UNIVERSIDAD NACIONAL AUTÓNOMA
DE MÉXICO

FACULTAD DE CIENCIAS

DESARROLLO DE UN EDITOR PARA DOCUMENTOS XML

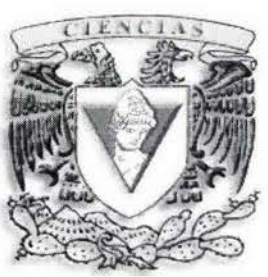
T E S I S

QUE PARA OBTENER EL TÍTULO DE
LICENCIADO EN CIENCIAS DE LA
COMPUTACIÓN

P R E S E N T A :

Ruiz Salinas Oscar

DIRECTORA DE TESIS:
Dra. Amparo López Gaona



2004

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

DRA. MARÍA DE LOURDES ESTEVA PERALTA
Jefa de la División de Estudios Profesionales de la
Facultad de Ciencias
Presente

Comunicamos a usted que hemos revisado el trabajo escrito:

"Desarrollo de un editor para documentos XML"

realizado por Ruiz Salinas Oscar con número de cuenta 9421099-2

quién cubrió los créditos de la carrera de Ciencias de la Computación

Dicho trabajo cuenta con nuestro voto aprobatorio.

Atentamente

Director de Tesis Propietario	Dra. Amparo López Gaona	
Propietario	Mat. Salvador López Mendoza	
Propietario	M. en C. María Guadalupe Elena Ibargüengoitia González	
Suplente	Mat. Víctor Hugo Dorantes González	
Suplente	M. en I.O. María de Luz Gasca Soto	

Consejo Departamental de Matemáticas

Mat. María Concepción Ana Luisa Solís González Cosío

"A mis padres y hermanos . . . "

Agradecimientos

Gracias a mis padres Raymundo Ruiz y Romana Salinas por brindarme toda su confianza y apoyo para terminar mis estudios. Agradezco también a mis hermanos Silvestre, Eduardo, Daniel y a todos amigos por el apoyo y amistad que me brindaron.

Gracias a mi asesora Dra. López Gaona Amparo, por toda la confianza y ayuda que me proporcionó durante todo el desarrollo de este trabajo y a todos mis profesores por todos los conocimientos y experiencias que me proporcionaron a lo largo de la carrera, en especial: *Pablo Barrera, Jaqueline Cañetas, María Luz Gasca, Mary Glazman, Guadalupe Ibargüengoitia, Francisco Hernández, Miguel Lara, Salvador López, Benjamín Macías, Gustavo Marquez, Hanna Oktaba, Angélica Orozco, Concepción Ruiz, Ana Luisa Solis y Eliza Viso.*

Introducción General.

Actualmente existe gran cantidad de herramientas, que son empleadas tanto en la industria como en el ámbito académico, para generar diversos tipos de documentos los cuales son conocidos como *documentos electrónicos.*

Generalmente la mayoría de estas aplicaciones emplean formatos muy rígidos para representar documentos y varían mucho entre una aplicación y otra. Esto ha generado graves problemas de incompatibilidad dificultando el compartir, manejar y presentar información, por lo que surge la necesidad de contar con una redefinición de los documentos electrónicos para solucionar estos problemas.

XML (Extensible Markup Languaje) es un metalenguaje que permite representar información con un formato estándar, se caracteríza por separar el contenido, la representación y la estructura de los documentos, lo que hace posible manejar a cada uno de ellos de forma más efectiva. Esto ha permitido comenzar a solucionar los problemas y limitaciones definidas en los formatos anteriores.

Los documentos XML pueden ser generados con cualquier editor de texto disponible, sin embargo, la funcionalidad que ofrecen estos editores a personas con poca experiencia en el manejo de este tipo de documentos es mínima. Los usuarios tienen una mayor carga de trabajo debido a que tienen que conocer las reglas sintácticas y semánticas de los documentos para poder crearlos.

