción determinada al momento de ocurrir un evento.

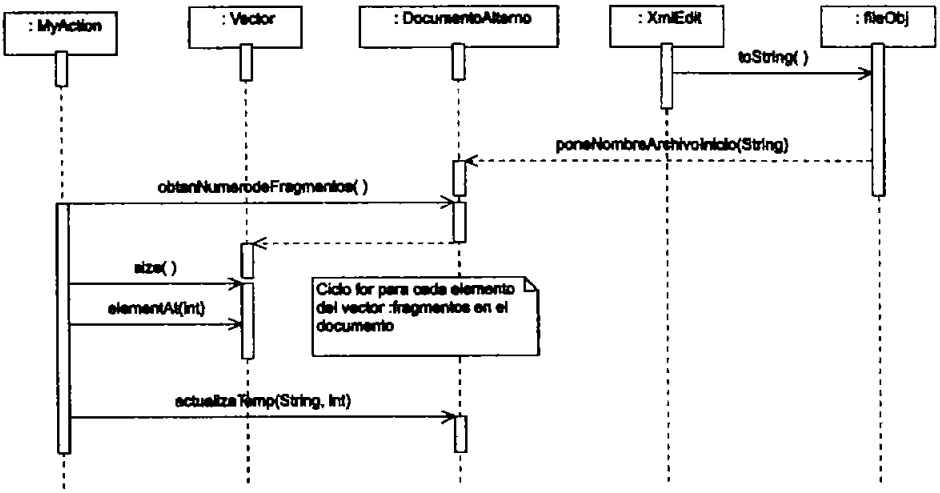


Figura 5.3: Armado del **Documento Alterno**

Para que el servidor pueda tener las imágenes actualizadas del documento, cada vez que se agreguen nuevas imágenes desde el cliente, deberán ser copiadas con la misma estructura de directorios, antes mencionada, al servidor. Si algún cliente realiza cambios en un fragmento que tenga en edición, éste será reflejado en el servidor. Esta acción conduce a que, si en ese momento algún otro cliente pide la visualización del documento, esta visualización se hará pidiendo las imágenes del documento con las imágenes que acaba de modificar el cliente que actualmente tiene la edición del fragmento. En la figura 5.4 se puede observar una imagen contenida en un fragmento de una Norma.

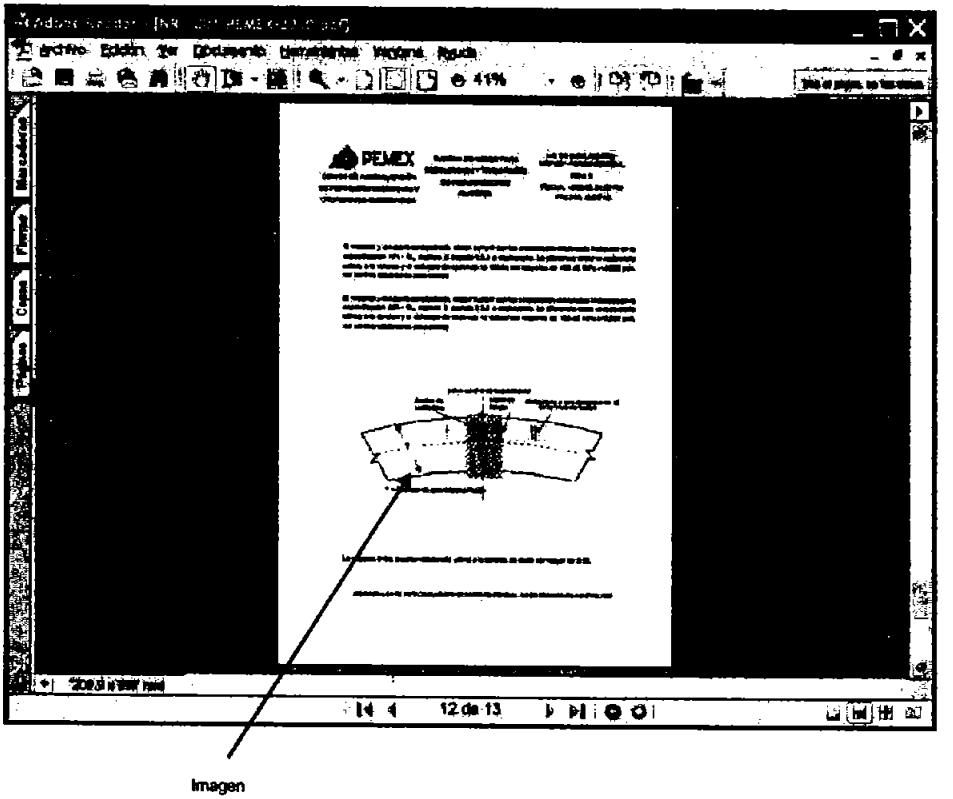


Figura 5.4: Imagen en un documento Norma

Los métodos implementados en el servidor son los siguientes:

public void uploadFile (byte[] datosArchivo, String nomubicacion, String dirServ,String dirImages,String idFrag) Permite *subir* archivos de un cliente al servidor, pasando como parámetros: los datos del archivo; el nombre del archivo; la dirección a donde se quiere subir en el servidor; la estructura de la dirección donde se agregará la imagen para ese fragmento; y el identificador del fragmento.

public byte[] downloadFile(String fileName, String dirDocServ) Baja una imagen del servidor, recibiendo un arreglo con los datos de la imagen y recibiendo como parámetros el nombre del archivo y el directorio del cual se desea bajar.

public String[] downloadListaFilesDir(String dirDocServ) Permite bajar la lista de archivos contenidos en un directorio, recibiendo como parámetro la ruta del directorio donde se quiere bajar la lista de archivos.

Con estos métodos, desde el cliente, es posible bajar los directorios de imágenes del servidor o subir los directorios de imágenes a éste. Por ejemplo si se necesitan bajar los directorios con las imágenes del documento para su visualización, primero se obtiene la lista de las imágenes del directorio de imágenes de un fragmento. Después con la lista de imágenes se bajan (con una estructura sencilla de control de tipo *for*) cada una de las imágenes. Con esta idea también, se implementó la transferencia de archivos hacia el servidor, sólo que ahora la transferencia se realiza en dirección contraria. La imagen a subir en el servidor se toma del cliente y se transfiere a directorio de imágenes del documento en edición.

Los formatos de imágenes soportados son aquellos que permite la máquina de transformación utilizada. Es decir, los tipos de imágenes soportados para los documentos **HTML** en **ELXI** son los formatos que permite insertar **XALAN**. De igual forma, los formatos soportados para los documentos **PDF** generados en **ELXI** son lo que permite insertar la máquina de transformación **FOP**. Para ambas máquinas los formatos soportados actualmente son GIF y JPEG.

El código para obtener la lista de nombres de archivos es el siguiente:

```
00 public String[] downloadListaFilesDir(String dirDocServ){
01
02     try {
03
04     String separador = System.getProperty("file.separator");
05     String rutaServ = System.getProperty("user.dir") + separador +
06                       "datos" + separador +"Imágenes"+ separador+
07                       dirDocServ+separador;
08     System.out.println("La ruta de donde bajará la lista de archivos es: "
09     +rutaServ);
10
11     File file = new File(rutaServ);
12     if (file.isDirectory()) {
13     System.out.println("La ruta pasada es un directorio y"+
14                        "obtendrá la lista de archivos");
15     String[] listaFiles = file.list();
16     return listaFiles;
17         }else{
18         System.out.println("La ruta pasada no es de un directorio");
19         String[] listaFiles = {};
20         return listaFiles;
21         }
22
23
24         } catch(Exception e){
25             System.out.println("FileImpl: "+e.getMessage());
26             e.printStackTrace();
27             return(null);
28         }
29     }
```

En la línea 5 se arma la ruta de las imágenes en el servidor. Con esta ruta en la línea 16 se obtienen los nombres de cada uno de los archivos contenidos en el directorio. Cabe mencionar que este código es parte de los métodos remotos implementados en el servidor con RMI de **ELXI**, los cuales son llamados desde los clientes para su ejecución.

Para la transferencia de archivos se tiene el siguiente código:

```
00 public byte[] downloadFile(String fileName, String dirDocServ){
01
02     try {
03
04         String separador = System.getProperty("file.separator")
05         String rutaServ = System.getProperty("user.dir") +
06                           separador + "datos"+ separador +
07                           "Imagenes"+ separador+dirDocServ+
08                           separador;
09             System.out.println("La ruta de donde bajará la imag
10                                 +rutaServ);
11             String rutaServCompl= rutaServ+fileName;
12             System.out.println("La ruta completa de donde bajar
13                                 "la imagen es: "+rutaServCompl);
14             File file = new File(rutaServCompl);
15             byte buffer[] = new byte[(int)file.length()];
16             BufferedInputStream input = new
17             BufferedInputStream(new FileInputStream(rutaServCom
18             input.read(buffer,0,buffer.length);
19             input.close();
20             return(buffer);
21         } catch(Exception e){
22             System.out.println("FileImpl: "+e.getMessage());
23             e.printStackTrace();
24             return(null);
25         }
26     }
```

Se observa, en la línea 5, de nuevo el armado de la ruta de imágenes en el servidor, la ruta completa con el nombre del archivo a bajar se construye en la línea 11. De la línea 14 a la 20 se obtiene el contenido del archivo y se regresa como salida del método.

5.1.2. Diseño e implementación de la visualización de documentos

Para la implementación de las hojas de estilo, en cada uno de los documentos, se usó XSL y XSL-FO. Estos lenguajes permiten generar la estructura de

transformación de un documento XML a otro tipo de formato diferente [17]. En la versión 1.0 de **ELXI**, se encontraba ya desarrollada la hoja de estilo para el manejo del documento **Norma**, en lo relativo al manejo de secciones y la presentación de cada una de éstas.

Las hojas de estilo XSL y XSL-FO permiten generar documentos **HTML** o **PDF** respectivamente. **RTF** es un formato de texto usado por muchos editores. Alimentando el motor de transformación XALAN [44] con el documento XML y la hoja de estilo XSL se obtiene un documento **HTML** formateado. Una de las opciones de configuración de XALAN es la generación de documentos **RTF**, recibiendo como entrada el documento XML y la misma plantilla **HTML**. Para **ELXI**, dentro de los requerimientos, se necesitaba la generación de documentos **RTF** con el fin de que el usuario pudiera abrir un documento generado en **ELXI** desde el editor **Word** de Microsoft. Por tanto, se implementó la generación de este formato en **ELXI**.

Como la plantilla XSL genera documentos **HTML** o **RTF**, de aquí en adelante utilizaremos la palabras **HTML** o **RTF** indistintamente, sabiendo que la generación del documento, ya sea **HTML** o **RTF**, se realiza con la misma hoja de estilo XSL.

Ahora bien, el nuevo código generado en las hojas de estilo, agregado a lo ya hecho [18] [17], se planteó como solución a la necesidad del manejo de listas, tablas, imágenes, pies de página, etc., y a la estructura del documento **General**. La implementación de los nuevos elementos de edición se presenta a continuación.

Listas

El lenguaje XSL hace un énfasis especial en el uso de la recursividad para la definición de nuevos objetos de flujo (*templates*). Así para cada nuevo elemento, se definió el código recursivo capaz de transformar elementos de edición anidados, definiendo recursivamente el manejo de los elementos de edición encontrados en el documento XML. Por ejemplo, en el caso de una lista, el código implementado para la visualización en **HTML** es:

```
00 <xsl:template match="lista">
01
02     <ul>
03     <ul>
04        <xsl:apply-templates select="elemento"/>
05     </ul>
06     </ul>
07
08 </xsl:template>
09
10 <xsl:template match="elemento">
11
12     <ul>
13         <li>
14             <xsl:value-of select="text()"/>
15             <xsl:apply-templates select="texto"/>
16             <xsl:apply-templates select="textoP"/>
17             <xsl:apply-templates select="lista"/>
18             <xsl:apply-templates select="listaN"/>
19             <xsl:apply-templates select="listaD"/>
20         </li>
21     </ul>
22
23 </xsl:template>
```

Un objeto de flujo (*template*)[3] define qué formato se le va a dar a los datos que estén contenidos dentro de esta *etiqueta de marca* en el documento XML cuando sea encontrado [17]. Para nuestros fines, lo que se desea es generar el código **HTML** que permita la visualización de una lista en una página dentro de un navegador. Por tanto, se agregan las respectivas marcas **HTML** para la generación de listas (ver líneas 12, 13, 20, 21).

Luego de esto, se define la aplicación de una *etiqueta de marca* para el elemento *elemento de lista* (línea 10). Después, se relaciona esta *etiqueta de marca* con el *objeto de flujo* de *lista* en el XSL para la visualización del documento (línea 04). Con esto se logra que una *lista* este formada por *elementos de lista*.

[3]Del término en inglés *template*: plantilla, patrón, machote.

Dentro del *objeto de flujo* para *elemento de lista*, se aplica el objeto de flujo de *texto* junto con los objeto de flujo correspondientes a los demás elementos de tipo lista (líneas 14 a 19).

Como se observa, los *objetos de flujo* de los diferentes tipos de listas se aplican de forma recursiva para permitir la generación de listas anidadas. El lenguaje XSLT es un lenguaje declarativo, el cual permite manejar de forma robusta la implementación recursiva en sus declaraciones.

Para el caso de la hoja de transformación, en el documento **PDF**, la idea es la misma, sólo que con una sintaxis diferente relacionada con la estructura de comandos propios de un documento **PDF**. Por ejemplo, a diferencia de la implementación para XSL, en XSL-FO se debe dar una indentación adecuada a cada una de las sublistas que se pueden anidar. Para lograr esto, se debe hacer una pequeño cálculo para permitir las distancias de indentación. Cabe aquí mencionar que el manejo de variables permitido por XSL o XSL-FO son en realidad asignaciones de valores que no cambian en el transcurso de la ejecución, sólo se reasignan nuevamente al volver a llamar al *objeto de flujo*.

Para implemetar con XSL-FO las lista anidadas, se hace uso de la recursividad. Al definir el espacio para la indentación y la figura apropiada de la lista (bala, número, etc.), se llama de forma recursiva el *objeto de flujo* que resuelve de nuevo una lista. En la figura 5.5, en la página 117, se puede observar una lista de elementos dentro una **Norma** en formato **PDF**.

Existen tres tipos de listas en **ELXI**, lista, lista con balas y lista numerada. De esta forma, para cada tipo de lista se implementó un código muy similar, difiriendo sólo en las figuras para cada elemento de la lista: balas, numeración o ninguna figura.

El código para el caso de la lista con *balas* en XSL-FO, se muestra a continuación:

```
00 <xsl:template match="lista">
01   <fo:list-block provisional-distance-between-starts="1cm"
02     provisional-label-separation="0.5cm">
03     <xsl:attribute name="space-after">
```

```
04        <xsl:choose>
05          <xsl:when test="ancestor::lista">
06            <xsl:text>0pt</xsl:text>
07          </xsl:when>
08          <xsl:otherwise>
09            <xsl:text>12pt</xsl:text>
10            </xsl:otherwise>
11        </xsl:choose>
12      </xsl:attribute>
13      <xsl:attribute name="start-indent">
14        <xsl:variable name="ancestors">
15          <xsl:choose>
16            <xsl:when test="count(ancestor::lista)">
17              <xsl:value-of select="1 +
18                                    (count(ancestor::lista) *
19                                    1.25"/>
20            </xsl:when>
21            <xsl:otherwise>
22              <xsl:text>1</xsl:text>
23            </xsl:otherwise>
24          </xsl:choose>
25        </xsl:variable>
26        <xsl:value-of select="concat($ancestors, 'cm')"/>
27      </xsl:attribute>
28      <xsl:apply-templates select="*"/>
29    </fo:list-block>
30 </xsl:template>
31
32 <xsl:template match="lista/elemento">
33    <fo:list-item>
34      <fo:list-item-label end-indent="label-end()">
35        <fo:block>
36          &#x2022;
37        </fo:block>
38      </fo:list-item-label>
39      <fo:list-item-body start-indent="body-start()">
40        <fo:block>
41          <xsl:apply-templates select="*|text()"/>
```

```
42        </fo:block>
43      </fo:list-item-body>
44    </fo:list-item>
45 </xsl:template>
```

En la línea 03, se define un atributo *space-after*, el cual puede tomar varios valores con el fin de indentar los primeros elementos de la lista. De igual forma, en la línea 13 se define *start-ident* con el fin de indentar los elementos de la lista, pero ahora, para elementos posiblemente anidados. En XSL-FO una lista se arma con las etiquetas de marca *list-block* y *list-item*[4]. En la línea 28, se pide la aplicación de los templates para las etiquetas de marca contenidos en la lista —list-block—, en este caso *elemento*.

Por último, en la línea 32, para la resolución de elementos *list-item* de XSL-FO se define la etiqueta y su formato de presentación. En la línea 41, se pide la presentación del contenido de las etiquetas de marca *elemento*.

Tablas

La implementación para el *template* que permite el manejo de un nuevo elemento **tabla** con las *etiquetas de marca <tabla>, <renglon>, <celda>* es la siguiente:

```
00 <xsl:template match="tabla">
01
02    <table border="1" align="center">
03        <xsl:apply-templates select="renglon"/>
04    </table>
05
06 </xsl:template>
09
10 <xsl:template match="renglon">
11
12    <tr>
13        <xsl:apply-templates select="celda"/>
```

[4]Una lista —*list-block*— contiene —*list-item*—.

```
14    </tr>
15
16 </xsl:template>
17
20 <xsl:template match="celda">
21
22    <th>
23        <xsl:value-of select="text()"/>
24        <xsl:apply-templates select="imagen"/>
25        <xsl:apply-templates select="lista"/>
26        <xsl:apply-templates select="tabla"/>
27        <xsl:apply-templates select="texto"/>
28        <xsl:apply-templates select="textoP"/>
29    </th>
30
31 </xsl:template>
```

De forma semejante que en las listas, se manejó recursividad para poder agregar elementos anidados a la tabla. Una tabla puede contener, dentro de cada una de sus celdas, imágenes, listas, texto, texto sin formatear o tablas mismas (líneas 24-28); donde cada uno de estos elementos mencionados puede anidar otros elementos, si le es permitido, en la definición de la DTD. Al final de este capítulo, se muestra la DTD completa generada en este trabajo.

Para el caso de tablas en **HTML**, se asignan las *etiquetas de marca* correspondientes de una tabla en **HTML**. Para ir formando la estructura deseada de la tabla, cada llamada a un *objeto de flujo*, en la forma correcta, armará en **HTML** una tabla que pueda ser visualizada en una página de un navegador (líneas 02, 04, 12, 14, 22, y 29). En el código se ve de forma clara (líneas 03, 13) que el *objeto de flujo tabla* busca etiquetas de marca *renglones*, que a su vez, contienen *celdas*. En la figura 5.6, se puede observar una tabla en un documento **Norma** en formato **RTF**. Más adelante, en el siguiente capítulo (ver sección 6.2), se mostrará la forma de generar elementos en nuestro lenguaje de marcas creado.

Debido a la complejidad misma de la estructura de un documento **PDF**, la implementación en XSL-FO fue más elaborada. Así, por ejemplo, para de-

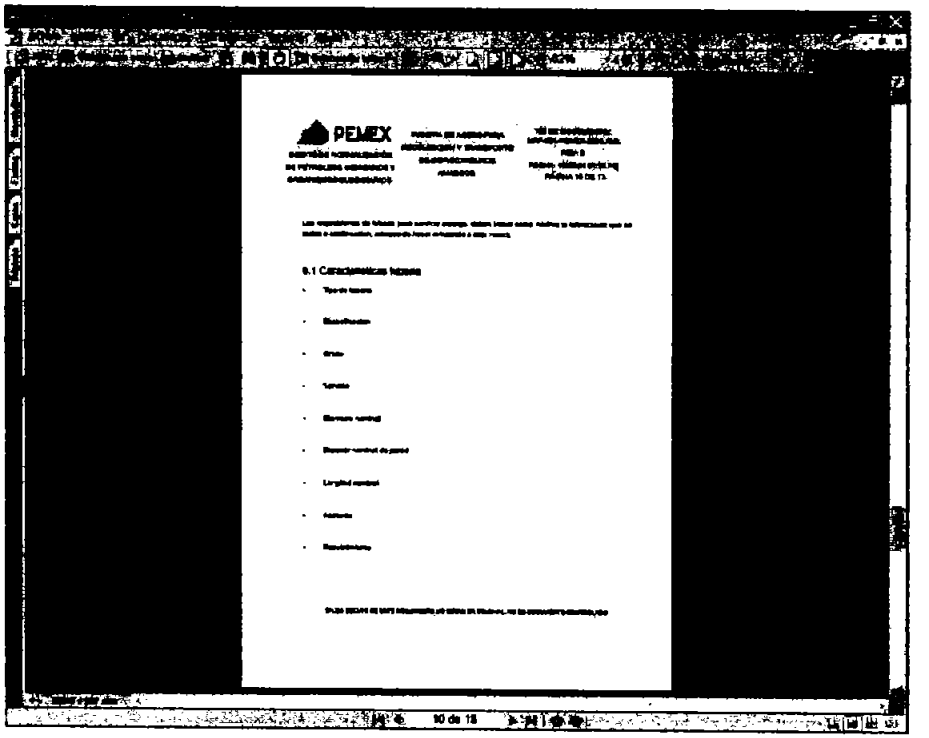
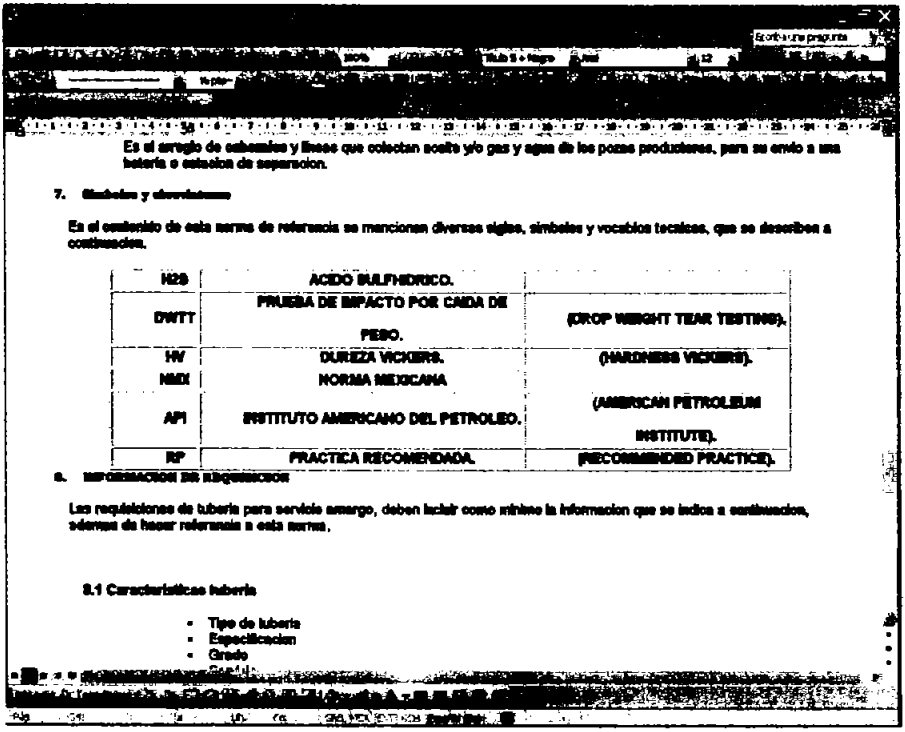


Figura 5.5: Lista en un documento **Norma** PDF

finir el número de columnas, éstas debían de ser creadas de forma dinámica para cada una de las tablas. De igual forma, el ancho de cada columna debía de ser fijado para la construcción de la tabla y ser dado dinámicamente al documento que se estaba creando. En el código mostrado —en el apéndice A— para la creación de tablas, se puede observar el incremento de líneas de código, debido al aumento de complejidad. Por ejemplo, en formato **PDF** se llama a un objeto de flujo *template* independiente para armar el número de columnas y su ancho. El llamado a otros *templates*, sin ser parte de la estructura del XML, puede ser logrado con la instrucción *xsl:call-template* [17].



Es el arreglo de cabezales y líneas que colectan aceite y/o gas y agua de los pozos productores, para su envío a una batería o estación de separación.

7.

Es el contenido de esta norma de referencia se mencionan diversas siglas, símbolos y vocablos técnicos, que se describen a continuación.

H2S	ACIDO SULFHIDRICO.	
DWTT	PRUEBA DE IMPACTO POR CAIDA DE PESO.	(DROP WEIGHT TEAR TESTING).
HV	DUREZA VICKERS.	(HARDNESS VICKERS).
NMX	NORMA MEXICANA	
API	INSTITUTO AMERICANO DEL PETROLEO.	(AMERICAN PETROLEUM INSTITUTE).
RP	PRACTICA RECOMENDADA.	(RECOMMENDED PRACTICE).

8.

Las requisiciones de tubería para servicio amargo, deben incluir como mínimo la información que se indica a continuación, además de hacer referencia a esta norma.