Este trabajo surgió con el objetivo de desarrollar un editor que facilite la creación, edición y revisión de documentos XML. Para lograr el objetivo planteado, este trabajo se estructura de la siguiente manera:

- En el primer capítulo de este documento se presentan aspectos generales de las herramientas edición, cubriendo temas como: aspectos históricos, características, consideraciones de implementación y evaluación, así como la tendencia actual de estas herramientas.
- En el capítulo 2 se presenta un panorama general sobre lo que es XML cubriendo temas como: aspectos históricos, características, ventajas, desventajas, sintaxis, así como la generación y revisión de los documentos XML.
- En el capítulo 3 se presenta el desarrollo de *Un editor para documentos XML*, usando la metodología definida en el *Proceso unificado de desarrollo de software.*

Finalmente se presentan las conclusiones obtenidas y la bibliografía empleada a lo largo de este proyecto.

Índice General

Índice de Figuras

Capítulo 1

Aspectos generales de los editores

1.1 Introducción.

Manejar información y representarla ha sido siempre muy demandado por las personas, con el paso del tiempo se han desarrollado técnicas y herramientas para el procesamiento de información, cada una de estas herramientas cuenta con su propio ambiente de trabajo y tiene como objetivo facilitar la creación, edición y almacenamiento de documentos.

En este capítulo se presenta un panorama general sobre el desarrollo de herramientas para la edición de documentos, que van desde editores en línea hasta herramientas visuales, finalmente se presenta una descripción de la tendencia y futuro de estas herramientas.

1.2 Aspectos Históricos.

El procesamiento de la información puede localizarse en la historia de las computadoras en sí, al comenzar a desarrollarse los primeros sistemas operativos, también comenzaron a desarrollarse un conjunto de herramientas y utilerías para el procesamiento de información.

Los primeros editores surgen como una herramienta para ayudar a los programadores e investigadores a producir reportes, manuales y otros documentos asociados con el desarrollo de sus investigaciones, desafortunadamente fueron desarrollados para ser usados por "profesionales" y no por gente común, por lo que los primeros editores resultaron difíciles de manejar.

Las primeras herramientas que se desarrollan eran muy limitadas en cuanto a presentación y a funcionalidad se refiere, sólo permitían generar o modificar documentos de texto y cualquier proceso como revisión, compilación, etcétera, eran independientes a los editores.

Peter D. Smith[1], define una clasificación de los editores que han surgido a lo largo de la historia. En la Figura 1 puede observarse los diferentes tipos de editores que definen esta clasificación.

1.2.1 Editores de línea.

Los primeros editores desarrollados fueron los *editores de línea*, este modelo permite crear y modificar documentos de texto desde terminales. Contaban con varias limitaciones encuanto a funcionalidad y representación de líneas o *unidades de información*. Ejemplos de este tipo de

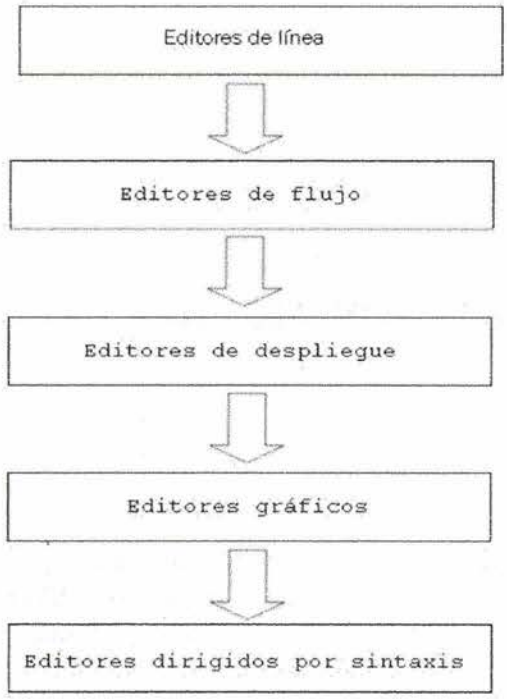


Figura 1.1: Tipos de editores.

editores son *ATS* y *VIP* [2], los cuales contaban con un lenguaje de comandos que permitía a los usuarios hacer modificaciones dentro de una línea o un grupo de líneas contiguas.