8.1 Características tubería

- Tipo de tubería
- Especificación
- Grado

Figura 5.6: Tabla en un documento Norma RTF

Las estructuras de control del lenguaje son usadas para decidir los posibles atributos de la tabla. Con dichos atributos se pueden alinear los textos contenidos en las celdas, el borde de la líneas de la tabla, la alineación de nuevas tablas anidadas, y otros formatos de presentación. Estas estructuras de control (if, else, while, etc.) permiten el recorrido a través de los *objetos de flujo* de cada uno de los elementos, asignandose valores predeterminados para los casos donde no se contemple valor por asignar.

Para la generación de tablas en **PDF** se puede observar el código en el Apéndice A.

Texto

La *etiqueta de marca* tipo *texto* permite agregar, más de un bloque de texto por fragmento. La implementación de este elemento definió principalmente estructuras de control que permitieran diferenciar los tipos de texto dentro del documento (texto de los elementos de una lista, texto de los elementos de una tabla, texto de los autores de la **Norma**, y texto sin formato). Para cada tipo de documento, **HTML** o **PDF**, se implementaron diferentes flujos de ejecución. Para el caso de texto manejado en **HTML** se tiene:

```
00 <xsl:template match="texto">
01
02 <xsl:choose>
03
04 <xsl:when test="ancestor::fragmento[@tipo='autores']">
05
06 <p align="center">
07       <b>  <xsl:apply-templates/></b>
08 </p>
09
10 </xsl:when>
11
12 <xsl:otherwise>
13
14 <ul> <p align="justify">
15         <xsl:apply-templates/>
16     </p>
17 </ul>
18
19 </xsl:otherwise>
20
21 </xsl:choose>
22
23 </xsl:template>
```

Sólo se considera la posibilidad de un texto diferente para los autores (línea 04), el cual debe de estar centrado (línea 06) y debe tener el cargo de la persona dependiendo del número de fragmento de tipo *autor* agregados al

documento (ver figura 5.7, página 122). Para los demás casos, la alineación del texto es justificada (líneas 12 y 14).

En cambio, para el elemento texto, dentro del documento **PDF** se debe asignar el formato para cada uno de los textos en los diferentes elementos que lo contengan (subsecciones, tablas, listas, pies de páginas, etc.). En el Apéndice A, se muestra el manejo del formato de texto en cada uno de los tipos de fragmentos.

El caso de texto sin formato también debió ser considerado. Se necesitaba en momentos específicos la visualización de texto de forma idéntica a la que se escribía dentro de un fragmento (por ejemplo, al escribir la dirección de una persona en una **Norma**). Para lograr esto, se agregaron las *etiquetas de marca* específicas de estilo **HTML** y **PDF** para visualizar de forma deseada el texto a visualizar (línea 12). El caso en el cual se usa texto sin formato (en el tipo de fragmento *autores*, línea 03) se consideró por separado, debido a que este tipo de fragmento no necesitaba indentación como en cualquier otro caso donde sí se identa el texto —línea 10 y 13—.

Para el texto sin formato en **HTML** se tiene el siguiente código:

```
00 <xsl:template match="textoP">
01 <xsl:choose>
02 <xsl:when
03 test="ancestor::fragmento[@tipo='autores']">
04
05 <pre style="font-family: times new roman,times,serif;">
06         <xsl:apply-templates/>
07 </pre>
08 </xsl:when>
09
10 <xsl:otherwise>
11
12 <pre style="font-family: times new roman,times,serif;">
13         <ul><xsl:apply-templates/></ul>
14 </pre>
```

```
15
16 </xsl:otherwise>
17 </xsl:choose>
18 </xsl:template>
```

Y para el texto sin formato en **PDF** se tiene el siguiente código:

```
00 <xsl:template match="textoP">
01     <xsl:choose>
02     <xsl:when test="ancestor::fragmento[@tipo='autores']">
03         <fo:block
04         space-after="10pt" start-indent=".1cm"
05         font-family="sans-serif"
06         white-space-collapse="false"
07         wrap-option="no-wrap">
08         <xsl:apply-templates/>
09         </fo:block>
10     </xsl:when >
11
12     <xsl:otherwise>
13         <fo:block
14         space-after="10pt" start-indent="1cm"
15         font-family="sans-serif"
16         white-space-collapse="false"
17         wrap-option="no-wrap">
18         <xsl:apply-templates/>
19         </fo:block>
20     </xsl:otherwise>
21     </xsl:choose>
22 </xsl:template>
```

De la línea 02 a la 10 se observa la estructura de control para manejar el formato del tipo *autores*, de la 12 a la 20 para el texto normal con 1 cm de indentación.

Secciones y subsecciones

Para poder visualizar bloques de texto dentro de fragmentos, se implementó también la estructura de presentación de estos bloques. Se generaron

las *etiquetas seccion*, *subseccion*, *subDos* y *subTres*. Estos permiten la visualización de los bloques de forma identada, así como la asignación de títulos o encabezados en cada una de estas secciones.

En la interfaz de **ELXI** aparece un árbol que permite ver la estructura del documento que se ésta editando —ver fig 3.12— así como el grupo de trabajo al cual pertenece el documento [18]. Cada fragmento en este árbol se identifica por un título, el cual, al ser seleccionado, permite ubicar el fragmento dentro del editor. Este título puede ser editado, por lo cual este texto

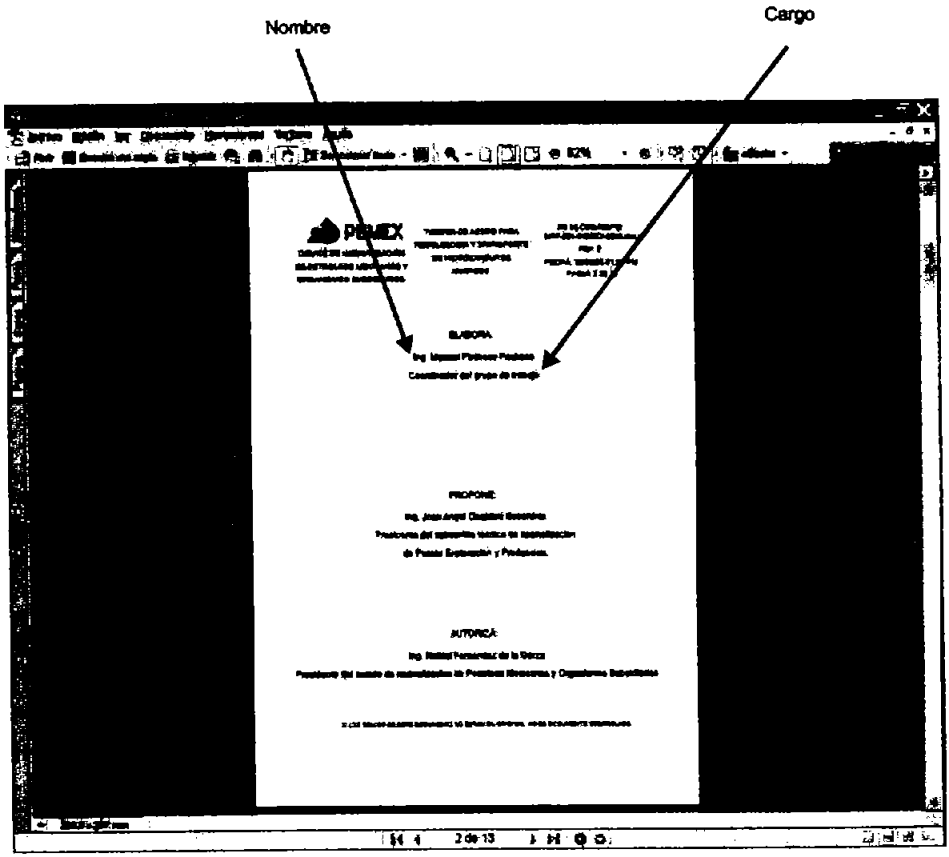


Figura 5.7: Texto de tipo *Autores*

fue asignado como encabezado de cada sección o subsección al momento de visualizar el documento. Para cada tipo de sección, se formateó la presentación en tamaños y tipos de letra con relación a las secciones o subseciones, y además se identó cada uno de los bloques de texto para una presentación correcta.

Las secciones y subsecciones en **HTML** se implementaron de la forma siguiente:

```
00 <xsl:template
01 match="fragmento[@tipo='section']|fragmento[@tipo='subsection']|
02       fragmento[@tipo='subDos']|fragmento[@tipo='subTres']">
03   <div>
04     <xsl:if test="@tipo='section'">
05     <h3 font="bold">
06      <pre style="font-weight: bold; font-family: bookman old style;">
07      <xsl:value-of select="@titulo"/></pre>
08     </h3>
09     <xsl:apply-templates/>
10     </xsl:if>
11
12     <xsl:if test="@tipo='subsection'">
13      <br/>
14       <left><font size="3">
15       <b>
16         <div style="margin-left: 80px;">
17         <xsl:apply-templates select="@titulo"/>
18         </div>
19       </b></font></left>
20       <p align="left" >
21       <div style="margin-left: 80px;">
22       <xsl:apply-templates/>
23       </div></p>
24     </xsl:if>
25
26     <xsl:if test="@tipo='subDos'">
27         <br/>
```

```
28        <left><font size="3"><b>
29          <div style="margin-left: 120px;">
30          <xsl:apply-templates select="@titulo"/>
31          </div>
32        </b></font></left>
33        <p align="left" >
34        <div style="margin-left: 160px;">
35        <xsl:apply-templates/>
36        </div></p>
37      </xsl:if>
38
39      <xsl:if test="@tipo='subTres'">
40          <br/>
41        <left><font size="3"><b>
42          <div style="margin-left: 160px;">
43          <xsl:apply-templates select="@titulo"/>
44          </div>
45        </b></font></left>
46        <p align="left" >
47        <div style="margin-left: 200px;">
48        <xsl:apply-templates/>
49        </div></p>
50      </xsl:if>
51
52    </div>
53 </xsl:template>
```

En la líneas 04, 12, 26, y 39, se puede observar el manejo de formato para cada uno de los tipos de subsecciones definidas. En la línea 14 se observa el tamaño asignado al título de una subsección tomado del árbol de **ELXI** por el código de la línea 17, para el caso de una subsección. La indentación que se observa en las líneas 21, 34, y 47 va incrementandose respecto a los tipos de sección.

El título, que aparece en el árbol de documentos de **ELXI** (ver figura 5.8), se pudo obtener como atributo de un fragmento de la estructura del **Documento Alterno**.

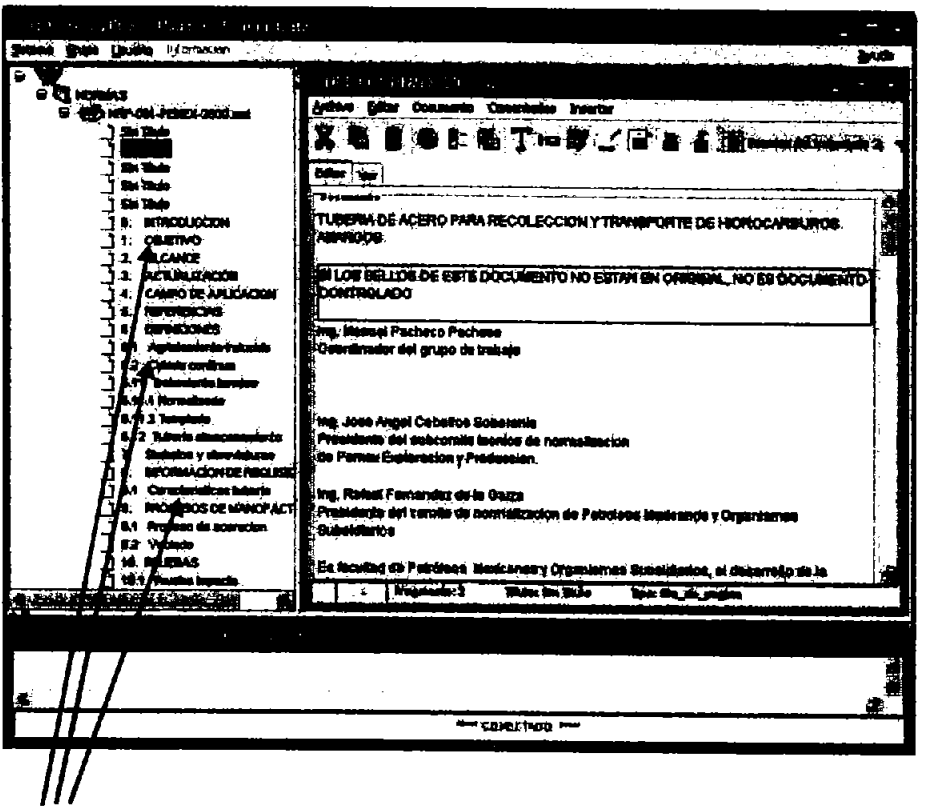


Figura 5.8: Árbol de documentos en ELXI

Para el caso de la implementación de las secciones y subsecciones en el documento **PDF**, nos podemos referir al Apéndice A.

Código embebido en fragmentos

Cada vez que se necesite la visualización de un documento, el analizador sintáctico buscará primero, dentro de cada fragmento, una *etiqueta de marca* llamada **DocXMLEmbebido**. Una vez ubicada esta etiqueta se podrán resolver los elementos de edición (tablas, listas, imágenes, etc.) que hasta ahora hemos venido implementando.

El contenido de esta *etiqueta de marca* **DocXMLEmbebido**, en la estructura del documento XML, se toma sólo como texto durante la distribución del documento. Al llegar al cliente, este contenido se transforma en código XML, con el cual se forma el **Documento Alterno** completo.

Se muestra a continuación parcialmente (por cuestiones de espacio) el documento XML de un documento **Norma**, en el que se pueden observar varias *etiquetas de marca* para las cuales hemos venido implementado código XSL. Además se puede observar la estructura del documento referente a fragmentos y documentos XML embebidos *DocXMLEmbebido*.

```
<?xml version="1.0" encoding="UTF-8"?> <!--Documento Alte-->
<?xml-stylesheet type="text/xsl" href="Norma de Referencia.xsl"?>
<!DOCTYPE documento SYSTEM "documento.dtd">

<documento id_file="376" tipo_f="Norma de Referencia">

<fragmento id_fragment="1" tipo="titulo" titulo="Sin Titulo">

<DocXMLEmbebido>

<texto>TUBERÍA DE ACERO PARA RECOLECCIÓN Y TRANSPORTE DE
HIDROCARBUROS AMARGOS </texto>

</DocXMLEmbebido>
</fragmento>

<fragmento id_fragment="2" tipo="Pie_de_pagina"titulo="Sin
Titulo"> <DocXMLEmbebido> <texto>SI LOS SELLOS DE ESTE DOCUMENTO
NO ESTÁN EN ORIGINAL, NO ES DOCUMENTO CONTROLADO </texto>
</DocXMLEmbebido> </fragmento>

<fragmento id_fragment="3" tipo="autores" titulo="Sin Titulo">
<DocXMLEmbebido> <texto>Ing. Manuel Pacheco Pacheco </texto>
<texto>Coordinador del grupo de trabajo </texto>
</DocXMLEmbebido>
</fragmento>
```

<fragmento id_fragment="4" tipo="autores" titulo="Sin Titulo">
<DocXMLEmbebido> <texto>Ing. José Angel Ceballos Soberanis</texto>
<texto>Presidente del Subcomité Técnico de Normalización </texto>
<texto>de Pemex Exploración y Producción. </texto>
</DocXMLEmbebido> </fragmento>

<fragmento id_fragment="5" tipo="autores" titulo="Sin Titulo">
<DocXMLEmbebido> <texto>Ing. Rafael Fernandez de la Garza </texto>
<texto>Presidente del Comité de Normalización de Petróleos
Mexicanos y Organismos Subsidiarios </texto> </DocXMLEmbebido>
</fragmento>

<fragmento id_fragment="6" tipo="introduccion" titulo="0.
INTRODUCCION"> <DocXMLEmbebido> <texto>Es facultad de Petróleos
Mexicanos y Organismos Subsidiarios, el desarrollo de la
normatividad técnica que garantice la calidad de los materiales e
instalaciones, a fin de que estas operen de manera eficiente,
segura y se manifieste en la preservación de vidas humanas, medio
ambiente e instalaciones. </texto> <texto>Es responsabilidad del
usuario considerar las características operativas de la tubería y
su correlación con la normatividad de construcción y mantenimiento
de ductos, a fin de establecer en la requisición, las
características normativas y de metrología de la ingeniería de
diseño resultante, así como requisitos adicionales de inspección y
pruebas si el proyecto en particular lo requiere. Este documento
normativo se realizó en atención y cumplimiento a: </texto>
<lista> <elemento>La Ley Federal sobre Metrología y Normalización
</elemento> <elemento>La Ley de Adquisiciones, Arrendamientos y
Servicios del Sector Público </elemento> <elemento>La
ley...</elemento> <elemento>La ley...</elemento> </lista>

<texto>Participaron en su elaboración las direcciones de Petróleos
Mexicanos, Instituciones, empresas y consultores técnicos, que se
indican a continuación: <lista> <elemento>Pemex Exploración y
Producción </elemento> <elemento>Pemex Refinación </elemento>
<elemento>Pemex Gas y Petroquímica Básica</elemento>
<elemento>Pemex Petroquímica</elemento> <elemento>Petróleos

```
Mexicanos </elemento> <elemento>IMP</elemento>
<elemento>Corporación química de productores </elemento>
<elemento>Canacero S.A</elemento> <elemento>Floban S.A.</elemento>
<elemento>Dr. Juan Hernández F. </elemento> </lista> <texto>
</texto> <texto> </texto> </DocXMLEmbebido> </fragmento>

<fragmento id_fragment="7" tipo="objetivo" titulo="1. OBJETIVO">
<DocXMLEmbebido> <texto>Esta norma de referencia establece los
requisitos mínimos de calidad en la fabricación, inspección y
pruebas, de tubería de acero microaleado para la recolección y
transporte de hidrocarburos amargos. </texto> <texto> </texto>
<texto> </texto> <texto> </texto> </DocXMLEmbebido> </fragmento>
...
```

En la versión 1.0 de **ELXI** la estructura de la **Norma** se basó, primero, en el orden en que deben de aparecer cada una de las secciones del documento —Título, Concordancia con otras normas, Autores, etc.—. La anterior fue una buena solución para seguir el orden de estructura que marcaba la **Norma** [17], pero ahora, dentro de cada fragmento, los nuevos elementos (*etiquetas de marca*) tendrán la libertad de ser asignados en el orden que se desee y las secciones de la **Norma** tendrán que permitir la inserción de nuevos tipos de fragmento entre ellas. Podría ir, por ejemplo, una lista antes de algún texto, o una tabla después de tres imágenes si se deseara o también insertar un tipo de fragmento *libre* entre dos secciones de la **Norma**.

Para asignar un orden a cada uno de los fragmentos, **ELXI** implementaba en la versión 1.0 [18] [17] el orden que la guía de elaboración de Normas [19] define, lo cual fue útil con las necesidades planteadas para esa versión de **ELXI**.

Una **Norma** debe tener los tipos de fragmentos siguientes [17]:

- Introducción
- Objetivo
- Alcance
- Campo de aplicación

- Actualización
- Referencias
- Definiciones
- Símbolos y abreviaturas
- Secciones específicas al tipo de Norma
- Responsabilidades
- Concordancia con otras normas
- Bibliografía
- Anexos

Este orden fue el implementado y es el orden en el que aparecen los fragmentos de cada **Norma** visualizada. En la parte de la **Norma** *Sección, subsección* (ver línea 12 en el código), es donde se permite la asignación de nuevos fragmentos y es hasta aquí donde aparecería el texto añadido, por ejemplo, a una sección de tipo *Alcance*. Esto por la razón de que sólo en ese lugar es permitido agregar subsecciones o secciones.

La estructura de control para la presentación de los fragmentos de la **Norma** en la versión 1.0 de **ELXI** fue:

```
00 <xsl:apply-templates select="fragmento[@tipo='introduccion']"/>
01 <xsl:apply-templates select="fragmento[@tipo='objetivo']"/>
02 <xsl:apply-templates select="fragmento[@tipo='alcance']"/>
03 <xsl:apply-templates select="fragmento[@tipo='campo de
04 aplicacion']"/> <xsl:apply-templates
05 select="fragmento[@tipo='actualizacion']"/>
06 <xsl:apply-templates
07 select="fragmento[@tipo='referencias']"/>
08 <xsl:apply-templates
09 select="fragmento[@tipo='definiciones']"/>
10 <xsl:apply-templates
11 select="fragmento[@tipo='simbolos y abreviaturas']"/>
```

```
12 select="fragmento[@tipo='section']|fragmento[@tipo='subsection']
13 <xsl:apply-templates
14 select="fragmento[@tipo='responsabilidades']"/>
15 <xsl:apply-templates select="fragmento[@tipo='concordancia con
16 otras normas']"/>
17 <xsl:apply-templates
18 select="fragmento[@tipo='bibliografia']"/>
19 <xsl:apply-templates
20 select="fragmento[@tipo='anexos']"/>
```

La nueva estructura de control implementada es:

```
00 <xsl:apply-templates select=" fragmento[@tipo='agradecimientos']
01 fragmento[@tipo='conclusiones']| fragmento[@tipo='resultados']|
02 fragmento[@tipo='anexos']| fragmento[@tipo='bibliografia']|
03 fragmento[@tipo='concordancia con otras normas']|
04 fragmento[@tipo='responsabilidades']|fragmento[@tipo='simbolos
05 yabreviaturas']|fragmento[@tipo='definiciones']|
06 fragmento[@tipo='referencias']|fragmento[@tipo='actualizacion']|
07 fragmento[@tipo='campo de aplicacion']|
08 fragmento[@tipo='alcance']| fragmento[@tipo='objetivo']|
09 fragmento[@tipo='introduccion']| fragmento[@tipo='section']|
10 fragmento[@tipo='subsection']| fragmento[@tipo='subDos']|
11 fragmento[@tipo='subTres']"/>
```

Ahora, cada fragmento no tiene necesariamente el orden de la **Norma**, debido a la condición *or* (|) de secuencia de tipos de fragmentos (línea 00), pero permite asignar a cada uno de los fragmentos secciones o subsecciones que aparecerán en el lugar correcto. Esta forma de definir el orden será muy útil en la implementación del documento **General**, ya que permite la agregación de cualquier tipo de fragmento en cualquier lugar del documento a visualizar.

Encabezados y pies de página

El *encabezado* no se implementó como un tipo de fragmento en una **Norma**, ya que éste es especificado directamente, para este tipo de documento, en la hoja de estilo [17]. El *encabezado* es parte ya de la misma **Norma**. Si

se quisiera editar y visualizar otro tipo de documento en **ELXI**, por ejemplo una factura, la hoja de estilo XSL definiría partes especificas de la factura, como por ejemplo, el área para el RFC, la dirección, monto, etc. De este modo, para una **Norma**, este elemento está implementado directamente en la hoja de estilo XSL. El nombre del documento y la fecha son tomados del **Documento Alterno** de **ELXI** y del sistema respectivamente. En la figura 5.9, se puede observar la presentación del encabezado en una **Norma** visualizada por un navegador.

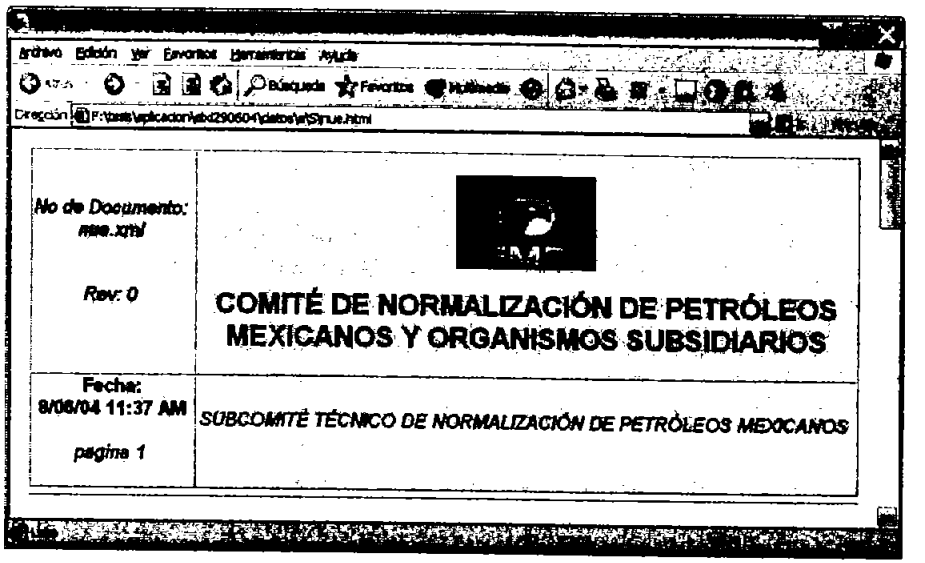


Figura 5.9: Encabezado en una Norma

En el pie de página de una **Norma** se escribe un enunciado referente a la validez del documento (ver figura 5.10). La implementación de los pies de página se realizó, como un tipo de fragmento, para el documento en formato **PDF**. En **HTML** no existen pies de página puesto que una página **Web** tiene un presentación sin paginación.