En los primeros editores de línea, la longitud máxima para la representación de una línea era de 80 caracteres y algunas funcionalidades, como por ejemplo *recorrer el documento*, no eran soportadas. Así por ejemplo si al realizar alguna operación y la cadena excedía en más de 80 caracteres entonces los caracteres que estaban al final de la línea eran eliminados. Con el paso del tiempo estas limitaciones comenzaron a ser eliminadas, al comenzar a manejarse el concepto de *salto de línea.*

Con la creación de los *Editores de línea manejados por contexto*, los usuarios podían editar sus documentos en base a patrones para que el editor identificará una sección de texto y aplicara alguna operación, lo cual era mejor que propocionar el número de línea de forma explícita. Ejemplos de los primeros editores de este tipo fueron *Editor trend-sefting* el cual fue desarrollado en una máquina *IBM 7090* y *CMS*[2] desarrollado por IBM.

Hasta este punto de la historia de la edición, los usuarios estaban obligados a pensar acerca de las *entidades multilínea* como son: parrafos, bloque de programas, etcétera, como un grupo de líneas integrales, debido a que no se permitía el manejo de comandos interlineados (aplicar una instrucción que inicie en un línea y termine en otra).

Uno de los primeros editores que elimina la restricción sobre la longitud de línea fue *QuickEditor*[2] con lo cual surgen los *Editores de línea de lóngitud variable,* en donde el principal elemento de operación fue la línea de longitud arbitraria. Surge así el concepto de

Superlínea el cual establece limitaciones a 500 caracteres, lo cual era un espacio mucho mayor en comparación a la línea de 80 caracteres. Con el surgimiento de este nuevo concepto, las operaciones de presentación en pantalla fueron redefinidas.

Al eliminar la restricción sobre la longitud de la línea, generó un fuerte impacto para el procesamiento de texto. Otro de los beneficios obtenidos fue que el texto desplegado no necesariamente tiene una relación uno a uno con la representación interna del documento.

Con los *Editores de superlínea* se elimina el problema de truncación de caracteres y se permite el realizar operaciones sobre líneas traslapadas. La búsqueda de patrones en líneas cruzadas aún no eran soportadas en su totalidad por este tipo de editores.

Ejemplo de un editor de línea:

- **ed**: Editor de línea de UNIX.

  Este editor se caracteriza porque emplea un buen reconocedor de expresiones regulares, para aplicar instrucciones a un rango de líneas del documento.

  Ejemplo: el siguiente comando ejecuta al editor, pidiendo trabajar con el archivo *animales.*

  $ ed animales

  10,40s/cat/dog/g

  El usuario al escribir la instrucción anterior solicita remplazar todas las ocurrencias de *cat* por *dog* en el rango de líneas 10-40. La g indica que se realice de forma global en cada línea, si se omite entonces solo se aplica esta instrucción a la primer ocurrencia de cada línea.

Debido a la estructura de este modelo, resultaba difícil conocer y combinar instrucciones para editar documentos, los usuarios tenían que ver todo el archivo para ver las líneas a modificar, así como memorizar y proporcionar explícitamente los comandos para obtener el resultado de la edición.

1.2.2 Editores de flujo.

Un editor de flujo toma el contenido de un documento de texto, lo edita y lo transforma en un flujo de datos que puede ser direccionado a un archivo o ser la entrada de otro programa. Similar a los editores de línea soporta scripts con operaciones a realizar y no requiere de terminales específicas.

Con el desarrollo de los editores de flujo *Stream editor* se desarrolla una nueva forma de contemplar a los documentos. Este tipo de editores se caracterizan por considerar al documento completo como una sola unidad o cadena que fue rota en sublíneas para su despliegue, logrando así mejorar el desarrollo las operaciones de búsqueda.

HES y *FRESS*[2] son ejemplos de este tipo de editores. Tiempo después se desarrolla *TECO*[2], un editor basado en los anteriores, soporta caracteres de *carro de retorno* dentro del stream, lo que permite que pudiera ser editado como cualquier caracter.

La generalidad del modelo *TECO* hizo posible la construcción de diversos editores sofisticados usando a esta herramienta como núcleo.

Ejemplo de un editor de flujo:

- **sed**: Stream EDitor

  Es una versión no interactiva de *ed*, emplea filtros que toman una entrada y producen una salida. Los comandos de *sed* son similares a los de *ed*, pero con una diferencia principal, *sed* toma como entrada un archivo, lo edita, pero no modifica el contenido del archivo original, sino que deja el resultado en la salida estándar.

  Debido a que es un editor de flujo todos los comandos a ejecutar son proporcionados desde el principio. No tiene restricciones sobre el tamaño de los archivos, todos sus comandos son compilados antes de iniciar la ejecución, una vez compilados se extrae una línea del documento, sobre la cual se aplican todos los comandos que le correspondan y se presenta a la salida estándar y se repite el proceso para el resto de las líneas.

  Ejemplo:

  $sed 's/son/eran/' poema

  Remplaza la palabra *son* por *eran* en el archivo *poema* y el resultado lo presenta en la salida estándar.

Los *Editores de flujo* permiten eliminar las limitaciones de *línea* y de *superlínea* empleando el concepo de *despliegue de multilíneas* (lo que más a delante se conocería como *áreas de despliegue*, lo cual dió pie a la siguiente generación de tipos de editores.

1.2.3 Editores de despliegue

Los editores de despliegue o *Full-screen Display*, se caracterizan porque pueden trabajar con líneas de longitud variable o como streams, ofreciendo al usuario un presentación entera del texto para poder realizar cambios los cuales son reflejados inmediatamente sin que el usuario emplee algún comando adicional. Uno de los primeros editores de despliegue fue *TVEDIT*[2] desarrollado en la Universidad de Stanford.

Inicialmente los editores de despliegue tenían un área de trabajo de 24 líneas, debido a que las terminales donde se mostraba el documento estaban limitadas y usualmente sólo una porción de texto del documento era presentada y cuando se llegaba a la parte inferior el texto se recorría automáticamente.

En este tipo de editores los comandos eran representados por una serie de caracteres que podían ser proporcionados como la entrada de texto normal. El cursor indicaba la posición actual y los usuarios podían desplazarlo al texto que se deseaban editar. Así los caracteres podían ser remplazados escribiendo sobre ellos, podían ser borrados poniendo el cursos encima de ellos y presionando la tecla de borrado, o podían ser insertados a partir de la posición actual del cursor.

El usuario podía escribir en cualquier parte del área de edición y no habían limitaciones en cuanto al número de líneas del documento. Con el paso del tiempo se desarrollan extensiones a este modelo permitiendo manejar múltiples áreas de trabajo.

Ejemplo de un editor de despliegue:

- **vi (Visual editor)**

  Es un editor altamente configurable que permite crear, editar y almacenar documentos de texto. Representa una herramienta muy poderosa, generalmente es empleado para desarrollar programas, pero muchos principiantes lo consideran como una herramienta compleja debido a que sus instrucciones no son sencillas.

  Tiene definido una línea de comandos y un área de trabajo, se caracteriza por soportar múltiples buffers y cuenta con ayuda en línea.

  Ejemplo:

  $ **vi** poemas.txt

  i

  Al proporcionar la primera instrucción se indica que se va a editar el archivo *poemas.txt*, mientras que con la segunda instrucción se indica al editor que va a insertar texto después de la posición actual del cursor.

1.2.4 Editores gráficos.

En 1959 *Douglas Engelbart* de la universidad de *Stanford* introduce una nueva y mejor forma de pensar acerca de los editores, desarrolla el sistema *NLS oNLine System*[2] el cual proporciona un ambiente de desarrollo aprovechando el poder de las terminales de despliegue.

NLS permite soportar vistas multicontexto con lo cual se genera el concepto de *interfaz de usuario*, otra de las principales contribuciones del proyecto *NLS* fue el dispositivo ratón o *mouse* como dispositivo de entrada y que ahora es un producto comercial. *NLS* fue el primer editor que provee soporte para manipular documentos en términos de su estructura y no sólo con respecto a su contenido.

Tiempo después con el surgimiento de nuevos sistemas de ventanas y al comenzar a desarrollarse el concepto de *altas resoluciones*, el proceso de edición de documentos comienza a tener una mayor importancia y permite extender aun más la funcionalidad de los editores.