Un pie de página puede contener cualquier elemento de edición (listas, tablas, imágenes, texto, etc.) que se quiera agregar. Esto no es necesario

para una **Norma**, pues, con solo visualizar el texto en el pie de página es suficiente. Sin embargo, se implementó también la posibilidad de que un pie de página en una **Norma** pudiera contener cualquier elemento de edición. Esto debido a que esta implementación fue útil para el documento de tipo **General**.

Se muestra el código para ubicar el pie de página en el documento. Se observa la palabra *static* y *xsl-region-after* en la línea 0, esto permite la ubicación del pie de página en la parte más baja de cada hoja del documento.

```
0 <fo:static-content flow-name="xsl-region-after">
1       <fo:block text-align="center" font-size="8pt">
2       <xsl:apply-templates select=
3       "fragmento[@tipo='Pie_de_pagina']/DocXMLEmbebido" />
4 </fo:block>
```

Una *nota a pie de página*, en un documento **HTML**, no se puede agregar como tal, ya que solo existe una sola página. Sin embargo, se puede agregar una liga desde alguna parte del documento hasta el final de la página **HTML**. Estas referencias se implementaron a través de una referencia numerada consecutiva (línea 02 y 03). Así, cuando se agrega una nota a pie de página, ésta será referenciada al final de la página **HTML**.

El *objeto de flujo* implementado es el siguiente:

```
00 <xsl:template match="notaAPie" mode="procesa.notaAPie">
01
02 <br/> <xsl:variable name="indice"> <xsl:number level="any"
03 count="notaAPie"/>  </xsl:variable>
04
05 <a name="pdp_{$indice}" href="#{$indice}" >
06     [<xsl:value-of select="$indice"/>]
07 </a><xsl:apply-templates/><br/>
08
09 </xsl:template>
```

Este *objeto de flujo* se ubicó justo debajo de las llamadas a todos los *objetos de flujo* del documento (página 130), con el fin de que las referencias creadas se liguen hasta la parte más baja de la página.

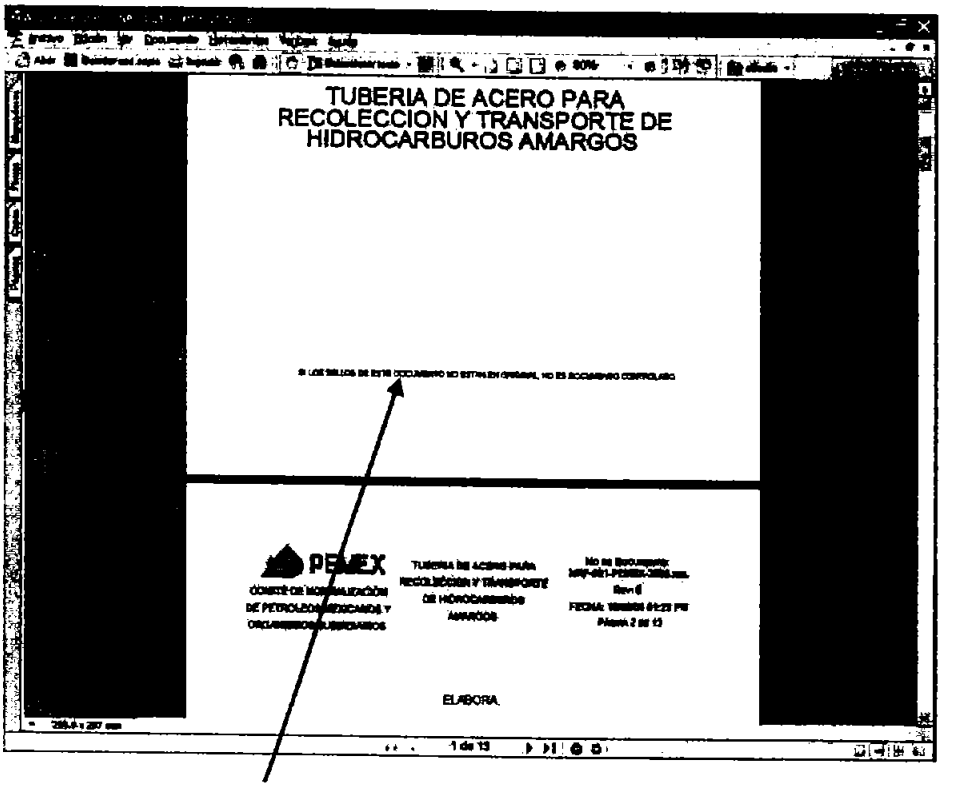


Figura 5.10: Pie de página en una **Norma**

Para crear estas referencias, dentro de cada fragmento, se define una nueva *etiqueta de marca* llamada *notaAPie*, la cual, igual que las listas o tablas, puede ser ubicada dentro del documento XML embebido en los fragmentos del documento. Se puede observar (líneas 06 – 08) la definición de referencias cruzadas para obtener las ligas de la nota con el texto referenciado. El *objeto de flujo* es el siguiente:

```
00 <xsl:template match="notaAPie">
01
02     <xsl:variable name="indice">
```

```
03    <xsl:number level="any" count="notaAPie"/>
04    </xsl:variable>
05
06    <a name="{$indice}" href="#pdp_{$indice}" >
07    [<xsl:value-of select="$indice"/>]
08    </a>
09
10 </xsl:template>
```

Imágenes

Las imágenes, como se explicó en la sección 5.1.1, se ubicaron en directorios específicos, por lo que la dirección del directorio de imágenes de un documento es transferido como variable de entrada al documento XSL. Además de que, para cada imagen, en el documento XML, se guarda su posición relativa del directorio del fragmento que le corresponde. De esta forma, la implementación del *objeto de flujo* para las imágenes es la siguiente.

```
00 <xsl:template match="imagen">
01   <fo:block font-size="14pt"
02     font-family="sans-serif"
03     line-height="18pt"
04     space-before.optimum="30pt"
05     space-after.optimum="5pt"
06     text-align="center">
07      <xsl:variable name="n" select="."/>
08      <xsl:value-of select="concat($rutaParaImag,$n)"/>
09
10      <fo:external-graphic src="{concat($rutaParaImag,$n)}"
11      content-height="1em" content-width="1em" />
12   </fo:block>
13 </xsl:template>
```

Se puede observar como se concatenan las rutas del directorio de imágenes que fueron pasadas al XSL, con la dirección relativa del documento XML (línea 08). Una vez formada la dirección completa, se pide la visualización de la imagen (línea 10 y 11).

Índices

Para el *índice*, se realizó la implementación en el documento **PDF** (ver figura 5.11). El índice en XSL-FO se creó con base al tipo de fragmento que necesitaba ser ubicado en éste. Por ejemplo, una *sección* se implementó con un tipo de letra más grande y una menor indentación que una *subsección*, y una *subsección* a su vez, se implementó con una indentación menor que un fragmento de tipo *subDos*. Para poder tener cada una de las referencias a la página en la que se encontraba el fragmento deseado, en cada uno de los *objetos de flujo* de los tipos de fragmentos (Responsabilidades, Alcance, Objetivo, etc.), se agregó una referencia dentro de ellos (línea 12), dicha referencia es usada como identificador para generar la ubicación exacta donde se encuentra esa sección de la **Norma**.

Por ejemplo, para el caso del *objeto de flujo* del fragmento de tipo *Alcance* se tiene:

```
00 <xsl:template match="fragmento[@tipo='alcance']">
01
02
03   <fo:block font-size="14pt"
04     font-family="sans-serif"
05     line-height="18pt"
06     space-before.optimum="10pt"
07     space-after.optimum="5pt"
08     text-align="left"
09     white-space-collapse="false"
10     wrap-option="no-wrap"
11
12     id="{generate-id()}">
13
14     <xsl:apply-templates select="@titulo"/>
15   </fo:block>
16
17   <fo:block font-size="10pt"
18     font-family="sans-serif"
19     line-height="15pt"
20     space-after.optimum="10pt"
```

```
21      text-align="left">
22    <xsl:apply-templates/>
23    </fo:block>
24
25 </xsl:template>
```

Aquí se puede observar la generación del identificador con el cual se hace la referencia en el índice. Se ve claramente también, en este código, la estructura del tipo de fragmento *alcance*. Por una parte, se tiene el código para asignarle un título al fragmento (líneas 03 a 15) y por otra lado la parte donde se pide la visualización del contenido del fragmento (líneas 17 a 23).

El código que permite la generación del índice, para el formato **PDF**, se puede observar en el apéndice A.

por razones de espacio, sólo se muestra la implementación del índice, en dicho apéndice, para algunos de los fragmentos de las secciones de la **Norma**.

Tipos de fragmento

Para la implementación de cada uno de los tipos de fragmentos, se agregó, como encabezado, el texto que contiene el árbol del documento de **ELXI**. Dicho texto sirve como título a la sección. En la versión 1.0 de **ELXI**, este encabezado era directamente insertado en la hoja de estilo XSL [17]. En esta nueva implementación, se asignó el texto del árbol, ya que esto dá solución al problema de la numeración de las secciones de la **Norma**. El usuario enumera directamente las secciones del documento, dando con esto una libertad más grande para asignar orden a los fragmentos. Con esto se logra también una implementación de un documento **Norma** con secciones más generales, tanto en orden como en contenido. Esta forma de diseñar una **Norma** más general y abierta ayudó a la creación e implementación de un documento de tipo **General**.

Para cada sección de la **Norma** se observa la implementación de la estructura de cada fragmento. Cuando se llama al *objeto de flujo*, de alguna sección de la **Norma**, se pide la visualización del título del fragmento y luego se pide la aplicación del template **DocXMLEmbebido**. Este elemento está

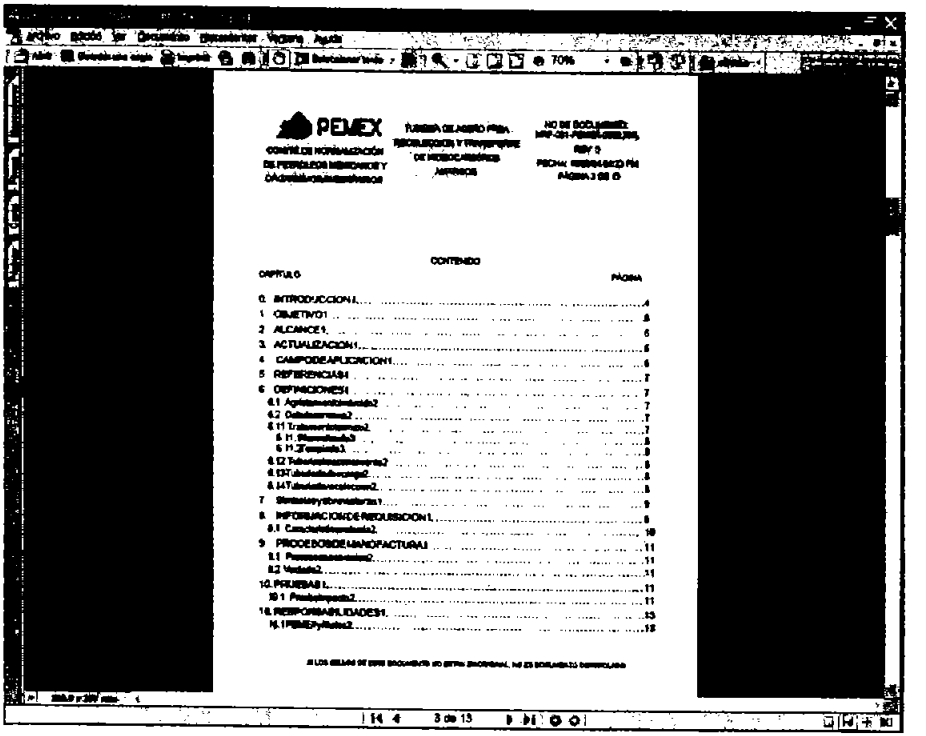


Figura 5.11: Índice de una **Norma**

en todos los fragmentos y es definido en el momento de crear el documento XML. La *etiqueta de marca* **DocXMLEmbebido** contiene las etiqueta de los elementos de edición recién creados (listas, tablas, pies de página, etcetera).

De esta manera, cada fragmento puede hacer uso de las *etiquetas de marca* creadas para los elementos de edición. El siguiente es un ejemplo de un fragmento que forma parte de un documento XML de tipo **Norma** en **ELXI**.

```
<fragmento id_fragment="8" tipo="simbolos y abreviaturas"
titulo="6. Simbolos y abreviaturas"><DocXMLEmbebido><texto> En el
contenido de esta norma de referencia se mencionan diversas
```

```
siglas, símbolos y vocablos técnicos, que se describen a
continuación:</texto>

<tabla cols="150pt 150pt 150pt" border="1"> <renglon>
<celda><texto>H2S</texto></celda>
        <celda><texto>ACIDO SULFHIDRICO.</texto></celda>
        <celda><texto></texto></celda>
</renglon>

<renglon> <celda><texto>DWTT</texto></celda>
        <celda>
        <texto>PRUEBA DE IMPACTO POR CAIDA DE PESO.</texto>
        </celda>
        <celda><texto>(DROP WEIGHT TEAR TESTING).</texto></celda>
</renglon>

<renglon> <celda><texto>HV</texto></celda>
        <celda><texto>DUREZA VICKERS.</texto> </celda>
        <celda><texto>(HARDNESS VICKERS).</texto></celda>
</renglon>

<renglon> <celda><texto>NMX</texto></celda>
        <celda><texto>Norma Mexicana</texto></celda>
        <celda><texto></texto> </celda>
</renglon>

<renglon> <celda><texto>API</texto></celda>
        <celda>
        <texto>INSTITUTO AMERICANO DEL PETROLEO.</texto>
        </celda>
        <celda>
        <texto>(AMERICAN PETROLEUM INSTITUTE).</texto>
        </celda>
</renglon>

<renglon> <celda><texto>RP</texto></celda>
        <celda><texto>PRÁCTICA RECOMENDADA.</texto></celda>
        <celda><texto>(RECOMMENDED PRACTICE).</texto></celda>
```

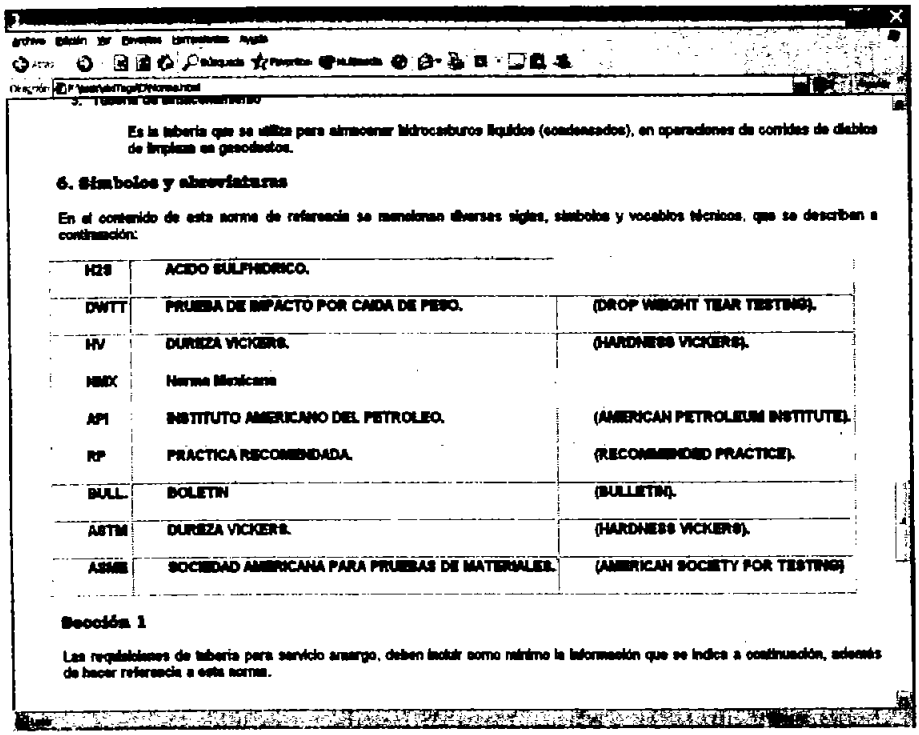
```
</renglon>

<renglon> <celda><texto>BULL.</texto></celda>
          <celda><texto>BOLETIN </texto></celda>
          <celda><texto>(BULLETIN).</texto></celda>
</renglon> <renglon> <celda><texto>ASTM</texto></celda>
          <celda><texto>DUREZA VICKERS.</texto></celda>
          <celda><texto>(HARDNESS VICKERS).</texto></celda>
</renglon> <renglon> <celda><texto>ASME</texto></celda>
          <celda>
          <texto>SOCIEDAD AMERICANA PARA PRUEBAS DE MATERIALES.
          </texto>
          </celda>
          <celda><texto>(AMERICAN SOCIETY FOR TESTING)
          </texto></celda>
</renglon>

</tabla></DocXMLEmbebido></fragmento>
```

La figura 5.12 muestra, en formato **HTML**, una tabla dentro de un fragmento tipo *Símbolos y abreviaturas* de una **Norma** visualizada por un navegador.

5.1.3. Diseño e implementación de interfaces de usuario

Hasta ahora el editor es capaz de tomar el código de marcas escrito directamente en cada uno de los fragmentos. Como se mencionó, este código, **ELXI** lo toma como texto durante la distribución del documento. Cuando se solicita la visualización en algún cliente, se realiza la generación del **Documento Alterno**, el cual maneja el contenido de los fragmentos ya como código de marcas dentro del la *etiqueta DocXMLEmbebido*.

Esta implementación fue parte de la estrategia para lograr la funcionalidad del sistema. Sin embargo, esta implementación no es viable en su manejo para un usuario que no conoce el lenguaje de marcas definido y tampoco tiene interés en aprenderlo. Así, para implementar la capa de software, que resolviera este requerimiento se diseñó un analizador, el cual permite cada



Es la tubería que se utiliza para almacenar hidrocarburos líquidos (condensados), en operaciones de corridas de diablos de limpieza en gasoductos.

6. Símbolos y abreviaturas

En el contenido de esta norma de referencia se mencionan diversas siglas, símbolos y vocablos técnicos, que se describen a continuación:

H2S	ACIDO SULPHIDRICO.	
DWTT	PRUEBA DE IMPACTO POR CAIDA DE PESO.	(DROP WEIGHT TEAR TESTING).
HV	DUREZA VICKERS.	(HARDNESS VICKERS).
NMX	Norma Mexicana	
API	INSTITUTO AMERICANO DEL PETROLEO.	(AMERICAN PETROLEUM INSTITUTE).
RP	PRACTICA RECOMENDADA.	(RECOMMENDED PRACTICE).
BULL.	BOLETIN	(BULLETIN).
ASTM	DUREZA VICKERS.	(HARDNESS VICKERS).
ASME	SOCIEDAD AMERICANA PARA PRUEBAS DE MATERIALES.	(AMERICAN SOCIETY FOR TESTING)

Sección 1

Las requisiciones de tubería para servicio amargo, deben incluir como mínimo la información que se indica a continuación, además de hacer referencia a esta norma.

Figura 5.12: Tabla en una Norma HTML

vez que se realiza la edición del documento, analizar la entrada del usuario, y transformarla al código de marcas correspondiente. De igual forma, cuando se pide la edición de algún documento se tiene la capacidad de leer el código de marcas contenido en cada uno de los fragmentos y mostrarlo de forma clara al usuario —sin etiquetas— como cualquier otro editor de texto.

Para el caso de los elementos de texto, el problema consistió en analizar el contenido de las etiquetas *texto* y simplemente visualizar su contenido en un área de texto limpia. Este texto limpio de etiquetas en el fragmento muestra el contenido del elemento de edición *texto* de forma que el usuario puede editar su contenido sin tener relación con etiqueta alguna.

Para los demás elementos de edición como tablas, imágenes, listas, etc., se implementó de nuevo un algoritmo de análisis del código de marcas. En el momento en que se encuentra una *etiqueta de marca* que sea diferente de texto, se visualiza para el usuario un **botón** en el fragmento, el cual representa el contenido de este elemento de edición. De esta forma, cada elemento de edición queda representado y ubicado en el contenido del fragmento y del documento por botones (ver figura 5.13).

Si se desea editar el elemento, al presionar sobre el botón, se abre la interfaz específica al elemento que se quiere editar. En esa misma interfaz, al aceptar los cambios hechos en ese elemento de edición, se modifica el código de marcas relacionado, se cierra la interfaz y se visualiza de nuevo el botón que representa al elemento en el fragmento.

De igual forma, el usuario puede agregar nuevos elementos o modificar el texto existente de un fragmento. Cualquier cambio implicará una modificación al código de marcas, tanto para elementos *texto* como para cada uno de los otros elementos de edición.

El trabajo de implementación de este analizador se realizó con las bibliotecas DOM para Java, las cuales permiten el manejo de métodos de análisis en documentos XML. Para el presente trabajo, se realizó la primera etapa de este analizador y la implementación de las interfaces para cada uno de los elementos de edición de forma separada, las etapas restantes y la integración de las interfaces de cada uno de los elementos en **ELXI** se llevó a cabo por los integrantes del grupo de desarrollo de **ELXI** en el IMP.

Interfaces de imágenes

Para una imagen, la interfaz presenta una lista de los archivos posibles a agregar, además de una pequeña ventana de visualización de la imagen a ser agregada. Tres acciones son posibles para esta interfaz, agregar imagen, aceptar y cancelar, el evento agregar imagen permite la previsualización de la imagen y el evento aceptar agrega el nuevo botón al editor y genera el código XML del elemento agregándolo al documento en edición. Cuando se abre la interfaz para una imagen ya existente en el documento, sólo los eventos de agregación y cancelación son aceptados, hasta que se haya cambiado

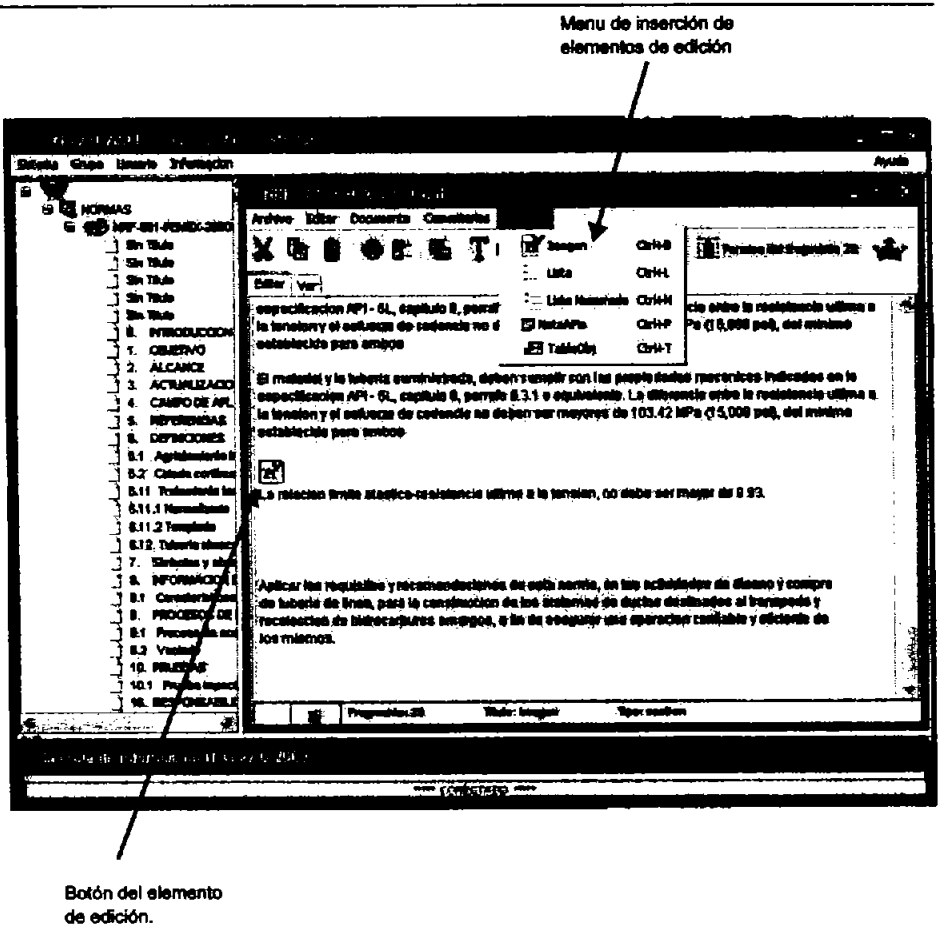


Figura 5.13: Inserción de elementos de edición en el documento

la imagen anteriormente referenciada se habilitará el evento de aceptación. En la figura 5.14, se observa el icono de una imagen en un fragmento y en la figura 5.15 se muestra la interfaz del elemento imagen.

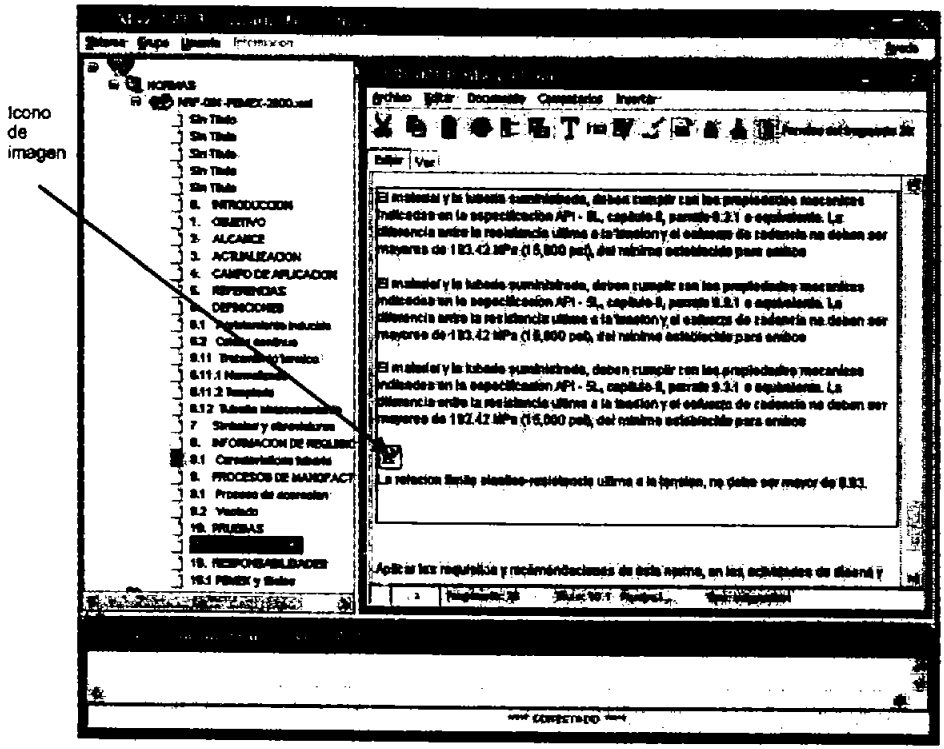


Figura 5.14: Icono de imágenes dentro del fragmento en edición

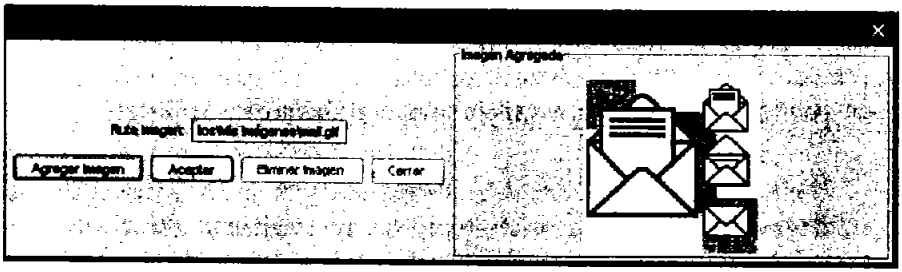


Figura 5.15: Interfaz para la agregación de Imágenes

Interfaz de listas

Para la interfaz de lista se implementaron dos vistas: la primera solicita el número de elementos de la lista y contiene los eventos de aceptación y cancelación. La segunda, como consecuencia del evento de aceptación de la primera, presenta un panel dividido en tres partes donde se muestran el número del elemento, las áreas de texto de contenido de cada uno de los elementos y los eventos de control de agregación, eliminación y modificación de estos.

La parte central del panel muestra un campo y una área de texto. En el primero se agrega el contenido del elemento y en el segundo una posible descripción de éste. Este formato es parecido a una *lista con descripción* de LaTeX. Además, por cada elemento, se implementaron tres botones de opción excluyentes, cada uno con la marca, eliminar, agregar arriba, agregar abajo. En la parte derecha, se encuentran varios botones: eliminar elemento, agregar elemento, aceptar, cancelar, entre otros, los cuales se relacionan con los botones de opción excluyentes para efectuar los eventos específicos. Por ejemplo, si se desea eliminar un elemento se selecciona la opción eliminar y se pulsa el botón *eliminar elemento* de la parte derecha del panel (ver figura 5.16).

De igual forma que la interfaz de imagen, el botón cancelar regresará a la interfaz anterior y el botón aceptar generará el código de marcas y un nuevo botón que representa a la lista. Este botón será visible al usuario desde el editor. Cabe señalar que la interfaz muestra, de forma dinámica, el contenido y modificaciones de los elementos modificados y generados por los botones de eventos, cada cambio se refleja inmediatamente en la interfaz.

Cada uno de los elementos y descripciones de la lista tienen las mismas características que las áreas de texto de un fragmento (en el cual aparece texto y botones para el usuario), con lo cual dentro de cada elemento o descripción puede existir texto o botones de la misma manera en la que aparecen en los fragmentos. Dentro de cada elemento de una lista puede haber la posibilidad de existir botones de lista a su vez. Esto permite anidar elementos deseados y generar el código de marcas con elementos anidados que, como se explicó anteriormente, es soportado por las plantillas de transformación implementadas en XSL.

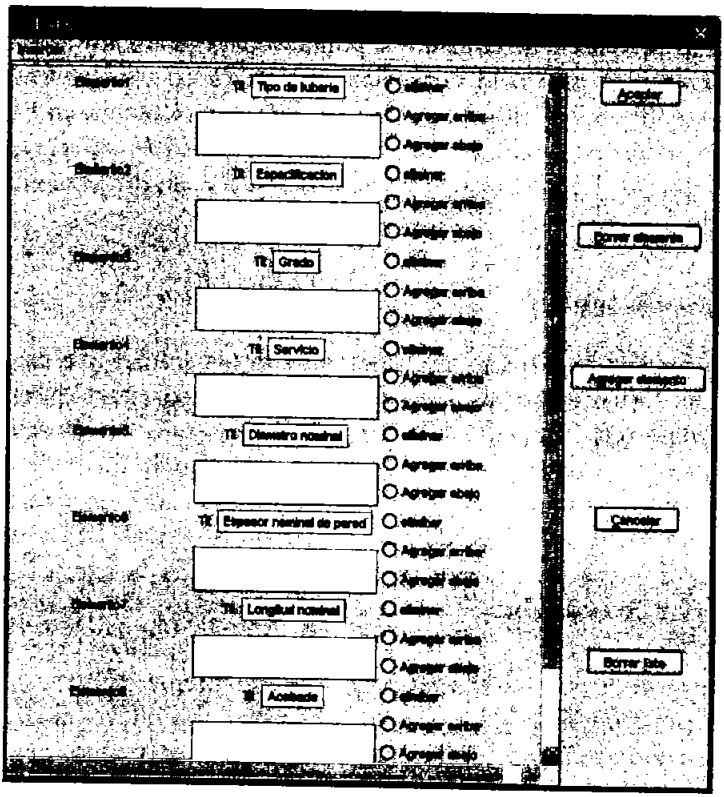


Figura 5.16: Interfaz para el manejo de listas

Las interfaces de listas, en la parte superior, presentan un menú desplegable el cual permite abrir nuevas interfaces del tipo de elemento que se quiera anidar. Por ejemplo, para la lista aparece la opción *insertar lista*, la cual visualiza de nuevo la primera interfaz de la lista, solicitando el número de elementos de la lista anidada, luego muestra en una nueva interfaz los nuevos elementos de la lista anidada. Al pulsar el evento aceptar, se genera en la primera interfaz, en el elemento de descripción seleccionado, un botón con las mismas características que el que aparece en un fragmento para referenciar una lista.

Ahora, si en esta interfaz se oprime el botón del evento aceptar, se genera el código de marcas de la lista con la nueva lista anidada y se dibuja el botón de referencia a esta lista con elementos anidados para el fragmento. Si se desea reeditar esta lista, se pulsa el botón de referencia en el fragmento y se busca el elemento de descripción. Este elemento de descripción contendrá otro botón de lista, el cual permitirá la edición de los elementos de la lista anidada.

Interfaz de tablas.

La interfaz de tabla se implementó de forma similar a la lista, pero ahora para una forma matricial. Una primera interfaz solicita el número de celdas y renglones que contendrá la tabla, además de contener los eventos cancelar y aceptar. La segunda interfaz muestra una matriz de celdas generada de acuerdo con los datos dados en la primera interfaz y una serie de botones para los eventos relacionados con el manejo de celdas. En la figura 5.17, se pueden observar los botones de tablas y listas. En la figura 5.18 se observa la interfaz de una tabla.

Los eventos pueden agregar renglones y columnas o borrarlas, borrar todo el contenido de las celdas, aceptar y cancelar. De nuevo, cancelar retrocede a la primera interfaz y aceptar genera el código de marcas para el elemento tabla.

5.1.4. Diseño e implementación del documento General

Una vez que se tuvo la **Norma** terminada, se implementó la estructura que pudiera resolver las *etiquetas de marca* necesarias para implementar un documento más general en su formato que una **Norma**. Desde el comienzo del análisis para las nuevas características del editor **ELXI**, se definió el poder resolver los requerimientos que este trabajo de tesis planteaba: **ELXI** debía de ser capaz de editar el mayor número de tipos de documentos posibles, por ejemplo, una carta, un curriculum, un documento técnico o científico, etc.

Al ir implementado cada una de las partes de una **Norma**, se definió una estructura muy general, con la idea de lograr el objetivo anteriormente planteado. Se discutió previamente en la sección 4.5 la propuesta de diseño

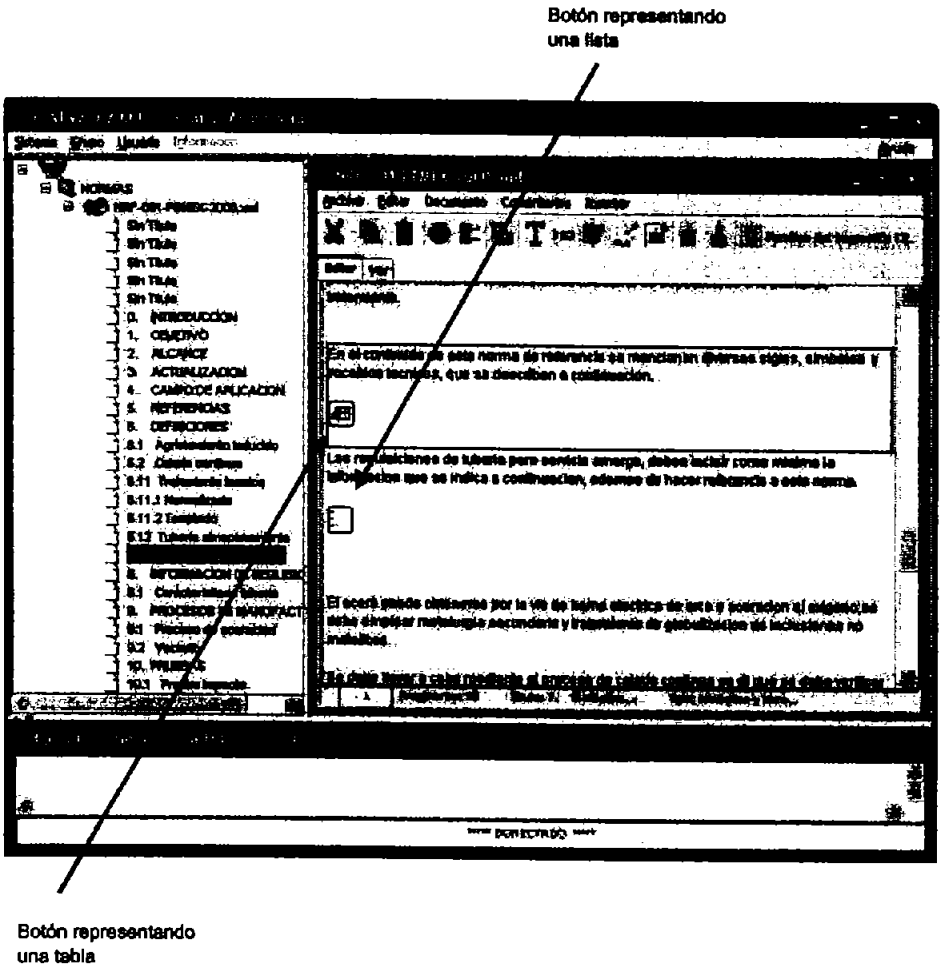


Figura 5.17: Icono de tablas y listas dentro de fragmentos en edición

de el documento **General**, la cual permite la existencia de tipos más abiertos para la edición de una mayor cantidad de tipos de documentos.

Textos

A excepción del elemento de edición *texto* de una **Norma**, cada uno de los elementos de edición para un fragmento fue también usado en el documento

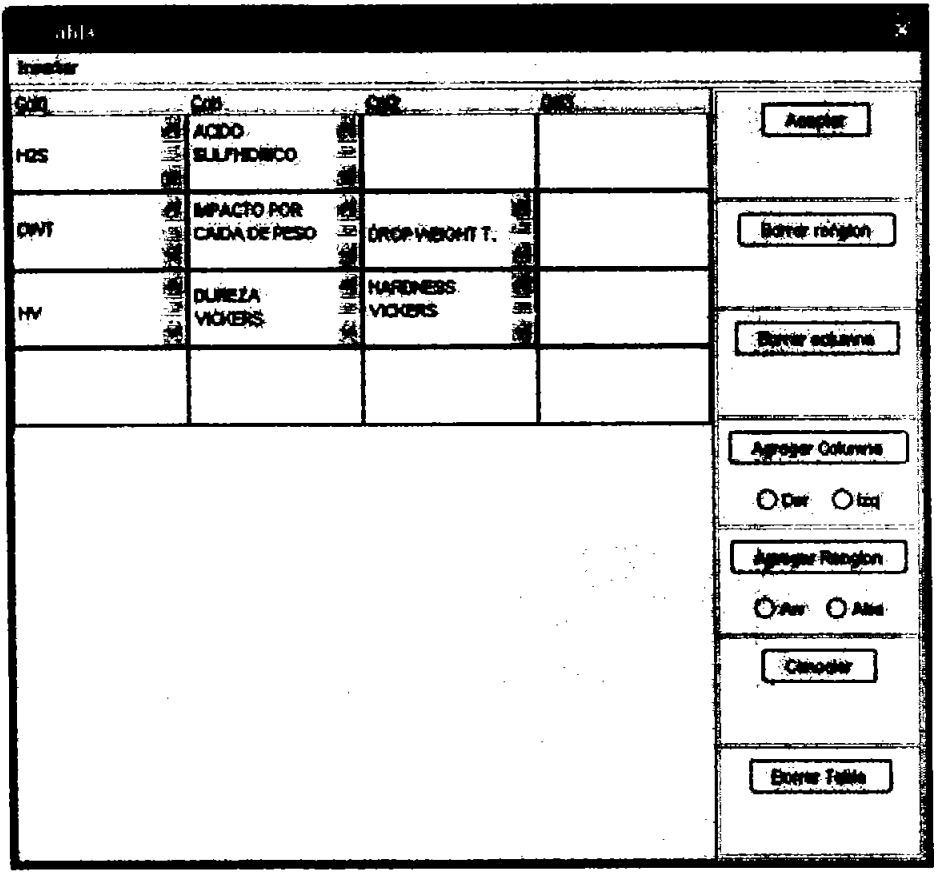


Figura 5.18: Interfaz para una tabla

General. Además, cada fragmento en el tipo de documento **General** tiene la posibilidad de contener cualquier tipo de elemento de edición.

El elemento *texto* se implementó con una estructura de control, la cual permite saber, por medio de XPATH [25], si alguno de los padres del nodo analizado pertenece a algún tipo de fragmento específico (Titulo1, Titulo2, parrafo-derecha, etc.) (líneas 02, 11, 18, y 24). Si alguno pertenece, el contenido de la etiqueta *texto* es mostrado en el formato determinado para ese

tipo de fragmento (líneas 05, 12, 19, y 25). La implementación es la siguiente:

```
00 <xsl:template match="texto">
01
02 <xsl:when test="ancestor-or-self::fragmento[@tipo='Titulo1'] |
03                    ancestor-or-self::fragmento[@tipo='Titulo2'] |
04                    ancestor-or-self::fragmento[@tipo='Titulo3'] ">
05     <p align="center">
06         <xsl:apply-templates/>
07     </p>
08 </xsl:when>
09
10 <xsl:when
11 test="ancestor-or-self::fragmento[@tipo='Parrafo-derecha']">
12     <p align="right">
13         <xsl:apply-templates/>
14     </p>
15 </xsl:when>
16
17 <xsl:when
18 test="ancestor-or-self::fragmento[@tipo='Parrafo-centrado']">
19     <p align="center">
20         <xsl:apply-templates/>
21     </p>
22 </xsl:when>
23
24 <xsl:otherwise>
25     <p align="justify">
26         <xsl:apply-templates/>
27     </p>
28 </xsl:otherwise>
29 </xsl:choose>
30 </xsl:template>
```

Lo anterior permite la alineación del contenido de texto de un fragmento.

De forma similar, para el tipo de letra del texto de un fragmento se tiene:

```
00 <xsl:template match="fragmento[@tipo='Italica']">
01     <I><xsl:apply-templates/></I>
02 </xsl:template>
03
04 <xsl:template match="fragmento[@tipo='Bold']">
05     <B><xsl:apply-templates/></B>
06 </xsl:template>
07
08 <xsl:template match="fragmento[@tipo='Bold-Italica']">
09     <I><B><xsl:apply-templates/></B></I>
10 </xsl:template>
```

Lo cual define que, para ese tipo de fragmento, el contenido será mostrado con un tipo de letra específico (a menos que el elemento de edición, llamado en un nivel más arriba, indique otra cosa). En la figura 5.19, se muestra el menú de tipos (con algunos de ellos) del documento **General**.

Los *tipos* de fragmento de un documento **General** fueron definidos previamente en la sección 4.5. Cada uno de estos tipos tiene características específicas. Por ejemplo, para los *templates Titulo1*, *Titulo2* y *Titulo3* (líneas 00, 07, y 14) se codifican diferentes tamaños de letras centradas (líneas 01, 08, y 15).

```
00 <xsl:template match="fragmento[@tipo='Titulo1']">
01     <h1 >
02     <center><xsl:apply-templates/></center>
03     </h1>
04     <br/>
05 </xsl:template>
06
07 <xsl:template match="fragmento[@tipo='Titulo2']">
08     <h2 align="center">
```

```
09      <xsl:apply-templates/>
10      </h2>
11      <br/>
12 </xsl:template>
13
14 <xsl:template match="fragmento[@tipo='Titulo3']">
15      <h3 align="center">
16      <xsl:apply-templates/>
17      </h3>
18      <br/>
19 </xsl:template>
```

Secciones y subsecciones

La implementación de secciones y subsecciones de una **Norma** fue usada sin mucha diferencia para el documento **General**, debido a que una sección en un documento **Norma** representa un tipo de fragmento en el cual se puede agregar cualquier elemento de edición.

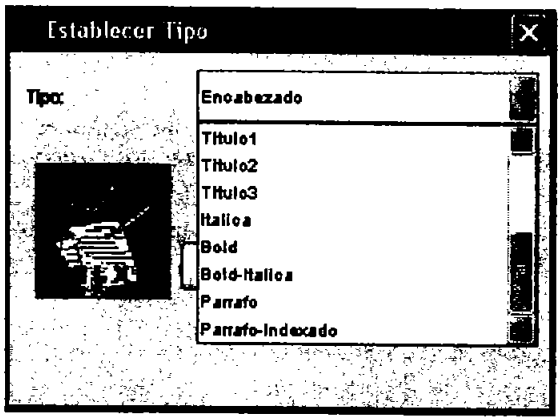


Figura 5.19: Menú de tipos de fragmento en el documento **General**

La forma de llamar a cada uno de los *objetos de flujo* de las secciones (sección, párrafo, subsección, etc.) es semejante a la forma utilizada en la **Norma**. Se construye la llamada a través de una función lógica, constituida con cada uno de los tipos creados (líneas de 01 a 09) y operados por conjunciones. Esto permite la ubicación de cada uno de los fragmentos en la posición en la que es llamado desde la estructura del **Documento Alterno** XML.

```
00 <xsl:apply-templates
01 select="fragmento[@tipo='libre']|fragmento[@tipo='section']|
02 fragmento[@tipo='subsection']|fragmento[@tipo='subDos']|
03 fragmento[@tipo='subTres']|fragmento[@tipo='Titulo1']|
04 fragmento[@tipo='Titulo2']|fragmento[@tipo='Titulo3']|
05 fragmento[@tipo='Italica']|fragmento[@tipo='Bold']|
06 fragmento[@tipo='Bold-Italica']|fragmento[@tipo='Parrafo']|
07 fragmento[@tipo='Parrafo-Indexado']|
08 fragmento[@tipo='Parrafo-centrado']|
09 fragmento[@tipo='Parrafo-derecha']"  />
```

Encabezados y pies de página

La forma en que los tipos *Nota a Pie de página* y *pie de página* fueron implementados en una **Norma**, fue también implementada en los objetos de flujo *templates* del documento **General**. Es decir, cada fragmento puede generar referencias a pies de página y existe un tipo de fragmento *pie de pagina*, el cual agrega su contenido al final de cada hoja del documento visualizado.

Para el caso específico del encabezado en el documento **General**, se implementó un *objeto de flujo* semejante al de tipo *pie de página*. A diferencia de una **Norma**, el encabezado no está predefinido por la misma forma del documento y no se implementa directamente en la hoja de estilo XSL. Por esta razón, el tipo encabezado puede tener en su contenido cualquier elemento de edición, lo que lo hace más útil para otros tipos de documentos.

Implementación de una Norma en el formato del documento General

Para comprobar la utilidad de edición del documento **General**, se consideró la capacidad de manejo de un documento **Norma** en un documento de tipo **General**. Se logró observar que varias especificaciones de una **Norma** son mejor logradas en el tipo de documento hecho para este fin. Por ejemplo, los encabezados tienen un mejor formato en un documento **Norma**, por contar con un fragmento específico para este tipo de documento.

Sin embargo, el documento **General** puede editar todas las partes de la **Norma** y aun presentar más características que no fueron necesarias para una **Norma**. En la figura 5.20 en la página 156, se puede observar este documento de normatividad elaborado con el documento de tipo **General** de **ELXI**.

5.2. Síntesis y discusión

Para terminar este capítulo mostramos las DTDs de los documentos de **ELXI**. Se muestra primeramente la DTD de la versión 1.0 de **ELXI** y posteriormente se muestra la DTD que **ELXI** usa, luego de este trabajo, para definir la estructura de un documento de tipo **Norma** o **General**.

DTD de la versión 1.0 de **ELXI**:

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

<!ELEMENT fragmento (#PCDATA)*> <!ATTLIST fragmento
          titulo CDATA #REQUIRED
          id_fragment CDATA #REQUIRED
          tipo CDATA "">
```

La DTD de la estructura de documentos en **ELXI** obtenida como resultado del presente trabajo es:

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

<!ELEMENT documento(fragmento+)> <!ATTLIST documento
          id_file CDATA #REQUIRED
          tipo_f CDATA "">

<!ELEMENT fragmento (DocXMLEmbebido+)> <!ATTLIST fragmento
          titulo CDATA #REQUIRED
          id_fragment CDATA #REQUIRED
          tipo CDATA #REQUIRED>

<!ELEMENT DocXMLEmbebido (#PCDATA|tabla|imagen|listaD |lista |
listaN |texto |textoP)*>


<!ELEMENT listaN (elementoN)+> <!ATTLIST lista
          titulo CDATA #IMPLIED
          id_lista CDATA #IMPLIED
          >

<!ELEMENT listaD (elementoD)+> <!ATTLIST lista
          titulo CDATA #IMPLIED
          id_lista CDATA #IMPLIED
          >

<!ELEMENT lista (elemento)+> <!ATTLIST lista
          titulo CDATA #IMPLIED
          id_lista CDATA #IMPLIED
          >

<!ELEMENT elemento (#PCDATA| lista |listaN| listaD | texto|
textoP)*>

<!ELEMENT elementoN (#PCDATA|lista | listaN| listaD| texto|
textoP)*>

<!ELEMENT elementoD (#PCDATA|lista| listaN | listaD| texto|
textoP)*>
```

```
<!ELEMENT tabla (renglon)*>

<!ELEMENT renglon (celda)*>

<!ELEMENT celda (#PCDATA|tabla|imagen|lista |listaN | listaD|
texto| textoP)*>

<!ELEMENT imagen (#PCDATA)>

<!ELEMENT texto (#PCDATA | notaAPie | B | S | U)*>
<!ATTLIST texto
          tam CDATA #IMPLIED
          alin CDATA #IMPLIED
          >

<!ELEMENT textoP (#PCDATA| notaAPie | B | S | U)*>
<!ATTLIST textoP
          tam CDATA #IMPLIED
          alin CDATA #IMPLIED
          >

<!ELEMENT notaAPie (#PCDATA)>

<!ELEMENT B (#PCDATA)>

<!ELEMENT S (#PCDATA)>

<!ELEMENT U (#PCDATA)>
```

Cabe hacer notar también que, al final de esta implementación, las clases y métodos en **ELXI** crecieron, en número y código, de forma importante. Como ya se mencionó, **ELXI** contaba en la versión 1.0 con cerca de 30 métodos remotos en el servidor RMI y 20 clases en el cliente **ELXI**. Después de la implementación de este trabajo, **ELXI** cuenta con 60 métodos en el servidor RMI y con 30 clases en el cliente **ELXI**.

En este capítulo se mostró el diseño y la implementación de las ideas propuestas en los capítulos anteriores. Este código permitió, de forma tangi-

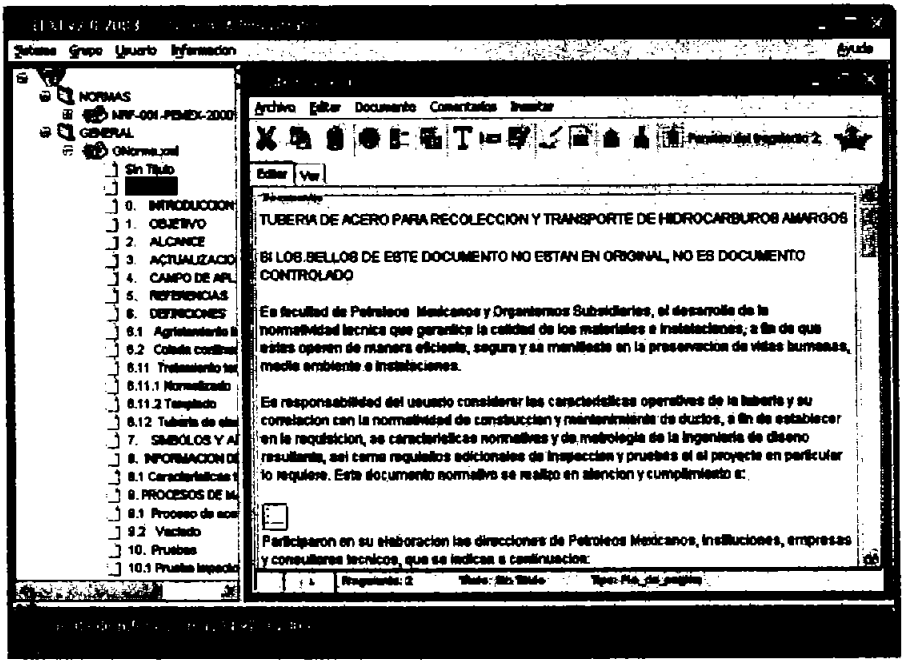


Figura 5.20: **Norma** implementada con un tipo de documento **General**

ble, al editor **ELXI** robustecerse, para ofrecer nuevas características en los documentos que en él se editan.

Se mostró la forma en que se implementó el analizador sintáctico necesario para la distribución de documentos; se mostró la codificación para la administración de nuevos elementos de edición a través de interfaces de usuario; se presento el código para la visualización de los diferentes formatos en **ELXI**; y por último, se codificaron los módulos necesarios para la implementación de un documento **General** en **ELXI**.

Capítulo 6

Evaluación por usuários reales

6.1. Evaluación del editor

El sistema fue utilizado por usuários del Instituto Mexicano del Petróleo para valorar sus alcances e impacto. Para este fin, clientes de **ELXI** fueron instalados en las máquinas de los distintos usuários, los cuales se encontraban en diferentes edificios del Instituto. El servidor **ELXI** fue instalado por separado, en la sala de servidores, en una máquina asignada para este fin (pentium III a 1130 MHZ con 2 GB de memoria RAM). Los clientes fueron instalados en máquinas con procesadores pentium III o pentium IV con memorias de 256 MB o 512MB, con Windows XP como sistema operativo.

La evaluación fue realizada por 14 usuários del Instituto. Se realizaron cada una de las instalaciones y capacitaciones necesarias para el uso del editor. Cada capacitación se enfocaba en mostrar el uso y capacidades de **ELXI**, alentando a los usuários a elaborar documentos usados de forma cotidiana en sus actividades laborables.

Luego de un periodo de uso, de un mes en promedio, se pidió a los usuários contestar un cuestionario de evaluación del sistema. Este cuestionario constó de diez preguntas organizadas según los módulos en los que se realizó este trabajo: distribución de fragmentos, visualización de documentos e interfaces de usuario.

Las preguntas se calificaron en una escala de valores según la siguiente relación.

1. Muy bien
2. Bien
3. Suficiente
4. Mal

Las instrucciones y preguntas planteadas se muestran en el apéndice B.

Los resultados, graficados y en forma de tabla, se presentan en la figura 6.1. En esta gráfica se observan los valores obtenidos para cada una de las preguntas planteadas[1].

De los datos observados, se tiene que las preguntas fueron calificadas en los rangos más altos. Lo que muestra una buena aceptación del sistema por parte de los usuários. Para los casos donde la valoración fue *suficiente*, se revisaron las opiniones otorgadas mediante las cuales se justificó la valoración. En casi todos los casos, esta calificación se dió a razón de que consideraban, como *suficientes*, las características del sistema para la edición de documentos y no como consecuencia del mal funcionamiento de alguna parte del editor. Como caso particular, la pregunta 2.2 muestra la única calificación más baja, para esta repuesta, al buscar en el cuestionario, no existe opinión o mejora y se presenta como un dato aislado dentro de la población. De lo anterior, se puede concluir que el sistema fue aceptado de buena forma por los usuários que lo utilizaron. Sin embargo, se vertieron algunas propuestas o mejoras de las cuales hablaremos y analizaremos a continuación.

Las respuestas a cambios y mejoras propuestas, por cada pregunta, dieron una valiosa información. Una de las propuestas, con relación a la pregunta 1.1, fue agregar a **ELXI** la funcionalidad de editar documentos con la sintaxis de LaTeX. Esta propuesta se debe a que un número considerable de los

[1] La suma de usuários por pregunta, en en algunos casos, no es igual al total de usuários. La razón de esto fue debido a que, por algunos grupos de trabajo, no fue necesario usar la funcionalidad en el documento que elaboraron.

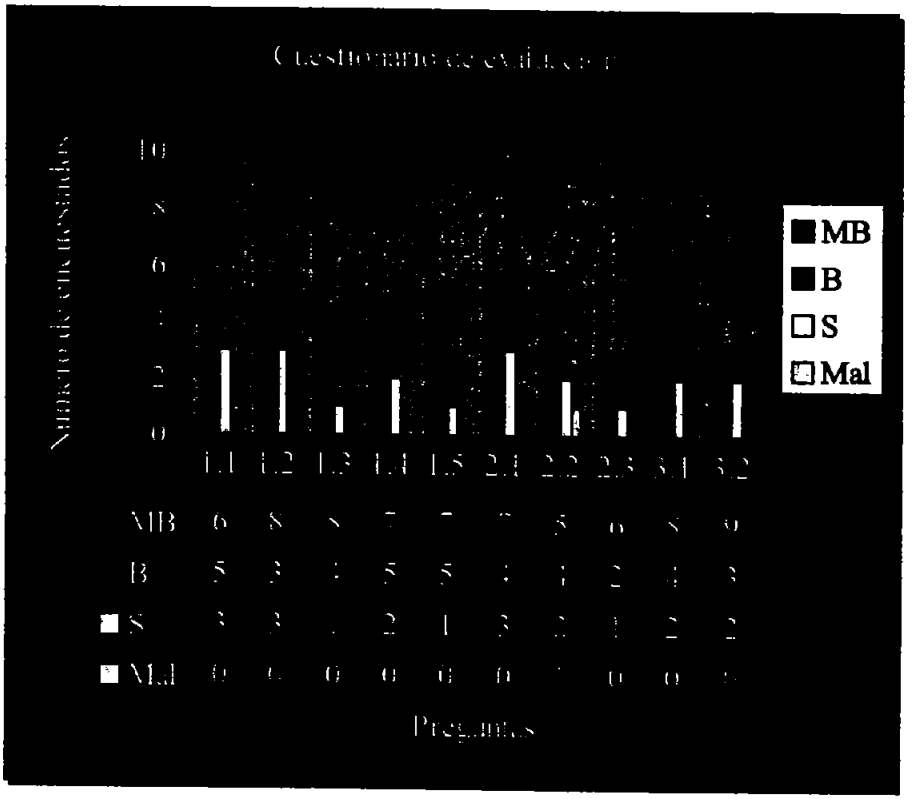


Figura 6.1: Resultados de la evaluación

usuários encuestados elaboran documentos científicos en este formato y sería un punto muy importante a abordar en el desarrollo futuro de **ELXI**.

Por otra parte, se observa en las respuestas, relacionadas con las mejoras, que muchas de ellas están dirigidas a las interfaces que realizan el manejo de listas, tablas e imágenes. Se menciona que se podrían mejorar agregando funcionalidades, esto es, podría, cada interfaz, ser capaz de realizar un mejor manejo de los elementos. Debido a limites de tiempos en este proyecto, estas interfaces se implementaron de forma básica y funcional. En trabajos futuros

se piensa hacer mucho más rico este *front-end*[2] (sea por ejemplo el caso de **Lyx** [29], un proyecto para el desarrollo de interfaces que manejan de forma transparente, para el usuario, código de LaTeX), debido a que en varias respuestas se comparan directamente todas las capacidades de las interfaces de Word contra las aquí implementadas. Cabe señalar que estas funcionalidades propuestas son deseables, pero las interfaces implementadas permiten crear y manejar los elementos de edición de forma básica.

Respecto a los tipos de fragmentos, se otorgan valoraciones de *bien* a *muy bien* pero se menciona que sería conveniente agregar otros (color de texto, estilos de letras, texto subrayado, texto tachado, etc.) debido a la comparación directa de *estilos*[3] del editor Word de Microsoft. Todas estas características pueden ser implementadas, en trabajos futuros, sin demasiados problemas, por la razón de que, el diseño de la DTD, para los documentos de **ELXI**, dá la capacidad de incrementar los tipos de fragmento del documento —ver sección 6.2—. Por causas de tiempo se definió un número finito de estos, pero se puede hacer crecer permitiendo una edición más rica de los documentos creados en **ELXI**.

6.2. Agregación de nuevos elementos y características en ELXI

ELXI en su diseño permite la agregación de nuevas características. Esta última sección expone la implementación de posibles nuevas capacidades en **ELXI**.

Para hacer más rica la visualización del texto en el documento, se podrían implementar, además de los tipos de fragmentos para el estilo del texto, otros *objetos de flujo* de estilo que pudieran ser contenidos dentro de las etiquetas *texto*. Estos *objetos de flujo* permitirían que partes del contenido de un elemento de texto pudieran ser visualizadas como *negritas*, *remarcadas*, *itálicas*, *hypervinculo*, etc. Para variar el tamaño de letra contenida en una etiqueta

[2]Capa de software que permite interactuar al usuario con capas intermedias del dominio del problema del sistema.

[3]Los estilos en Word permiten asignar tamaños y tipos de letra a párrafos o secciones de un documento escritos en este editor. Ej. Titulo1, texto sin formato, Normal, etc.

de *texto*, se le asignaría un atributo que definiera el tamaño requerido. A continuación se muestra, para más claridad, el código y la sección del documento XML que llamaría estos nuevos elementos.

```
00 <?xml version="1.0" encoding="UTF-8"?>
01
02 <?xml-stylesheet type="text/xsl" href="General.xsl"?>
03
04 <!DOCTYPE documento SYSTEM "documento.dtd">
05
06 <documento id_file="46" tipo_f="General">
07
08 <fragmento id_fragment="1" tipo="libre" titulo="">
09 <DocXMLEmbebido> <texto alin="center" tam="16"> Designing 10
10 interactive interfaces: theretical consideration of the complexity
11 11 of standars and guidlines, and the diference between envolving
12 and 12 formalised systems</texto> <texto align="center"
13 tam="12">Irma  Alm<notaAPie>Tel: +46-31-772-... fax:
14 +46-31...</notaAPie>  </texto> <texto tam="8"> <S>Division of
15 Human Factor Engineering,  Department of Product and Produc...</S>
16 </texto> <texto><B>Recived  20 February 2002; accepted 3 June
17 2002</B> </texto>  </DocXMLEmbebido> </fragmento>
18
19 <fragmento id_fragment="2" tipo="libre" titulo="Abstract">
20 <DocXMLEmbebido> <texto> Rapid growth of Internet resources has
21 stimulated much interest in their organization. The cataloging
22 community has developed new guidelines for cataloging these
23 materials, such as those presented in the manual edited by Olson
24 (1997), as well as those presented in ISBD (ER): The International
25 Standard Bibliographic Description for Electronic Resources
26 (Sandberg-Fox and Byrum, 1998). In addition, information
27 organizers have experimented with metadata standards such as
28 Dublin Core (DC), text encoding initiative (TEI), and encoded
29 archival description (EAD) to describe and encode information
30 objects for resource discovery (Hudgins et al., 1999). </texto>
31 <texto tam="8"> <S>Keywords:</S>Interactive interfaces; Envolving
32 systems...</texto> </DocXMLEmbebido> </fragmento>
33
34 continua...
```

Los tamaños de letra se pueden definir, como se mencionó, en forma de atributos para el *texto* (de forma análoga a la sintaxis de **HTML**) —líneas 09 y 31—. Para agregar más estilos tipo Word, se definirían *etiquetas* similares a las que permiten darle estilo al texto (negritas <B>, itálica <S>, etc.) —líneas 14, 15 y 16—.

Por ejemplo, para un estilo del tipo *hypervinculo*, se definiría la nueva *etiqueta* <hyper>y se implementará el código XSL y XSL-FO necesario para lograr su correcta visualización y funcionalidad. Un posible documento generado de la forma anterior se muestra en las figuras 6.3 y 6.2.

Así, **ELXI** presenta un entorno de desarrollo colaborativo funcional, pero esta abierto en su diseño a nuevas ideas que en un futuro podrían ser implementadas.

6.3. Síntesis y discusión

Como se observa, se obtuvieron ideas que permitirán mejorar muchas funcionalidades de nuestro editor. Todas éstas ideas planteadas muestran un interés tangible en **ELXI**, lo cual lo hace un producto interesante y con potencial de uso.

La evaluación arrojó datos medibles donde se muestra la aceptación del editor por parte de los usuários. Esta actividad permitió evaluar los resultados obtenidos en este trabajo. Se observa que estos resultados fueron alcanzados de forma satisfactoria.

Se mostró también, la posibilidad de agregar nuevos elementos y características en **ELXI**. **ELXI** por su diseño permite añadir nuevas características y mejoras, siendo así bastante flexible a nuevas implementaciones.

En el apéndice B, se anexan los cuestionarios aplicados, como referencia y fundamento a lo dicho en esta sección.

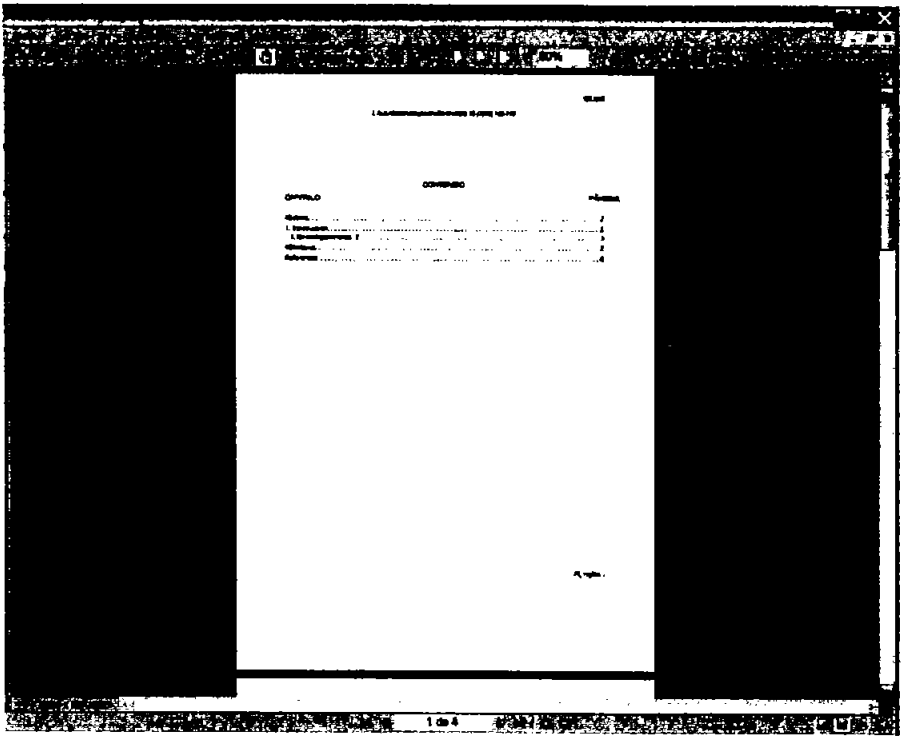


Figura 6.2: Índice del documento propuesto

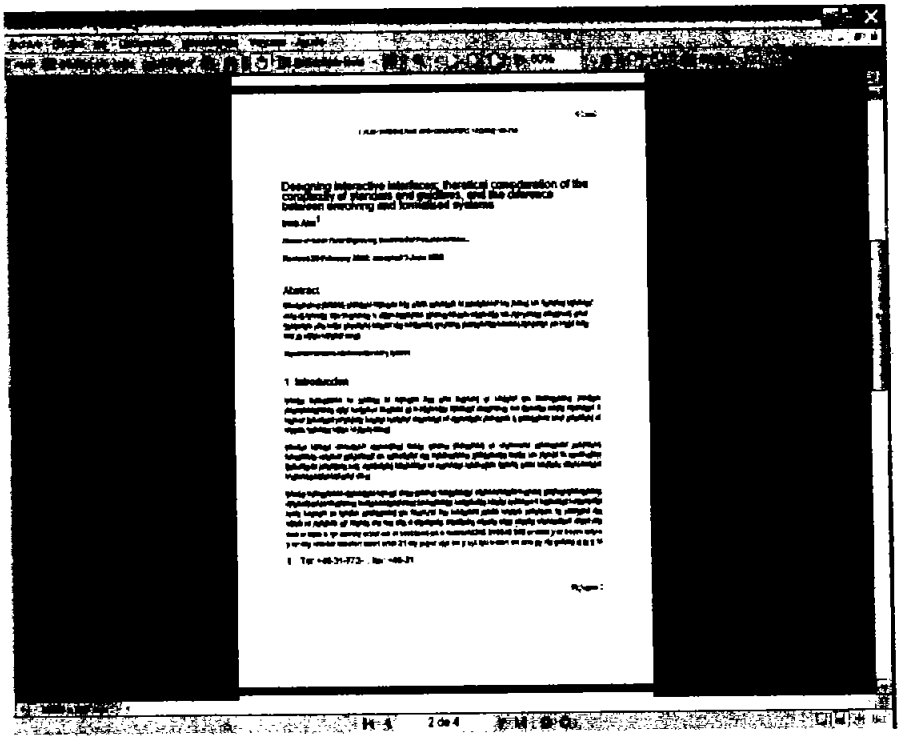


Figura 6.3: Visualización del documento propuesto PDF

Capítulo 7

Conclusiones y perspectivas

7.1. Resumen

En este trabajo se abordó la problemática de desarrollo y evaluación de un conjunto de mecanismos y algoritmos para la edición y visualización de documentos técnicos y científicos.

Se tomó como punto de partida el editor colaborativo **ELXI** en su versión 1.0. Este editor presentaba varias características susceptibles de ser mejoradas. **ELXI**, de manera general, divide un documento en fragmentos, de tal forma que, el documento puede ser editado por diferentes autores. De esta manera los autores pueden editar el mismo documento de forma síncrona o asíncrona, guiados por un conjunto de roles de edición, accediendo a cada uno de estos fragmentos en lectura y escritura.

En dicha versión, cada fragmento sólo aceptaba un bloque de texto sin ningún tipo de separación, es decir, no existía la posibilidad de agregar más de un párrafo de texto por fragmento. Además, dentro del documento, no existía la posibilidad de agregar otros elementos útiles para la elaboración de documentos más complejos tales como: listas, tablas, imágenes, encabezados, pies de página, notas a pie de página e índices. Por tanto se diseñó e implementó la estructura de un documento capaz de contener todos estos elementos con la capacidad además, de poder ser agregado al entorno distribuido del editor **ELXI**.

En cada uno de los fragmentos, se agregó un pequeño documento estructurado, el cual permite formar, una vez en el cliente, un documento completo el cual representa el total de fragmentos editados por los autores. Se implementó un analizador sintáctico, el cual toma los contenidos de cada fragmento del documento, después de ser distribuido a cada cliente. El analizador genera entonces el documento completo. Durante la distribución del documento, los contenidos de cada uno de los fragmentos son tomados sólo como texto.

Se diseñaron e implementaron las plantillas en XSL para permitir la correcta visualización de los documentos a ser editados en **ELXI**. Los formatos que se generan en el editor para cada uno de sus documentos son **HTML**, **PDF** Y **RTF**. Cada una de estas plantillas genera los documentos planteados como casos de estudio en el inicio de este trabajo: **Norma** y **General**. El documento General se propuso con el fin de lograr una estructura que pudiera cubrir la mayor cantidad de documentos técnicos y científicos posibles. Esto se logró al dejar abiertos cada uno de los fragmentos de un escrito a la inserción de elementos de edición, (listas, párrafos, imágenes, etc.) en el orden que el usuario creyera más conveniente para la elaboración de su documento. Del mismo modo, en el documento Norma se implementaron los nuevos elementos de edición logrando con esto generar este tipo de documento de forma distribuida desde el editor **ELXI**.

Se crearon las interfaces necesarias para el manejo de los elementos de edición, con el fin de lograr un manejo más sencillo del editor. Cada una de las interfaces permite agregar de forma transparente los elementos de edición de un documento sin que el usuario tenga que saber la sintaxis de edición de documentos de **ELXI**.

Se aporta, por lo anteriormente dicho, la estructura y sintaxis mejorada de un documento que puede utilizarse en editores colaborativos, además de la implementación tangible de estas ideas con el fortalecimiento funcional de editor colaborativo **ELXI**.

7.2. Contribuciones

Gracias al trabajo desarrollado en esta tesis, se tuvieron los siguientes logros

Documento General. Se logro la estructura y sintaxis de un documento útil para documentación técnica y científica. Este documento permite a **ELXI**, en su versión 2.0, editar documentos de forma colaborativa y generar visualizaciones de este documento de forma correcta en diferentes tipos de formatos.

Documento Norma. Se implementaron cada una de las mejoras anteriormente expuestas a el documento de tipo Norma que estaba implementado en la versión 1.0 de **ELXI**. Con esto se logró que la edición de documentos de normatividad en **ELXI** fuera más completa, obteniendo de este modo documentos enteramente útiles para la edición de este tipo de documentos.

Interfaces de usuario. Se generaron las interfaces de usuario necesarias para el manejo de los diferentes elementos de edición, evitando con esto que el autor del documento tenga la necesidad de aprender la sintaxis para el edición de documento. Con esto se logra un manejo más sencillo del editor, a diferencia, por ejemplo, de editores de documentos LaTeXdonde el autor debe saber necesariamente la sintaxis de generación de documentos.

Elementos de edición. Se generaron los elementos de edición (tablas, listas, imágenes, pies de página, notas a pie de página, índices) necesarios para la correcta visualización de los documento generados en **ELXI** en su versión 2.0. Estos elementos son agregados a objetos de flujo de plantillas XSL para lograr, por cada uno de ellos, una visualización aceptable en los formatos **PDF**, **HTML** Y **RTF** que genera el editor colaborativo **ELXI**.

7.3. Alcances y limitaciones

La nueva estructura de los documentos en **ELXI** permite agregar nuevos elementos de edición en los documentos generados. Para cada uno de estos nuevos elementos de edición se implementó una interfaz que pudiera administrar sus contenidos. Sin embargo, estas interfaces son susceptibles de ser mejoradas. Por ejemplo, aunque la nueva estructura del documento permite, sin ningún problema, la inserción de tablas anidadas, la interfaz que administra este elemento de edición sólo permite la generación de tablas sencillas.

Los mecanismos y algoritmos implementados para la estructura de los documentos se implementaron de forma satisfactoria, pero por razones de tiempo y recursos, algunas interfaces, en específico la de tabla, no explota en su totalidad las capacidades de la nueva estructura creada para los documentos de **ELXI**.

Cada uno de los fragmentos en **ELXI** tiene asignado un *tipo*. Este *tipo* define la presentación asignada al fragmento a través de las plantillas generadas en XSL. Si se desea asignar un tamaño o un estilo a una sección específica del documento, ésta tendrá que ser confinada al contenido de todo un fragmento. El fragmento es entonces visualizado dependiendo del *tipo* asignado a éste. Por lo anterior, como se observa, dentro de un fragmento no se pueden tener diferentes estilos y tamaños de letras.

La nueva sintaxis y estructura de los documentos en **ELXI** permite, sin ningún problema, asignar diferentes tamaños y estilos de letra dentro de un fragmento. Enriqueciendo las capacidades de los módulos y clases que manejan la interfaz de edición de los documentos se lograría, de forma similar a las interfaces de elementos de edición, usar por completo toda la nueva estructura de los documentos y lograr una mejor presentación de estos.

7.4. Beneficios

Se obtuvo la versión 2.0 del editor colaborativo ELXI. Se agregaron nuevos módulos y clases al editor ELXI, lo cual permitió robustecer y aumentar las características del editor. Las ideas planteadas al inicio de este trabajo fueron implementadas de forma tangible en el editor, permitiendo de esta forma mostrar la viabilidad y factibilidad de estas.

Se mejoró la estructura de los documentos Norma en ELXI. Se logró una mejor estructura y visualización de los documentos tipo **Norma** que **ELXI** implementaba en su versión 1.0. Esta nueva estructura y presentación permite generar documentos de normatividad de forma más clara y sencilla en **ELXI**.

Se implementó un documento estructurado General para ELXI. En la versión 1.0 de ELXI, sólo era posible visualizar documentos con plantillas XSL específicas a cada uno de ellos. En este trabajo se obtuvo un

documento de tipo **General** que dá la posibilidad al editor ELXI de generar documentos técnicos y científicos de forma general.

El trabajo realizado dejó también como aporte, la experiencia ganada en la realización de un proyecto de investigación industrial como lo fue este trabajo. Además de ganar la conciencia de saber que es posible competir y generar tecnología propia, aún a veces a contraviento, para generar bases más solidas en las cuales podamos apoyarnos para ayudar a crecer, desde la posición en la que estamos, a nuestro país.

Apéndice A

Código para la generación de documentos PDF en ELXI con XSL-FO

A.1. Código para la generación de tablas en formato PDF con XSL-FO

```
<xsl:template match="tabla" >
  <fo:table table-layout="fixed">
    <xsl:choose>
      <xsl:when test="@cols">
        <xsl:call-template name="build-columns">
          <xsl:with-param name="cols"
            select="concat(@cols, ' ')"/>
        </xsl:call-template>
      </xsl:when>
      <xsl:otherwise>
        <fo:table-column column-width="200pt"/>
      </xsl:otherwise>
    </xsl:choose>
    <fo:table-body>
      <xsl:apply-templates/>
    </fo:table-body>
```

```
    </fo:table>
  </xsl:template>

  <xsl:template name="build-columns">
    <xsl:param name="cols"/>

    <xsl:if test="string-length(normalize-space($cols))">
      <xsl:variable name="next-col">
        <xsl:value-of select="substring-before($cols, ' ')"/>
      </xsl:variable>
      <xsl:variable name="remaining-cols">
        <xsl:value-of select="substring-after($cols, ' ')"/>
      </xsl:variable>
      <xsl:choose>
        <xsl:when test="contains($next-col, 'pt')">
          <fo:table-column column-width="{$next-col}"/>
        </xsl:when>
        <xsl:when test="number($next-col) > 0">
          <fo:table-column column-width="{concat($next-col, 'pt')}"/
        </xsl:when>
        <xsl:otherwise>
          <fo:table-column column-width="50pt"/>
        </xsl:otherwise>
      </xsl:choose>

      <xsl:call-template name="build-columns">

        <xsl:with-param name="cols" select="concat($remaining-cols,
      </xsl:call-template>
    </xsl:if>
  </xsl:template>

  <!-- Renglon-->

  <xsl:template match="renglon">
    <fo:table-row>
      <xsl:apply-templates select="celda"/>
    </fo:table-row>
```

```
</xsl:template>

 <!--celda-->

<xsl:template match="celda">
  <fo:table-cell
    padding-start="1pt" padding-end="1pt"
    padding-before="1pt" padding-after="1pt">

    <xsl:if test="@colspan">
      <xsl:attribute name="number-columns-spanned">
        <xsl:value-of select="@colspan"/>
      </xsl:attribute>
    </xsl:if>
    <xsl:if test="@rowspan">
      <xsl:attribute name="number-rows-spanned">
        <xsl:value-of select="@rowspan"/>
      </xsl:attribute>
    </xsl:if>
    <xsl:if test="@border='1' or
                  ancestor::renglon[@border='1'] or
                  ancestor::tabla[@border='1']">
      <xsl:attribute name="border-style">
        <xsl:text>solid</xsl:text>
      </xsl:attribute>
      <xsl:attribute name="border-color">
        <xsl:text>black</xsl:text>
      </xsl:attribute>
      <xsl:attribute name="border-width">
        <xsl:text>2pt</xsl:text>
      </xsl:attribute>
    </xsl:if>

    <xsl:variable name="align">
      <xsl:choose>
        <xsl:when test="@align">
          <xsl:choose>
            <xsl:when test="@align='center'">
```

```
        <xsl:text>center</xsl:text>
      </xsl:when>
      <xsl:when test="@align='right'">
        <xsl:text>end</xsl:text>
      </xsl:when>
      <xsl:when test="@align='justify'">
        <xsl:text>justify</xsl:text>
      </xsl:when>
      <xsl:otherwise>
        <xsl:text>start</xsl:text>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:when>
  <xsl:when test="ancestor::renglon[@align]">
    <xsl:choose>
      <xsl:when test="ancestor::renglon/@align='center'">
        <xsl:text>center</xsl:text>
      </xsl:when>
      <xsl:when test="ancestor::renglon/@align='right'">
        <xsl:text>end</xsl:text>
      </xsl:when>
      <xsl:when test="ancestor::renglon/@align='justify'">
        <xsl:text>justify</xsl:text>
      </xsl:when>
      <xsl:otherwise>
        <xsl:text>start</xsl:text>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:when>

  <xsl:when test="ancestor::tabla[@align]">
    <xsl:choose>
      <xsl:when test="ancestor::tabla/@align='center'">
        <xsl:text>center</xsl:text>
      </xsl:when>
      <xsl:when test="ancestor::tabla/@align='right'">
        <xsl:text>end</xsl:text>
      </xsl:when>
```

```
            <xsl:when test="ancestor::tabla/@align='justify'">
              <xsl:text>justify</xsl:text>
            </xsl:when>
            <xsl:otherwise>
              <xsl:text>start</xsl:text>
            </xsl:otherwise>
          </xsl:choose>
        </xsl:when>
        <xsl:otherwise>
          <xsl:text>start</xsl:text>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:variable>

    <fo:block text-align="{$align}">

    <xsl:apply-templates select="texto"/>
    <xsl:apply-templates select="lista"/>
    <xsl:apply-templates select="listaN"/>
    <xsl:apply-templates select="listaD"/>
    <xsl:apply-templates select="tabla"/>

    </fo:block>
  </fo:table-cell>
</xsl:template>
```

A.2. Código para el manejo de texto en formato PDF con XSL-FO

```
<!-- Texto -->

<xsl:template match="texto">

  <xsl:choose>
```

```
<xsl:when test="ancestor::fragmento[@tipo='autores']">
      <fo:block
      space-after="10pt" start-indent=".1cm"
      font-family="sans-serif"
      white-space-collapse="false"
      wrap-option="no-wrap">
      <xsl:apply-templates/>
      </fo:block>
</xsl:when >

<xsl:when test="ancestor::fragmento[@tipo='subDos']">
<fo:block
space-after="10pt" start-indent="2.5cm"
text-align="justify"
font-family="sans-serif"
line-height="15pt">
<xsl:apply-templates/>
</fo:block>
</xsl:when>

<xsl:when test="ancestor::fragmento[@tipo='subTres']">
<fo:block
space-after="10pt" start-indent="3.5cm"
text-align="justify"
font-family="sans-serif"
line-height="15pt">
<xsl:apply-templates/>
</fo:block>
</xsl:when>

<xsl:when test="ancestor::fragmento[@tipo='subsection']">
<fo:block
space-after="10pt" start-indent="1.5cm"
text-align="justify"
font-family="sans-serif"
line-height="15pt">
<xsl:apply-templates/>
</fo:block>
```

```
</xsl:when>

<xsl:when test="ancestor::renglon | ancestor::elemento |ancestor::tabla">
<fo:block
space-after="10pt" start-indent="0.3cm"
text-align="justify"
font-family="sans-serif"
line-height="15pt">
<xsl:apply-templates/>
</fo:block>
</xsl:when>

<xsl:when test="ancestor::listaN">
<fo:block
space-after="10pt" start-indent="2.0cm"
text-align="justify"
font-family="sans-serif"
line-height="15pt">
<xsl:apply-templates/>
</fo:block>
</xsl:when>

<xsl:when test="ancestor::listaN/elementoN">
<fo:block
space-after="10pt" start-indent="2.5cm"
text-align="justify"
font-family="sans-serif"
line-height="15pt">
<xsl:apply-templates/>
</fo:block>
</xsl:when>

<xsl:when test="ancestor::fragmento[@tipo='Pie_de_pagina']">

<fo:block
space-after="10pt" start-indent="1cm"
text-align="center"
```

```
 font-family="sans-serif"
 line-height="15pt">
 <xsl:apply-templates/>
 </fo:block>
 </xsl:when>

 <xsl:otherwise>
 <fo:block
 space-after="10pt" start-indent="1cm"
 text-align="justify"
 font-family="sans-serif"
 line-height="15pt">
 <xsl:apply-templates/>
 </fo:block>
 </xsl:otherwise>
 </xsl:choose>

</xsl:template>
```

A.3. Código para el manejo de secciones y subsecciones en formato PDF con XSL-FO.

```
<xsl:template
match="fragmento[@tipo='section']|fragmento[@tipo='subsection']
      |fragmento[@tipo='subDos']|fragmento[@tipo='subTres']">

    <xsl:if test="@tipo='section' ">
    <xsl:for-each select=".">
        <fo:block padding-top="5pt"
          text-align="justify"
          font-family="sans-serif"
          font-size="14pt"
          line-height="18pt"
          border-start-width=".1mm"
          id="{generate-id()}">
```

```
      <xsl:apply-templates select="@titulo"/>
      </fo:block>
    <fo:block font-size="10pt"
    space-after="10pt" start-indent="1cm"
      font-family="sans-serif"
      line-height="15pt"
      space-after.optimum="10pt"
      text-align="justify">
       <xsl:apply-templates/>
      </fo:block>
</xsl:for-each>
</xsl:if>

<xsl:if test="@tipo='subsection'">
<xsl:for-each select=".">
  <fo:block padding-top="5pt"
      space-after="10pt" start-indent="0.5cm"
      text-indent="1em"
      text-align="justify"
      font-family="sans-serif"
      font-size="14pt"
      line-height="18pt"
      border-start-width=".1mm"
      id="{generate-id()}">
    <xsl:apply-templates select="@titulo"/>
  </fo:block>
     <fo:block font-size="10pt"
     space-after="10pt" start-indent="1.5cm"
        font-family="sans-serif"
        line-height="15pt"
        space-after.optimum="10pt"
        text-align="left">
    <xsl:apply-templates/>
  </fo:block>
</xsl:for-each>
</xsl:if>

<xsl:if test="@tipo='subDos'">
```

```
<xsl:for-each select=".">
       <fo:block padding-top="5pt"
       space-after="10pt" start-indent="1.5cm"
      text-indent="1em"
      text-align="justify"
      font-family="sans-serif"
      font-size="14pt"
      line-height="18pt"
      border-start-width=".1mm"
      id="{generate-id()}">
    <xsl:apply-templates select="@titulo"/>
  </fo:block>
     <fo:block font-size="10pt"
     space-after="10pt" start-indent="2.5cm"
        font-family="sans-serif"
        line-height="15pt"
        space-after.optimum="10pt"
        text-align="left"
        border-start-width="1cm">
    <xsl:apply-templates/>
  </fo:block>
</xsl:for-each>
</xsl:if>

<xsl:if test="@tipo='subTres'">
<xsl:for-each select=".">
       <fo:block padding-top="5pt"
       space-after="10pt" start-indent="2.5cm"
      text-indent="1em"
      text-align="justify"
      font-family="sans-serif"
      font-size="14pt"
      line-height="18pt"
      border-start-width=".1mm"
      id="{generate-id()}">
    <xsl:apply-templates select="@titulo"/>
  </fo:block>
     <fo:block font-size="10pt"
```

```
        space-after="10pt" start-indent="3.5cm"
           font-family="sans-serif"
           line-height="15pt"
           space-after.optimum="10pt"
           text-align="left">
         <xsl:apply-templates/>
       </fo:block>
    </xsl:for-each>
    </xsl:if>
</xsl:template>
```

A.4. Código para la creación de índices en formato PDF con XSL-FO

```
<xsl:template name="indice"  >

<fo:block font-size="10pt"
    break-before="page"
    font-family="sans-serif"
    line-height="10pt"
    space-before.optimum="5pt"
    space-after.optimum="5pt"
    background-color="white"
    text-align="center"
    padding-top="35pt"
    font-variant="small-caps"
    >
    CONTENIDO
  </fo:block>

<fo:block font-size="10pt"
    font-family="sans-serif"
    line-height="10pt"
    space-before.optimum="5pt"
    space-after.optimum="5pt"
```

```
    white-space-collapse="false"
    wrap-option="no-wrap"
    >CAPÍTULO
  </fo:block>

<xsl:for-each select="//fragmento[@tipo='introduccion'] |
                      //fragmento[@tipo='objetivo'] |
                      //fragmento[@tipo='alcance'] |
                      //fragmento[@tipo='campo de aplicacion'] |
                      //fragmento[@tipo='actualizacion'] |
                      //fragmento[@tipo='referencias'] |
                      //fragmento[@tipo='definiciones'] |
                      //fragmento[@tipo='simbolos y abreviaturas
                      //fragmento[@tipo='section']|
                      //fragmento[@tipo='subsection'] |
                      //fragmento[@tipo='subDos']|
                      //fragmento[@tipo='subTres'] |
                      //fragmento[@tipo='responsabilidades'] |
            //fragmento[@tipo='concordancia con otras normas'] |
                      //fragmento[@tipo='bibliografia'] |
                      //fragmento[@tipo='anexos'] ">
<xsl:call-template name="toc"/> </xsl:for-each>
<fo:blockbreak-after="page"> </fo:block> </xsl:template>

<xsl:template name="toc"  >

    <xsl:param name="toc-level-default" select="3"/>
    <xsl:variable name="toc-level-max">
        <xsl:value-of select="$toc-level-default"/>
    </xsl:variable>

<xsl:choose>

    <xsl:when test="ancestor-or-self::fragmento[@tipo='introducci
    ancestor-or-self::fragmento[@tipo='objetivo'] |
    ancestor-or-self::fragmento[@tipo='alcance'] |
    ancestor-or-self::fragmento[@tipo='campo de aplicacion'] |
    ancestor-or-self::fragmento[@tipo='actualizacion'] |
```

```
ancestor-or-self::fragmento[@tipo='referencias'] |
ancestor-or-self::fragmento[@tipo='definiciones']|
ancestor-or-self::fragmento[@tipo='simbolos y abreviaturas'] |
ancestor-or-self::fragmento[@tipo='section'] |
ancestor-or-self::fragmento[@tipo='responsabilidades']|
ancestor-or-self::fragmento[@tipo='concordancia con otras normas'] |
ancestor-or-self::fragmento[@tipo='bibliografia'] |
ancestor-or-self::fragmento[@tipo='anexos'] ">
<xsl:variable name="level" select="1" />

        <fo:block text-align-last="justify"
        white-space-collapse="false"
        wrap-option="no-wrap"
        font-size="10pt"
        font-family="sans-serif">

        <xsl:attribute name="margin-left">
            <xsl:value-of select="$level - 1"/>
            <xsl:text>em</xsl:text>
        </xsl:attribute>

        <xsl:attribute name="space-before">
        <xsl:choose>
            <xsl:when test="$level=1">5pt</xsl:when>
            <xsl:when test="$level=2">3pt</xsl:when>
            <xsl:when test="$level=3">1pt</xsl:when>
        <xsl:otherwise>1pt</xsl:otherwise>
        </xsl:choose>
        </xsl:attribute>

        <xsl:attribute name="font-size">
        <xsl:choose>
            <xsl:when test="$level=1">1em</xsl:when>
            <xsl:otherwise>0.9em</xsl:otherwise>
        </xsl:choose>
        </xsl:attribute>

            <fo:basic-link color="blue"
```

```
                internal-destination="{generate-id()}">

                    <xsl:value-of select="@titulo" />
                    <xsl:value-of select="$level"/>

                </fo:basic-link>

                <fo:leader leader-pattern="dots"
                leader-pattern-width="5pt"/>
                <fo:page-number-citation
                ref-id="{generate-id()}"/>
            </fo:block>


        <!-- <Para las siguientes secciones> -->

          </xsl:when>

              <xsl:when test=
              "ancestor-or-self::fragmento[@tipo='subsection']">
                  <xsl:variable name="level" select="2" />

            Continua...
```

Apéndice B

Cuestionarios de evaluación aplicados

B.1. Cuestionario

Instrucciones: ***Por favor responda a cada una de las preguntas, dentro de los corchetes cuadrados, según la escala de valores siguiente.***

1. Acerca de los elementos de edición y formatos de visualización

1.1 ¿Cómo califica usted el número de formatos, disponibles en ELXI, a los que se pueden exportar los documentos XML. (Los formatos disponibles son: *Word*, *HTML*, *PDF*)? ¿Propone usted algún otro?

1.2 ¿Cual es su opinion respecto al número de tipos de fragmento disponible para un documento General (ej. título, párrafo, pie de página, etc.)? ¿Algún comentario justificando su respuesta? ¿Cree usted que son suficientes? ¿Propondría usted algunos otros? ¿Cuales?

1.3 ¿Cual es su opinion respecto al formato de visualización obtenido para un documento *HTML*? ¿Algún comentario justificando su respuesta? ¿Propondría usted algún cambio o mejora?

1.4 ¿Cual es su opinion respecto al formato de visualización obtenido para un documento *Word*? ¿Algún comentario justificando su respuesta? ¿Propondría usted algún cambio o mejora?

1.5 ¿Cual es su opinion respecto al formato de visualización obtenido para un documento *PDF*? ¿Algún comentario justificando su respuesta? ¿Propondría usted algún cambio o mejora?

2. Acerca de las interfaces de los elementos de edición

2.1 ¿Cual es su opinion respecto a la interfaz que sirve para manipular un elemento *lista*? ¿Algún comentario justificando su respuesta? ¿Propondría usted algún cambio o mejora?

2.2 ¿Cual es su opinion respecto a la interfaz que sirve para manipular un elemento *tabla*? ¿Algún comentario justificando su respuesta? ¿Propondría usted algún cambio o mejora?

2.3 ¿Cual es su opinion respecto a la interfaz que sirve para manipular un elemento *imagen*? ¿Algún comentario justificando su respuesta? ¿Propondría usted algún cambio o mejora?

3. Acerca de la distribución de fragmentos.

3.1 ¿Cómo califica usted el tiempo que transcurre, después de seleccionar la pestaña *ver*, con el fin de obtener la visualización del documento posterior a su edición? ¿Algún comentario justificando su respuesta? ¿Propondría usted algún cambio o mejora?

3.2 ¿Cual es su opinion en relación a que dentro de un fragmento se puedan tener uno o más párrafos, junto con uno o más elementos de edición (listas, tablas, imágenes, etc.)? ¿Algún comentario justificando su respuesta? ¿Propondría usted algún cambio o mejora?

B.2. Cuestionarios aplicados

Bibliografía

[1] C. A. Ellis, S. J. Gibbs, G. L. Rein, *Groupware: Some Issues and Experiences*, Communications of the ACM, 34(1), pp. 38-58, enero 1991.

[2] K. Sikkel, *A Group-based Authorization Model for Cooperative Systems*, Actas de la conferencia European Conference on Computer-Supported Cooperative Work (ECSCW'97), pp. 345-360, Lancaster, septiembre 1997.

[3] Z. Ercegovac, *Introduction*, Journal of the American Society for Information Science, 50(13), pp. 1165-1168, noviembre 1999.

[4] L. Grossman, *Search and Destroy*, Time, 162(25), pp. 46-50, diciembre 2003 .

[5] M. Bryan, *SGML: an author´s guide*, New York: Addison-Wesley, Reading, USA, 1998.

[6] T. Berners-Lee, O. Lassila, J. Hendler, *The Semantic Web*, Scientific American, 279(5), mayo 2001.

[7] Art Rhyno, *Is XML in your Future?*, North American Serials Interest Group, 42 (1/2), pp. 143-153, mayo 2001.

[8] C. Lagoze, *The Warwick Framework: A Container Architecture for Diverse Sets of Metadata*, D-Lib Magazine, Julio-Agosto 1996, http://www.dlib.org/dlib/july96/lagoze/07lagoze.html.

[9] Ignacio Barrancos Martínez, *XML para todos*, GNU Free Documentation, abril 2003, http://www.um.es/linux/xml/ponencia/XML_paratodos.html.

[10] Jorge M. Haake, Brian Wilson, *Supporting collaborative writing of Hyperdocuments in SEPIA*, Actas de la conferencia CSCW92 SIGHI/SIGOIS ACM New York, pp. 138-146, noviembre 1992.

[11] F. Pacull, A. Sandoz, A. Schiper, *DUPLEX: a distributed collaborative editing environment in large scale*, Actas de la conferencia ACM CSCW94, Computer Supported Cooperative Work, pp. 165-173, 1994.

[12] Y. Yang, C. Sun, Y. Zhang, X. Jia. *Real-time Cooperative Editing on the Internet*, IEEE Computing, pp. 18-25, mayo-junio 2000.

[13] D. Decouchant, V. Quint, M. Romero Salcedo. *Structured and distributed editing in a large-scale network*, Chapter 3, pp. 265-295 del libro Groupware and Authoring, Academic Press, London, mayo 1996.

[14] V. Quint, I. Vatton, *Grif: an Interactive system for structured document manipulation.* Actas de la conferencia Text for Structured Document Manipulation, pp. 200-213, Nottingham U.K., abril 1986,

[15] A. Boukottaya, C. Vanoirbeek, et.al., *A contract based model for creating structured virtual documents: key issues for reusability and collaboration*, Swiss Federal Institute of Technology (EPFL), Actas de la conferencia Documents Virtuels Personnalisables, julio 2002.

[16] F. Buscmann, R. Meunier, et-al., *A system of patterns.* John Wiley and sons Ltd. 1996.

[17] Anibal Jesus Avelar Rosales. *Diseño, implementación y evaluación de un editor de documentos XML en Java para el sistema de autoría colaborativa ELXI.* Pogrado en Ciencias e Ingeniería de la Computación. IIMAS–UNAM. 9 diciembre 2004.

[18] Silvia Ramirez. *Diseño e Implementación de Mecanismos de Control de la Concurrencia para un Sistema de Edición Colaborativa en Internet e Intranet.* Pogrado en Ciencias e Ingeniería de la Computación. IIMAS–UNAM. 8 de Diciembre 2003.

[19] M. Pacheco, S. Picazo, E Diaz, *Guía para la emisión de normas de referencia de Petróleos Mexicanos y organismos subsidiarios.* Comité de normalización de Petróleos Mexicanos y organismos subsidiarios. 2001.

[20] Iona R. Posner and Ronald M. Baecker. *How People Write Together.* Actas de la conferencia 25th Hawaii International Conference on System Sciences, vol 4, pp. 127-138, 1992.

[21] R. Furuta, V. Quint, J André, *Interactively Editing Structured Documents*, Electronic Publishing Origination, Dissemination and Design, 1(1), pp. 19-44, abril 1988.

[22] R. Heery, *Review of metadata formats*, UKOLN University of Bath, http://www.ukoln.ac.uk/metadata/review.html, [5 agosto 2004].

[23] Norman Walsh, *DocBook: The Definitive Guide*, O´Reilly and Associates, Inc., http://www.docbook.org, [9 agosto 2004].

[24] Philippe Le Hégaret, *Document Object Model (DOM)*, W3C Architecture domain, http://www.w3.org/DOM, [4 noviembre 2004].

[25] James Clark, *XML Path Language (XPath)*, W3C, http://www.w3.org-/TR/xpath, [8 agosto 2004].

[26] Sun Microsystems Inc, *Java Remote Method Invocation (Java RMI)*, developers.sun.com, http://java.sun.com/products/jdk/rmi, [5 julio 2004].

[27] Microsoft Corporation, *Microsoft SharePoint Products and Tecnologies*, SharePoint Home, http://www.microsoft.com/sharepoint/. [16 dic 2004].

[28] Sun Microsystems Inc, *RFC 1014 (RFC1014)*, Internet RFC/STD/FYI-/BCP Archives, http://www.faqs.org/rfcs/rfc1014.html, [2 agosto 2004].

[29] Web-team Lyx, *The Document Processor*, LyX developers'site, http://www.lyx.org, [26 julio 2004].

[30] Microsoft Corporation, *MSN Messenger para Windows*, Messenger Home, http://messenger.msn.com. [23 julio 2004].

[31] TEI Consortium, *Text Encoding Initiative*, TEI Home, http://www.tei-c.org, [22 agosto 2004].

[32] Dan Connolly, *Overview of SGML Resources*, W3C, http://www.w3.org/MarkUp/SGML, [11 julio 2004].

[33] OASIS Open, *OASIS*, OASIS Home, http://www.oasis-open.org/home/index.php, [23 agosto 2004].

[34] Dave Clark, *Federal Geographic Data Committee*, FGDC Web Site, http://www.fgdc.gov, [14 noviembre 2004].

[35] Dublin Core Metadata Initiative, *DCMI Making it easier to find information*, DCMI Home, http://dublincore.org, [9 agosto 2004].

[36] Library of Congress, *Encoded Archival Description (EAD)*, Official EAD Version 2002 Web Site, http://www.loc.gov/ead. [3 septiembre 2004].

[37] Bert Bos, *Cascading Style Sheets home page*, W3C, http://www.w3.org/Style/CSS, [25 julio 2004].

[38] HTML Working Group, *XHTML 1.0 The Extensible HyperText Markup Language (Second Edition)*, W3C, http://www.w3.org/TR/xhtml1, [2 julio 2004].

[39] Eric Miller, *Semantic Web*, W3C Technology and Society Domain, http://www.w3.org/2001/sw, [24 noviembre 2004].

[40] Microsoft Corporation, http://www.microsoft.com/windows/netmeeting, *Windows NetMeeting*, NetMeeting Home, [19 julio 2004].

[41] Liam Quin, *Extensible Markup Language (XML)*, W3C, http://www.w3.org/XML, [13 julio 2004].

[42] Sun Microsystems Inc., *Products and Technologies Java Technology*, Java Home, http://java.sun.com, [7 julio 2004].

[43] Kevin S. Clarke, *Medlane*, Lane Medical Library, http://xmlmarc.stanford.edu, [28 agosto 2004].

[44] Apache Project, *Xalan-Java version 2.6.0*, xalan-j Home, http://xml.apache.org/xalan-j, [24 julio 2004].

[45] Apache Project, *Xerces Java Parser 1.4.4 Release*, xerces-j Home, http://xml.apache.org/xerces-j, [20 julio 2004].

[46] Apache Project, *FOP*, The Apache XML Graphics Project, http://xml.apache.org/fop, [12 septiembre 2004].

[47] Microsoft Corporation, *Word 2003 Product Information*, Office Home, http://www.microsoft.com/office/word/prodinfo/default.mspx, [13 agosto 2004].

[48] LaTeX3 project, *LaTeX A document preparation system*, LaTeX-project, http://www.latex-project.org, [16 octubre 2004].

[49] Don Wells, *Extreme Programming a gentle introduction*, Don Wells Site, http://www.extremeprogramming.org, [3 enero 2004].

Cuestionario de evaluación parcial

Fase de prueba piloto del prototipo ELXI

Nombre: Patricia López Rivera
Extensión: 9175
E-mail: patricialr@correo.unam.mx
Adscripción: PUMAGE

Instrucciones: Por favor responda a cada una de las preguntas, dentro de los corchetes cuadrados, según la escala de valores siguiente:

1 - Muy Bien
2- Bien
3- Suficiente
4- Mal

Además, para cada pregunta, le rogamos nos aporte algún **comentario** que apoye su respuesta.

1. Acerca de los elementos de edición y formatos de visualización

1.1 ¿Cómo califica usted el número de formatos, disponibles en ELXI, a los que se pueden exportar los documentos *XML* (los disponibles son: *Word, HTML, PDF*)? [1]

Algún comentario justificando su respuesta:
Es el único que he usado

¿Propondría usted algún otro formato?
No por el momento

1.2 ¿Cuál es su opinión respecto al número de tipos de fragmento disponibles para un documento *General (ej. título, párrafo, pie de página, etc.)?* [1]

Algún comentario justificando su respuesta:
No ha funcionado bien en esta estructura.

¿Cree usted que son suficientes? ¿Propondría usted algunos otros ? ¿Cúales?
Si me han sido suficientes.

1.3 ¿Cuál es su opinión respecto al formato de visualización obtenido para un documento *HTML*? [ ]

Algún comentario justificando su respuesta:
No lo he usado

¿Propondría usted algún cambio o mejora?

1.4 ¿Cuál es su opinión respecto al formato de visualización obtenido para un documento *Word*? [1]

Algún comentario justificando su respuesta:

Funciona apropiadamente.

¿Propondría usted algún cambio o mejora?

por el momento No

1.5 ¿Cuál es su opinión respecto al formato de visualización obtenido para un documento *PDF*? [ ]

Algún comentario justificando su respuesta:

No lo he usado.

¿Propondría usted algún cambio o mejora?

2. Acerca de las interfaces de los elementos de edición

2.1 ¿Cuál es su opinión respecto a la interfaz que sirve para manipular un elemento *lista*? [1]

Algún comentario justificando su respuesta:

Me ha funcionado satisfactoriamente

¿Propondría usted algún cambio o mejora?

2.2 ¿Cuál es su opinión respecto a la interfaz que sirve para manipular un elemento *tabla*? [ ]

Algún comentario justificando su respuesta:

No lo he usado.

¿Propondría usted algún cambio o mejora?

2.3 ¿Cuál es su opinión respecto a la interfaz que sirve para manipular un elemento *imagen*? [ ]

Algún comentario justificando su respuesta:

No lo he usado

¿Propondría usted algún cambio o mejora?

3. Acerca de la distribución de fragmentos

3.1 ¿Cómo califica usted el tiempo que transcurre, después de seleccionar la pestaña *"ver"*, con el fin de obtener la visualización del documento posterior a su edición? [ ] 1

Algún comentario justificando su respuesta:
No se tanto Mucho, al menos en el tamaño de documentos que he usado

¿Propondría usted algún cambio o mejora?

3.2 ¿Cuál es su opinión en relación a que dentro de un fragmento se puedan tener uno o más párrafos, junto con uno o más elementos de edición (*listas, tablas, imágenes, etc.*)? [1]

Algún comentario justificando su respuesta:
Comunmente los documentos q manejamos necesitarían de esta opción aunque por el momento no la he usado.

¿Desearía usted poder compartir con algún colega parte de una *lista*, una *tabla*, un *párrafo, etc.*?
¿Propondría usted algún cambio o mejora?

—¿La manera en que la hemos estado usando, compartiendo párrafos nos ha funcionado bien así como el área de mensajes.

Le agradecemos su tiempo y valiosa colaboración. Todas las respuestas que nos ha brindado nos serán de gran ayuda para mejorar nuestro producto.

Cuestionario de evaluación parcial

Fase de prueba piloto del prototipo ELXI

Nombre: Benjamín Cordero Soriano
Extensión: 6338
E-mail: bcordero@imp.mx
Adscripción: DEPyAI

Instrucciones: Por favor responda a cada una de las preguntas, dentro de los corchetes cuadrados, según la escala de valores siguiente:

1 - Muy Bien
2- Bien
3- Suficiente
4- Mal

Además, para cada pregunta, le rogamos nos aporte algún comentario que apoye su respuesta.

1. Acerca de los elementos de edición y formatos de visualización

1.1 ¿Cómo califica usted el número de formatos, disponibles en ELXI, a los que se pueden exportar los documentos *XML* (los disponibles son: *Word, HTML, PDF*)? [ ]

SUFICIENTE

Algún comentario justificando su respuesta:

¿Propondría usted algún otro formato?

1.2 ¿Cuál es su opinión respecto al número de tipos de fragmento disponibles para un documento *General (ej. título, párrafo, pie de página, etc.)?* [ ]

SUFICIENTE

Algún comentario justificando su respuesta:

¿Cree usted que son suficientes? ¿Propondría usted algunos otros ? ¿Cúales?

LOS ESPECIFICOS DEL AREA, SEGUN LA NATURALEZA DE DOCUMENTO

1.3 ¿Cuál es su opinión respecto al formato de visualización obtenido para un documento *HTML*? [ ]

SUFICIENTE

Algún comentario justificando su respuesta:

COMPARADO CON MICROSOFT.

¿Propondría usted algún cambio o mejora?

MAYORES FACILIDADES O VENTAJAS.

1.4 ¿Cuál es su opinión respecto al formato de visualización obtenido para un documento *Word*? [ ]
SUFICIENTE
Algún comentario justificando su respuesta:

¿Propondría usted algún cambio o mejora?

1.5 ¿Cuál es su opinión respecto al formato de visualización obtenido para un documento *PDF*? [ ]
SUFICIENTE
Algún comentario justificando su respuesta:

¿Propondría usted algún cambio o mejora?

2. Acerca de las interfaces de los elementos de edición

2.1 ¿Cuál es su opinión respecto a la interfaz que sirve para manipular un elemento *lista*? [ ]
SUFICIENTE
Algún comentario justificando su respuesta:

¿Propondría usted algún cambio o mejora?

2.2 ¿Cuál es su opinión respecto a la interfaz que sirve para manipular un elemento *tabla*? [ ]
SUFICIENTE
Algún comentario justificando su respuesta:

¿Propondría usted algún cambio o mejora?

2.3 ¿Cuál es su opinión respecto a la interfaz que sirve para manipular un elemento *imagen*? [ ]

Algún comentario justificando su respuesta:
NO LO APLICAMOS EN ESTE EJERCICIO.

¿Propondría usted algún cambio o mejora?

3. Acerca de la distribución de fragmentos

3.1 ¿Cómo califica usted el tiempo que transcurre, después de seleccionar la pestaña *"ver"*, con el fin de obtener la visualización del documento posterior a su edición? [ ]

Bien

Algún comentario justificando su respuesta:

¿Propondría usted algún cambio o mejora?

3.2 ¿Cuál es su opinión en relación a que dentro de un fragmento se puedan tener uno o más párrafos, junto con uno o más elementos de edición (*listas, tablas, imágenes, etc.*)? [ ]

SUFICIENTES PARA TEXTOS

Algún comentario justificando su respuesta:

NO MANEJAMOS IMAGENES

¿Desearía usted poder compartir con algún colega parte de una *lista*, una *tabla*, un *párrafo*, *etc.*? ¿Propondría usted algún cambio o mejora?

Le agradecemos su tiempo y valiosa colaboración. Todas las respuestas que nos ha brindado nos serán de gran ayuda para mejorar nuestro producto.

Cuestionario de evaluación parcial

Fase de prueba piloto del prototipo ELXI

Nombre: Fernando Ferreira Becerra
Extensión: 6696
E-mail: ffbecerra@[illegible]
Adscripción: Admón y Finanzas

Instrucciones: Por favor responda a cada una de las preguntas, dentro de los corchetes cuadrados, según la escala de valores siguiente:

1 - Muy Bien
2- Bien
3- Suficiente
4- Mal

Además, para cada pregunta, le rogamos nos aporte algún **comentario** que apoye su respuesta.

1. Acerca de los elementos de edición y formatos de visualización

1.1 ¿Cómo califica usted el número de formatos, disponibles en ELXI, a los que se pueden exportar los documentos *XML* (los disponibles son: *Word, HTML, PDF*)? [2]

Algún comentario justificando su respuesta: Es bueno, pero quiza sería importante poder manejar algún otro.

¿Propondría usted algún otro formato? Si, Excel

1.2 ¿Cuál es su opinión respecto al número de tipos de fragmento disponibles para un documento *General (ej. título, párrafo, pie de página, etc.)?* [1]

Algún comentario justificando su respuesta: Me parece bueno

¿Cree usted que son suficientes? ¿Propondría usted algunos otros ? ¿Cúales? Son suficientes

1.3 ¿Cuál es su opinión respecto al formato de visualización obtenido para un documento *HTML*? [1]

Algún comentario justificando su respuesta: Me parece bueno, aunque sería bueno tenerlo como el original, en este caso el que más [illegible]

¿Propondría usted algún cambio o mejora?

1.4 ¿Cuál es su opinión respecto al formato de visualización obtenido para un documento *Word*? [1]

Algún comentario justificando su respuesta: Es bueno, aunque -

¿Propondría usted algún cambio o mejora? Sería muy bueno poder checar el Documento tal como se presenta el original

1.5 ¿Cuál es su opinión respecto al formato de visualización obtenido para un documento *PDF*? [2]

Algún comentario justificando su respuesta: bueno, aunque no los utilizamos PDF.

¿Propondría usted algún cambio o mejora?

2. Acerca de las interfaces de los elementos de edición

2.1 ¿Cuál es su opinión respecto a la interfaz que sirve para manipular un elemento *lista*? [/]

Algún comentario justificando su respuesta: Me parece muy bueno por que nos dice como se puede manipular las listas

¿Propondría usted algún cambio o mejora?

2.2 ¿Cuál es su opinión respecto a la interfaz que sirve para manipular un elemento *tabla*? [/]

Algún comentario justificando su respuesta: Es bueno para cuando se requiere utilizar alguna tabla

¿Propondría usted algún cambio o mejora?

2.3 ¿Cuál es su opinión respecto a la interfaz que sirve para manipular un elemento *imagen*? [/]

Algún comentario justificando su respuesta: Es bueno porque es muy factible que se requiera agregar alguna imagen

¿Propondría usted algún cambio o mejora?

3. Acerca de la distribución de fragmentos

3.1 ¿Cómo califica usted el tiempo que transcurre, después de seleccionar la pestaña *"ver"*, con el fin de obtener la visualización del documento posterior a su edición? [✓]

Algún comentario justificando su respuesta: Me parece un tiempo adecuado

¿Propondría usted algún cambio o mejora? Quizá que fuera un poco más rápido.

3.2 ¿Cuál es su opinión en relación a que dentro de un fragmento se puedan tener uno o más párrafos, junto con uno o más elementos de edición (*listas, tablas, imágenes, etc.*)? [✓]

Algún comentario justificando su respuesta: Me parece bien ya que en algunas ocasiones se requiere estar trabajando con varios [illegible] elementos de la edición (listas, tablas, imágenes, etc.)

¿Desearía usted poder compartir con algún colega parte de una *lista*, una *tabla*, un *párrafo*, *etc.*?

¿Propondría usted algún cambio o mejora? Sería bueno probarlo.

Le agradecemos su tiempo y valiosa colaboración. Todas las respuestas que nos ha brindado nos serán de gran ayuda para mejorar nuestro producto.

Cuestionario de evaluación parcial

Fase de prueba piloto del prototipo ELXI

Nombre: Beatriz B. Morales Díaz
Extensión: 7543
E-mail: bmorales@imp.mx
Adscripción: PIMAyC / Investigación

Instrucciones: Por favor responda a cada una de las preguntas, dentro de los corchetes cuadrados, según la escala de valores siguiente:

1 - Muy Bien
2- Bien
3- Suficiente
4- Mal

Además, para cada pregunta, le rogamos nos aporte algún **comentario** que apoye su respuesta.

1. Acerca de los elementos de edición y formatos de visualización

1.1 ¿Cómo califica usted el número de formatos, disponibles en ELXI, a los que se pueden exportar los documentos *XML* (los disponibles son: *Word, HTML, PDF*)? [ ] Bien

Algún comentario justificando su respuesta:
Es la documentación que más se maneja; falta mencionar la posibilidad a LATEX

¿Propondría usted algún otro formato?
Latex

1.2 ¿Cuál es su opinión respecto al número de tipos de fragmento disponibles para un documento *General (ej. título, párrafo, pie de página, etc.)?* [ ] Bien

Algún comentario justificando su respuesta:

¿Cree usted que son suficientes? ¿Propondría usted algunos otros ? ¿Cúales?
Son suficientes

1.3 ¿Cuál es su opinión respecto al formato de visualización obtenido para un documento *HTML*? [ ] Bien

Algún comentario justificando su respuesta:
Es suficiente para el manejo de documentos

¿Propondría usted algún cambio o mejora?
No

1.4 ¿Cuál es su opinión respecto al formato de visualización obtenido para un documento *Word*? [ ]
Suficiente
Algún comentario justificando su respuesta:
Si se llegan a perder algunos tipos de word a ELXI
¿Propondría usted algún cambio o mejora?
Incluir más tipos, p. ej. griegos

1.5 ¿Cuál es su opinión respecto al formato de visualización obtenido para un documento *PDF*? [ ]
Bien
Algún comentario justificando su respuesta:
No ni ningún problema
¿Propondría usted algún cambio o mejora?

2. Acerca de las interfaces de los elementos de edición

2.1 ¿Cuál es su opinión respecto a la interfaz que sirve para manipular un elemento *lista*? [ ]
Suficiente
Algún comentario justificando su respuesta:
es raro que las opciones son con botón izquierdo, es diferente a lo convencional
¿Propondría usted algún cambio o mejora?
Indicarlo claramente ó cambiarlo

2.2 ¿Cuál es su opinión respecto a la interfaz que sirve para manipular un elemento *tabla*? [ ]
Suficiente
Algún comentario justificando su respuesta:
Es adecuada pero no es tan versátil como Latex, word, entre otros, esto impide manejar documentos de uso frecuente.
¿Propondría usted algún cambio o mejora?

2.3 ¿Cuál es su opinión respecto a la interfaz que sirve para manipular un elemento *imagen*? [ ]
No lo use
Algún comentario justificando su respuesta:

¿Propondría usted algún cambio o mejora?

3. Acerca de la distribución de fragmentos

3.1 ¿Cómo califica usted el tiempo que transcurre, después de seleccionar la pestaña *"ver"*, con el fin de obtener la visualización del documento posterior a su edición? [ ] Suficiente

Algún comentario justificando su respuesta:
No

¿Propondría usted algún cambio o mejora?

3.2 ¿Cuál es su opinión en relación a que dentro de un fragmento se puedan tener uno o más párrafos, junto con uno o más elementos de edición (*listas, tablas, imágenes, etc.*)? [ ]
Bien
Algún comentario justificando su respuesta:
No

¿Desearía usted poder compartir con algún colega parte de una *lista*, una *tabla*, un *párrafo, etc.*? ¿Propondría usted algún cambio o mejora?

Le agradecemos su tiempo y valiosa colaboración. Todas las respuestas que nos ha brindado nos serán de gran ayuda para mejorar nuestro producto.

Cuestionario de evaluación parcial

Fase de prueba piloto del prototipo ELXI

Nombre: César Hudiel García
Extensión: 7593
E-mail: cesar_hudiel_garcia@hotmail.com
Adscripción:

Instrucciones: Por favor responda a cada una de las preguntas, dentro de los corchetes cuadrados, según la escala de valores siguiente:

1 - Muy Bien
2- Bien
3- Suficiente
4- Mal

Además, para cada pregunta, le rogamos nos aporte algún **comentario** que apoye su respuesta.

1. Acerca de los elementos de edición y formatos de visualización

1.1 ¿Cómo califica usted el número de formatos, disponibles en ELXI, a los que se pueden exportar los documentos *XML* (los disponibles son: *Word, HTML, PDF*)? [2]

Algún comentario justificando su respuesta:

¿Propondría usted algún otro formato?
Excel

1.2 ¿Cuál es su opinión respecto al número de tipos de fragmento disponibles para un documento *General (ej. título, párrafo, pie de página, etc.)?* [2]

Algún comentario justificando su respuesta:

¿Cree usted que son suficientes? ¿Propondría usted algunos otros ? ¿Cúales?

1.3 ¿Cuál es su opinión respecto al formato de visualización obtenido para un documento *HTML*? [1]

Algún comentario justificando su respuesta:

¿Propondría usted algún cambio o mejora?

1.4 ¿Cuál es su opinión respecto al formato de visualización obtenido para un documento *Word*? [1]

Algún comentario justificando su respuesta:

¿Propondría usted algún cambio o mejora?

1.5 ¿Cuál es su opinión respecto al formato de visualización obtenido para un documento *PDF*? [1]

Algún comentario justificando su respuesta:

¿Propondría usted algún cambio o mejora?

2. Acerca de las interfaces de los elementos de edición

2.1 ¿Cuál es su opinión respecto a la interfaz que sirve para manipular un elemento *lista*? [1]

Algún comentario justificando su respuesta:

¿Propondría usted algún cambio o mejora?

2.2 ¿Cuál es su opinión respecto a la interfaz que sirve para manipular un elemento *tabla*? [ ]

Algún comentario justificando su respuesta:
no lo use

¿Propondría usted algún cambio o mejora?

2.3 ¿Cuál es su opinión respecto a la interfaz que sirve para manipular un elemento *imagen*? [ ]

Algún comentario justificando su respuesta:
no lo use.

¿Propondría usted algún cambio o mejora?

3. Acerca de la distribución de fragmentos

3.1 ¿Cómo califica usted el tiempo que transcurre, después de seleccionar la pestaña *"ver"*, con el fin de obtener la visualización del documento posterior a su edición? [ ]

Algún comentario justificando su respuesta:

¿Propondría usted algún cambio o mejora?

3.2 ¿Cuál es su opinión en relación a que dentro de un fragmento se puedan tener uno o más párrafos, junto con uno o más elementos de edición (*listas, tablas, imágenes, etc.*)? [ ]

Algún comentario justificando su respuesta:

¿Desearía usted poder compartir con algún colega parte de una *lista*, una *tabla*, un *párrafo, etc.*? ¿Propondría usted algún cambio o mejora?

Le agradecemos su tiempo y valiosa colaboración. Todas las respuestas que nos ha brindado nos serán de gran ayuda para mejorar nuestro producto.

Cuestionario de evaluación parcial

Fase de prueba piloto del prototipo ELXI

Nombre: Alejandro Rodriguez Angeles
Extensión: 7735
E-mail: arangele @ imp.mx
Adscripción: Matematicas aplicadas y computación

Instrucciones: Por favor responda a cada una de las preguntas, dentro de los corchetes cuadrados, según la escala de valores siguiente:

1 - Muy Bien
2- Bien
3- Suficiente
4- Mal

Además, para cada pregunta, le rogamos nos aporte algún comentario que apoye su respuesta.

1. Acerca de los elementos de edición y formatos de visualización

1.1 ¿Cómo califica usted el número de formatos, disponibles en ELXI, a los que se pueden exportar los documentos *XML* (los disponibles son: *Word, HTML, PDF*)? [2]

Algún comentario justificando su respuesta:
Son los formatos mas comunes usados en Microsoft, pero existen otros muy usados
¿Propondría usted algún otro formato?
La posibilidad de editar en "Latex" y compilar en linea.

1.2 ¿Cuál es su opinión respecto al número de tipos de fragmento disponibles para un documento *General (ej. título, párrafo, pie de página, etc.)?* [1]

Algún comentario justificando su respuesta:
De los opciones mas usados, por lo que me parece Suficiente.
¿Cree usted que son suficientes? ¿Propondría usted algunos otros ? ¿Cúales?

1.3 ¿Cuál es su opinión respecto al formato de visualización obtenido para un documento *HTML*? [2]

Algún comentario justificando su respuesta:
La presentación es buena, aunque un poco simple.

¿Propondría usted algún cambio o mejora?
Incluir la opción para colores del texto, ademas de tamaños.

1.4 ¿Cuál es su opinión respecto al formato de visualización obtenido para un documento *Word*? [2]

Algún comentario justificando su respuesta:
Es muy buena, ya que presenta el documento en formato basico, para de ahi modificarlo a su formato final.
¿Propondría usted algún cambio o mejora?

1.5 ¿Cuál es su opinión respecto al formato de visualización obtenido para un documento *PDF*? [2]

Algún comentario justificando su respuesta:
Es bueno aunque puede presentar algunos problemas con las tablas, porque si no se selecciona el ancho adecuado desde el principio, la tabla se ve mala. Ver pregunta 2.2.
¿Propondría usted algún cambio o mejora?

2. Acerca de las interfaces de los elementos de edición

2.1 ¿Cuál es su opinión respecto a la interfaz que sirve para manipular un elemento *lista*? [1]

Algún comentario justificando su respuesta:
La manera de introducir los datos es amigable y sencilla
¿Propondría usted algún cambio o mejora?
- Se podrian unificar la tabla y tabla numerada en una sola opción y un boton para seleccionar si se quiere numerada
- Dar la opción para que el contenido aparezca alineado izq- derecha y no solo centrado

2.2 ¿Cuál es su opinión respecto a la interfaz que sirve para manipular un elemento *tabla*? [2]

Algún comentario justificando su respuesta:
- En la palabra renglón el acento aparece como "Ã³"
- Fuera de eso para introducir el contenido de las celdas la interfase es bastante amigable

¿Propondría usted algún cambio o mejora?
- La posibilidad de incluir los bordes de la tabla
- El que no se tenga que definir el ancho al empezar la tabla

2.3 ¿Cuál es su opinión respecto a la interfaz que sirve para manipular un elemento *imagen*? [1]

Algún comentario justificando su respuesta:
Bastante simple y eficaz

¿Propondría usted algún cambio o mejora?
La posibilidad de ver la imagen en el documento al pulsar la opción de ver y no tener que generar el word, pdf o html para verla.

3. Acerca de la distribución de fragmentos

3.1 ¿Cómo califica usted el tiempo que transcurre, después de seleccionar la pestaña *"ver"*, con el fin de obtener la visualización del documento posterior a su edición? [1]

Algún comentario justificando su respuesta:

¿Propondría usted algún cambio o mejora?

3.2 ¿Cuál es su opinión en relación a que dentro de un fragmento se puedan tener uno o más párrafos, junto con uno o más elementos de edición (*listas, tablas, imágenes, etc.*)? [1]

Algún comentario justificando su respuesta:
facilita la edición

¿Desearía usted poder compartir con algún colega parte de una *lista*, una *tabla*, un *párrafo, etc.*?
¿Propondría usted algún cambio o mejora?
Si, sería interesante y conveniente el estar editando un parrafo y dar la opción a otro colaborador para editarlo, sin necesidad de tener que [illegible] para cederle los derechos de edición

Le agradecemos su tiempo y valiosa colaboración. Todas las respuestas que nos ha brindado nos serán de gran ayuda para mejorar nuestro producto.

Cuestionario de evaluación parcial

Fase de prueba piloto del prototipo ELXI

Nombre: Blanca Estela Lara
Extensión: 6211
E-mail: blara@...
Adscripción: Programa de Investigación en Matemáticas Aplicadas y Computación.

Instrucciones: Por favor responda a cada una de las preguntas, dentro de los corchetes cuadrados, según la escala de valores siguiente:

1 - Muy Bien
2- Bien
3- Suficiente
4- Mal

Además, para cada pregunta, le rogamos nos aporte algún comentario que apoye su respuesta.

1. Acerca de los elementos de edición y formatos de visualización

1.1 ¿Cómo califica usted el número de formatos, disponibles en ELXI, a los que se pueden exportar los documentos *XML* (los disponibles son: *Word, HTML, PDF*)? [1]

Algún comentario justificando su respuesta: Ninguno

¿Propondría usted algún otro formato? Ninguno

1.2 ¿Cuál es su opinión respecto al número de tipos de fragmento disponibles para un documento *General (ej. título, párrafo, pie de página, etc.)?* [1]

Algún comentario justificando su respuesta: Ninguno

¿Cree usted que son suficientes? ¿Propondría usted algunos otros ? ¿Cúales? Uno de los subrayados y diferentes tipos de tamaño

1.3 ¿Cuál es su opinión respecto al formato de visualización obtenido para un documento *HTML*? [1]

Algún comentario justificando su respuesta: Ninguno

¿Propondría usted algún cambio o mejora? Ninguno

1.4 **¿Cuál es su opinión respecto al formato de visualización obtenido para un documento *Word*? [|]**

Algún comentario justificando su respuesta: Suficiente el formato de visualización ya que cubre mis necesidades.

¿Propondría usted algún cambio o mejora? No

1.5 **¿Cuál es su opinión respecto al formato de visualización obtenido para un documento *PDF*? [|]**

Algún comentario justificando su respuesta: La visualización es buena, pero aún no se pueden realizar modificaciones.

¿Propondría usted algún cambio o mejora? Que se pudieran realizar modificaciones directamente.

2. Acerca de las interfaces de los elementos de edición

2.1 **¿Cuál es su opinión respecto a la interfaz que sirve para manipular un elemento *lista*? [|]**

Algún comentario justificando su respuesta: Está bien

¿Propondría usted algún cambio o mejora? La distancia que hay entre un elemento y otro es muy grande, que sea más pequeña.

2.2 **¿Cuál es su opinión respecto a la interfaz que sirve para manipular un elemento *tabla*? [|]**

Algún comentario justificando su respuesta: Está bien

¿Propondría usted algún cambio o mejora? Que el tamaño de la tabla lo defina el usuario, y una vez insertada pueda modificarse.

2.3 **¿Cuál es su opinión respecto a la interfaz que sirve para manipular un elemento *imagen*? [|]**

Algún comentario justificando su respuesta: Todo está bien

¿Propondría usted algún cambio o mejora? Ninguna, todo está muy bien

3. Acerca de la distribución de fragmentos

3.1 ¿Cómo califica usted el tiempo que transcurre, después de seleccionar la pestaña *"ver"*, con el fin de obtener la visualización del documento posterior a su edición? [I]

Algún comentario justificando su respuesta: Bueno.

¿Propondría usted algún cambio o mejora? El tiempo invertido que consumió [illegible]

3.2 ¿Cuál es su opinión en relación a que dentro de un fragmento se puedan tener uno o más párrafos, junto con uno o más elementos de edición (*listas, tablas, imágenes, etc.*)? [I]

Algún comentario justificando su respuesta: Es necesario para compartir tablas, listas o imágenes completas.

¿Desearía usted poder compartir con algún colega parte de una *lista*, una *tabla*, un *párrafo, etc.*? Si, [illegible]
¿Propondría usted algún cambio o mejora? Ninguna [illegible] completas.

Le agradecemos su tiempo y valiosa colaboración. Todas las respuestas que nos ha brindado nos serán de gran ayuda para mejorar nuestro producto.

Cuestionario de evaluación parcial

Fase de prueba piloto del prototipo ELXI

Nombre: Angel Serrano
Extensión: 7543
E-mail: vserrano@imp.mx
Adscripción: PIMAYC

Instrucciones: Por favor responda a cada una de las preguntas, dentro de los corchetes cuadrados, según la escala de valores siguiente:

1 - Muy Bien
2- Bien
3- Suficiente
4- Mal

Además, para cada pregunta, le rogamos nos aporte algún comentario que apoye su respuesta.

1. Acerca de los elementos de edición y formatos de visualización

1.1 ¿Cómo califica usted el número de formatos, disponibles en ELXI, a los que se pueden exportar los documentos *XML* (los disponibles son: *Word, HTML, PDF*)? [3]

Algún comentario justificando su respuesta:

¿Propondría usted algún otro formato? ascii, hojas de excel, tablas

1.2 ¿Cuál es su opinión respecto al número de tipos de fragmento disponibles para un documento *General (ej. título, párrafo, pie de página, etc.)?* [2]

Algún comentario justificando su respuesta:

¿Cree usted que son suficientes? ¿Propondría usted algunos otros ? ¿Cúales?

1.3 ¿Cuál es su opinión respecto al formato de visualización obtenido para un documento *HTML*? [1]

Algún comentario justificando su respuesta:

¿Propondría usted algún cambio o mejora?

Cuestionario de evaluación parcial

Fase de prueba piloto del prototipo ELXI

Nombre: Angel Soriano
Extensión: 7543
E-mail: vsoriano@imp.mx
Adscripción: PIMAyC

Instrucciones: Por favor responda a cada una de las preguntas, dentro de los corchetes cuadrados, según la escala de valores siguiente:

1 - Muy Bien
2- Bien
3- Suficiente
4- Mal

Además, para cada pregunta, le rogamos nos aporte algún comentario que apoye su respuesta.

1. Acerca de los elementos de edición y formatos de visualización

1.1 ¿Cómo califica usted el número de formatos, disponibles en ELXI, a los que se pueden exportar los documentos *XML* (los disponibles son: *Word, HTML, PDF*)? [3]

Algún comentario justificando su respuesta:

¿Propondría usted algún otro formato? ascii, hojas de excel, tablas

1.2 ¿Cuál es su opinión respecto al número de tipos de fragmento disponibles para un documento *General (ej. título, párrafo, pie de página, etc.)?* [2]

Algún comentario justificando su respuesta:

¿Cree usted que son suficientes? ¿Propondría usted algunos otros ? ¿Cúales?

1.3 ¿Cuál es su opinión respecto al formato de visualización obtenido para un documento *HTML*? [1]

Algún comentario justificando su respuesta:

¿Propondría usted algún cambio o mejora?

3. Acerca de la distribución de fragmentos

3.1 ¿Cómo califica usted el tiempo que transcurre, después de seleccionar la pestaña *"ver"*, con el fin de obtener la visualización del documento posterior a su edición? [l]

Algún comentario justificando su respuesta: Ninguno

¿Propondría usted algún cambio o mejora? El tiempo [illegible]

3.2 ¿Cuál es su opinión en relación a que dentro de un fragmento se puedan tener uno o más párrafos, junto con uno o más elementos de edición (*listas, tablas, imágenes, etc.*)? [l]

Algún comentario justificando su respuesta: Es interesante para un documento [illegible] listas e imágenes completas.

¿Desearía usted poder compartir con algún colega parte de una *lista*, una *tabla*, un *párrafo*, *etc.*? Sí
¿Propondría usted algún cambio o mejora? Ninguna [illegible]

Le agradecemos su tiempo y valiosa colaboración. Todas las respuestas que nos ha brindado nos serán de gran ayuda para mejorar nuestro producto.

Cuestionario de evaluación parcial

Fase de prueba piloto del prototipo ELXI

Nombre:
Extensión:
E-mail:
Adscripción:

Instrucciones: Por favor responda a cada una de las preguntas, dentro de los corchetes cuadrados, según la escala de valores siguiente:

1 - Muy Bien
2- Bien
3- Suficiente
4- Mal

Además, para cada pregunta, le rogamos nos aporte algún comentario que apoye su respuesta.

1. Acerca de los elementos de edición y formatos de visualización

1.1 ¿Cómo califica usted el número de formatos, disponibles en ELXI, a los que se pueden exportar los documentos *XML* (los disponibles son; *Word, HTML, PDF*)? [ ]

Algún comentario justificando su respuesta: Adecuado 6

Word da un puente al mundo MS.

¿Propondría usted algún otro formato? ~~NO~~ Latex; opción para que el documento se compile como LATEX automaticamente.

1.2 ¿Cuál es su opinión respecto al número de tipos de fragmento disponibles para un documento *General (ej. título, párrafo, pie de página, etc.)?* [ ]

Algún comentario justificando su respuesta:

Adecuados

¿Cree usted que son suficientes? ¿Propondría usted algunos otros ? ¿Cúales?

1.3 ¿Cuál es su opinión respecto al formato de visualización obtenido para un documento *HTML*? [ ]

Algún comentario justificando su respuesta: Adecuado

cambiar VER por un icono mas informativo

¿Propondría usted algún cambio o mejora? ~~NO~~ Posibilidad de Ver en ~~de nuevo~~ HTML la parte del documento que escribo.

1.4 ¿Cuál es su opinión respecto al formato de visualización obtenido para un documento *Word*? [ ]
Adecuado
Algún comentario justificando su respuesta:

¿Propondría usted algún cambio o mejora?
NO

1.5 ¿Cuál es su opinión respecto al formato de visualización obtenido para un documento *PDF*? [ ]
Adecuado
Algún comentario justificando su respuesta:

¿Propondría usted algún cambio o mejora?
NO

La opcion crear docume
debiria aparecer en el
parte "mas superior" del menu.

2. Acerca de las interfaces de los elementos de edición

2.1 ¿Cuál es su opinión respecto a la interfaz que sirve para manipular un elemento *lista*? [ ]
Adecuado
Algún comentario justificando su respuesta:

¿Propondría usted algún cambio o mejora?
NO

2.2 ¿Cuál es su opinión respecto a la interfaz que sirve para manipular un elemento *tabla*? [ ]
Adecuado
Algún comentario justificando su respuesta:

¿Propondría usted algún cambio o mejora?
NO

2.3 ¿Cuál es su opinión respecto a la interfaz que sirve para manipular un elemento *imagen*? [ ]
Incomodo
Algún comentario justificando su respuesta:

¿Propondría usted algún cambio o mejora?

debería de aparecer un
arbol-directorio para seleccionar
el archivo.

opción para mostrar los segmentos de manera

3. Acerca de la distribución de fragmentos

3.1 ¿Cómo califica usted el tiempo que transcurre, después de seleccionar la pestaña *"ver"*, con el fin de obtener la visualización del documento posterior a su edición? [ ]

Algún comentario justificando su respuesta: razonable

¿Propondría usted algún cambio o mejora?

3.2 ¿Cuál es su opinión en relación a que dentro de un fragmento se puedan tener uno o más párrafos, junto con uno o más elementos de edición (*listas, tablas, imágenes, etc.*)? [ ]

Algún comentario justificando su respuesta:

¿Desearía usted poder compartir con algún colega parte de una *lista*, una *tabla*, un *párrafo, etc.*?
¿Propondría usted algún cambio o mejora?

Opción para definir el tiempo de actualización del documento en el servidor.

Le agradecemos su tiempo y valiosa colaboración. Todas las respuestas que nos ha brindado nos serán de gran ayuda para mejorar nuestro producto.

Cuestionario de evaluación parcial

Fase de prueba piloto del prototipo ELXI

Nombre: Mauricio Molina
Extensión:
E-mail:
Adscripción:

Instrucciones: Por favor responda a cada una de las preguntas, dentro de los corchetes cuadrados, según la escala de valores siguiente:

1 - Muy Bien
2- Bien
3- Suficiente
4- Mal

Además, para cada pregunta, le rogamos nos aporte algún comentario que apoye su respuesta.

1. Acerca de los elementos de edición y formatos de visualización

1.1 ¿Cómo califica usted el número de formatos, disponibles en ELXI, a los que se pueden exportar los documentos *XML* (los disponibles son: *Word, HTML, PDF*)? [1]

Algún comentario justificando su respuesta:
son los formatos de mayor uso

¿Propondría usted algún otro formato?
no

1.2 ¿Cuál es su opinión respecto al número de tipos de fragmento disponibles para un documento *General (ej. título, párrafo, pie de página, etc.)?* [1]

Algún comentario justificando su respuesta: son 17. Deberían tener nombres descriptivos subDos, Encabezado está mal, BddItálica está b

¿Cree usted que son suficientes? ¿Propondría usted algunos otros ? ¿Cúales?
Sí

1.3 ¿Cuál es su opinión respecto al formato de visualización obtenido para un documento *HTML*? [1]

Algún comentario justificando su respuesta:

¿Propondría usted algún cambio o mejora?

1.4 ¿Cuál es su opinión respecto al formato de visualización obtenido para un documento *Word*? [1]

Algún comentario justificando su respuesta:

¿Propondría usted algún cambio o mejora?

1.5 ¿Cuál es su opinión respecto al formato de visualización obtenido para un documento *PDF*? [1]

Algún comentario justificando su respuesta:

¿Propondría usted algún cambio o mejora?

se debería de poder establecer la carpeta donde se generan los PDF, RTF y HTML

2. Acerca de las interfaces de los elementos de edición

2.1 ¿Cuál es su opinión respecto a la interfaz que sirve para manipular un elemento *lista*? [2]

Algún comentario justificando su respuesta:

es poco intuitiva, sobre todo al insertar una lista x 1ª vez

la operación insertar-lista, pregunta por un popup el número de elementos, debería de poner inmediatamente el botón de la lista, en lugar de abrir directamente el editor de listas.

¿Propondría usted algún cambio o mejora?

tiene un bug: si se presiona "borrar lista" en una lista nueva, se queda en la interfaz el botón de la lista.

2.2 ¿Cuál es su opinión respecto a la interfaz que sirve para manipular un elemento *tabla*? [1]

Algún comentario justificando su respuesta:

correcto

¿Propondría usted algún cambio o mejora?

2.3 ¿Cuál es su opinión respecto a la interfaz que sirve para manipular un elemento *imagen*? [1]

Algún comentario justificando su respuesta:

es correcto

¿Propondría usted algún cambio o mejora?

3. Acerca de la distribución de fragmentos

3.1 ¿Cómo califica usted el tiempo que transcurre, después de seleccionar la pestaña *"ver"*, con el fin de obtener la visualización del documento posterior a su edición? [1] (1)

Algún comentario justificando su respuesta: adecuado

¿Propondría usted algún cambio o mejora?

3.2 ¿Cuál es su opinión en relación a que dentro de un fragmento se puedan tener uno o más párrafos, junto con uno o más elementos de edición (*listas, tablas, imágenes, etc.*)? [1]

Algún comentario justificando su respuesta: el usuario decide el tamaño de los fragmentos, y su cantidad, dependiendo del trabajo colaborativo.

¿Desearía usted poder compartir con algún colega parte de una *lista*, una *tabla*, un *párrafo, etc.*? ¿Propondría usted algún cambio o mejora?

Le agradecemos su tiempo y valiosa colaboración. Todas las respuestas que nos ha brindado nos serán de gran ayuda para mejorar nuestro producto.

Cuestionario de evaluación parcial

Fase de prueba piloto del prototipo ELXI

Nombre: Jorge Alvarez Mena
Extensión: 7239
E-mail: jamena@imp.mx
Adscripción: Ciencias Básicas

Instrucciones: Por favor responda a cada una de las preguntas, dentro de los corchetes cuadrados, según la escala de valores siguiente:

1 - Muy Bien
2- Bien
3- Suficiente
4- Mal

Además, para cada pregunta, le rogamos nos aporte algún **comentario** que apoye su respuesta.

1. Acerca de los elementos de edición y formatos de visualización

1.1 ¿Cómo califica usted el número de formatos, disponibles en ELXI, a los que se pueden exportar los documentos *XML* (los disponibles son: *Word, HTML, PDF*)? [2]

Algún comentario justificando su respuesta: Uso principalmente código para Latex.

¿Propondría usted algún otro formato?

1.2 ¿Cuál es su opinión respecto al número de tipos de fragmento disponibles para un documento *General (ej. título, párrafo, pie de página, etc.)?* [1]

Algún comentario justificando su respuesta:

¿Cree usted que son suficientes? ¿Propondría usted algunos otros ? ¿Cúales?

1.3 ¿Cuál es su opinión respecto al formato de visualización obtenido para un documento *HTML*? [1]

Algún comentario justificando su respuesta:

¿Propondría usted algún cambio o mejora?

1.4 ¿Cuál es su opinión respecto al formato de visualización obtenido para un documento *Word*? [4]

Algún comentario justificando su respuesta:

¿Propondría usted algún cambio o mejora?

1.5 ¿Cuál es su opinión respecto al formato de visualización obtenido para un documento *PDF*? [4]

Algún comentario justificando su respuesta:

¿Propondría usted algún cambio o mejora?

2. Acerca de las interfaces de los elementos de edición

2.1 ¿Cuál es su opinión respecto a la interfaz que sirve para manipular un elemento *lista*? [2]

Algún comentario justificando su respuesta:

¿Propondría usted algún cambio o mejora?

2.2 ¿Cuál es su opinión respecto a la interfaz que sirve para manipular un elemento *tabla*? [2]

Algún comentario justificando su respuesta:

¿Propondría usted algún cambio o mejora?

2.3 ¿Cuál es su opinión respecto a la interfaz que sirve para manipular un elemento *imagen*? [ ]

Algún comentario justificando su respuesta: No lo he usado.

¿Propondría usted algún cambio o mejora?

3. Acerca de la distribución de fragmentos

3.1 ¿Cómo califica usted el tiempo que transcurre, después de seleccionar la pestaña *"ver"*, con el fin de obtener la visualización del documento posterior a su edición? [2]

Algún comentario justificando su respuesta:

¿Propondría usted algún cambio o mejora?

3.2 ¿Cuál es su opinión en relación a que dentro de un fragmento se puedan tener uno o más párrafos, junto con uno o más elementos de edición (*listas, tablas, imágenes, etc.*)? [4]

Algún comentario justificando su respuesta: Esto da flexibilidad a la edición de documentos.

¿Desearía usted poder compartir con algún colega parte de una *lista*, una *tabla*, un *párrafo, etc.*? ¿Propondría usted algún cambio o mejora?

Le agradecemos su tiempo y valiosa colaboración. Todas las respuestas que nos ha brindado nos serán de gran ayuda para mejorar nuestro producto.

Cuestionario de evaluación parcial

Fase de prueba piloto del prototipo ELXI

Nombre: JOSE ARTURO ROZANO GUZMAN
Extensión: 6340
E-mail: arozano@imp.mx
Adscripción: GERENCIA DE DESARROLLO INSTITUCIONAL.

Instrucciones: Por favor responda a cada una de las preguntas, dentro de los corchetes cuadrados, según la escala de valores siguiente:

1 - Muy Bien
2- Bien
3- Suficiente
4- Mal

Además, para cada pregunta, le rogamos nos aporte algún comentario que apoye su respuesta.

1. Acerca de los elementos de edición y formatos de visualización

1.1 ¿Cómo califica usted el número de formatos, disponibles en ELXI, a los que se pueden exportar los documentos *XML* (los disponibles son: *Word, HTML, PDF*)? [1]

Algún comentario justificando su respuesta:
SON LOS FORMATOS MAS UTILIZADOS PARA EL MANEJO DE TEXTOS.

¿Propondría usted algún otro formato?
NO

1.2 ¿Cuál es su opinión respecto al número de tipos de fragmento disponibles para un documento *General (ej. título, párrafo, pie de página, etc.)?* [1]

Algún comentario justificando su respuesta:
ME PARECEN SUFICIENTES

¿Cree usted que son suficientes? ¿Propondría usted algunos otros ? ¿Cúales?
NO

1.3 ¿Cuál es su opinión respecto al formato de visualización obtenido para un documento *HTML*? [1]

Algún comentario justificando su respuesta:
NO FUE UTILIZADO

¿Propondría usted algún cambio o mejora?

1.4 ¿Cuál es su opinión respecto al formato de visualización obtenido para un documento *Word*? [x]

Algún comentario justificando su respuesta:

¿Propondría usted algún cambio o mejora?
DE SER POSIBLE QUE SE PUDIERA FORMATEAR EL DOCUMENTO O DOCUMENTOS PARA HACER MAS AGIL LA PRESENTACION.

1.5 ¿Cuál es su opinión respecto al formato de visualización obtenido para un documento *PDF*? [x]

Algún comentario justificando su respuesta:
NO UTILIZADO.

¿Propondría usted algún cambio o mejora?

2. Acerca de las interfaces de los elementos de edición

2.1 ¿Cuál es su opinión respecto a la interfaz que sirve para manipular un elemento *lista*? [x]

Algún comentario justificando su respuesta:
NINGUNO

¿Propondría usted algún cambio o mejora?

2.2 ¿Cuál es su opinión respecto a la interfaz que sirve para manipular un elemento *tabla*? [x]

Algún comentario justificando su respuesta:
ME PARECE ADECUADO.

¿Propondría usted algún cambio o mejora?

2.3 ¿Cuál es su opinión respecto a la interfaz que sirve para manipular un elemento *imagen*? [x]

Algún comentario justificando su respuesta:
NINGUNO

¿Propondría usted algún cambio o mejora?

3. Acerca de la distribución de fragmentos

3.1 ¿Cómo califica usted el tiempo que transcurre, después de seleccionar la pestaña "*ver*", con el fin de obtener la visualización del documento posterior a su edición? [*1*]

Algún comentario justificando su respuesta:
NINGUNO

¿Propondría usted algún cambio o mejora?

3.2 ¿Cuál es su opinión en relación a que dentro de un fragmento se puedan tener uno o más párrafos, junto con uno o más elementos de edición (*listas, tablas, imágenes, etc.*)? [*1*]

Algún comentario justificando su respuesta:
ES UN BUEN MANEJO DE VARIOS TIPOS DE INFORMACION.

¿Desearía usted poder compartir con algún colega parte de una *lista*, una *tabla*, un *párrafo, etc.*?
¿Propondría usted algún cambio o mejora?
NINGUNO

Le agradecemos su tiempo y valiosa colaboración. Todas las respuestas que nos ha brindado nos serán de gran ayuda para mejorar nuestro producto.

Cuestionario de evaluación parcial

Fase de prueba piloto del prototipo ELXI

Nombre: Eunice Campitán García
Extensión: 7243
E-mail: cu2820@yahoo.com
Adscripción: PYMAC

Instrucciones: Por favor responda a cada una de las preguntas, dentro de los corchetes cuadrados, según la escala de valores siguiente:

1 - Muy Bien
2- Bien
3- Suficiente
4- Mal

Además, para cada pregunta, le rogamos nos aporte algún comentario que apoye su respuesta.

1. Acerca de los elementos de edición y formatos de visualización

1.1 ¿Cómo califica usted el número de formatos, disponibles en ELXI, a los que se pueden exportar los documentos *XML* (los disponibles son: *Word, HTML, PDF*)? [2]

Algún comentario justificando su respuesta:

¿Propondría usted algún otro formato?

Los que vienen en Word

1.2 ¿Cuál es su opinión respecto al número de tipos de fragmento disponibles para un documento *General (ej. titulo, párrafo, pie de página, etc.)?* [3]

Algún comentario justificando su respuesta:

¿Cree usted que son suficientes? ¿Propondría usted algunos otros ? ¿Cúales?

Los que vienen Word.

1.3 ¿Cuál es su opinión respecto al formato de visualización obtenido para un documento *HTML*? [2]

Algún comentario justificando su respuesta:

¿Propondría usted algún cambio o mejora?

1.4 ¿Cuál es su opinión respecto al formato de visualización obtenido para un documento *Word*? [2]

Algún comentario justificando su respuesta:

¿Propondría usted algún cambio o mejora?

1.5 ¿Cuál es su opinión respecto al formato de visualización obtenido para un documento *PDF*? [2]

Algún comentario justificando su respuesta:

¿Propondría usted algún cambio o mejora?

2. Acerca de las interfaces de los elementos de edición

2.1 ¿Cuál es su opinión respecto a la interfaz que sirve para manipular un elemento *lista*? [2]

Algún comentario justificando su respuesta:

Creo que se deberían de agregar viñetas para que el usuario escoja.

¿Propondría usted algún cambio o mejora?

2.2 ¿Cuál es su opinión respecto a la interfaz que sirve para manipular un elemento *tabla*? [2]

Algún comentario justificando su respuesta:

¿Propondría usted algún cambio o mejora?

2.3 ¿Cuál es su opinión respecto a la interfaz que sirve para manipular un elemento *imagen*? [2]

Algún comentario justificando su respuesta:

Sería deseable que el usuario pudiese modificar el tamaño de la imagen

¿Propondría usted algún cambio o mejora?

3. Acerca de la distribución de fragmentos

3.1 ¿Cómo califica usted el tiempo que transcurre, después de seleccionar la pestaña *"ver"*, con el fin de obtener la visualización del documento posterior a su edición? [a]

Algún comentario justificando su respuesta:

No me desespere.

¿Propondría usted algún cambio o mejora?

3.2 ¿Cuál es su opinión en relación a que dentro de un fragmento se puedan tener uno o más párrafos, junto con uno o más elementos de edición (*listas, tablas, imágenes, etc.*)? [b]

Algún comentario justificando su respuesta:

Es algo deseable

¿Desearía usted poder compartir con algún colega parte de una *lista*, una *tabla*, un *párrafo, etc.*? ¿Propondría usted algún cambio o mejora?

Le agradecemos su tiempo y valiosa colaboración. Todas las respuestas que nos ha brindado nos serán de gran ayuda para mejorar nuestro producto.

Nota: Sería deseable que el programa tuviese una Ayuda e integrado "What's this" para acelerar el aprendizaje del programa.

También ~~una~~ desarrollar una interface WYSIWYG