Bravo editor y *Smalltalk enviroment* [2] fueron los primeros editores que demuestran el expresivo poder de las altas resoluciones. Estas herramientas ya no sólo soportaban su tarea sino además permiten presentar una mayor cantidad de información en una misma área de trabajo, emplean una mayor cantidad de colores, distintos tipos de fuentes, estilos y formatos(texto, imágenes,etcétera).

Poco a poco muchas funciones comunes como son: renombrar archivos, búsquedas, revisión, compilar, etcétera, comenzaron a ser integradas a los editores logrando así ser ambientes de desarrollo más efectivos y adaptados a las necesidades de los usuarios.

Ejemplo de un editor gráfico.

Forte for Java.(Ver Anexo C *Herramientas*).

1.2.5 Editores dirigidos por sintaxis.

Un editor dirigido por sintaxis garantiza que la estructura de un documento se realiza en base a un conjunto de reglas de sintaxis, desde que comienza a ser editado y durante todo el proceso de edición. La principal aplicación de los editores dirigidos por sintaxis es generar y presentar documentos en un lenguaje en específico.

Los editores dirigidos por sintaxis son una extensión lógica de los editores gráficos, se caracterizan porque permiten editar un tipo específico de documentos, al igual que los editores anteriores cuenta con un conjunto de comandos para editar el documento y provee de operaciones para manipular la estructura del documento.

Uno de los primeros editores que exitiende el concepto de los editores estructurados es *EMILY*[2], logrando así desarrollar el concepto de *Editores dirigidos por sintáxis* en el cual la estructura impuesta en un programa comienza a ser editado en base a la estructura del lenguaje de programación en sí. Los usuarios tenían la habilidad de poder manipular construcciones lógicas como son *do-while*.

Ejemplo de un editor dirigido por sintaxis:

Cornell: Cornell Program Synthesizer.

Este editor reconoce la sintaxis de PL, C y Pascal. Permite desarrollar documentos de cualquiera de estos lenguajes usando:

- Templates: son programas que construyen expresiones con nodos terminales y no terminales.
- Placeholders: son marcas en un template que describen una situación aceptable para un nodo no terminal.
- Phrases: Son nodos terminales, representan piezas de texto que el usuario puede sustituir por un *placeholder*.

El usuario ingresa el programa de arriba hacia abajo, seleccionando en un menú las opciones a remplazar en el texto actual. Por eficiencia algunos de los identificadores son símbolos terminales en la gramática requerida por el editor para habilitar y distinguir entre una expresión válida o inválida.

Las operaciones sobre el buffer están un poco restringidas en comparación con los editores de despliegue, por ejemplo se tienen restricciones al borrar parte de la estructura, hay secciones no editables en los *templates*, etcétera.

1.3 Presentación y Revisión de documentos.

Editar documentos y asignar un formato son las partes más importantes del procesamiento de edición, facilitan la presentación de ideas y conceptos.

Con el surgimiento de los primeros editores, comienzan a realizarse estudios para el reconocimiento de cadenas y algoritmos de corrección, comenzando así a desarrollar programas que permitieran realizar la *revisión* (dado un archivo de texto identificar que palabras son correctas) y *corrección* (para cada una de las palabras en el archivo de texto determinar si está en la base de conocimientos y si no está proporcionar la más parecida) de documentos y

formateadores.

A partir de esto se comenzaron a generar, adaptar y modificar los programas para generar distintas presentaciones sobre los documentos. En los 70's el procesamiento de texto comenzó a tener una mayor importancia comenzando a ser programas independientes y con el paso del tiempo estos sistemas van teniendo una mayor capacidad de procesamiento.

La edición de texto y el procesamiento de texto con el desarrollo del sistema *UNIX* el primer ambiente de propósito general en el cual las utilerias tienen mucho mayor peso como herramientas de programación, comenzando así a popularizarse.

1.3.1 Presentación.

Los primeros editores desarrollados sólo generaban texto y para poder asignar un formato de presentación se empleaban otras utilerías, estos programas eran usados para procesar los documentos de texto, en los cuales se insertaban marcas para asignar un formato, lo que permitía: centrar el texto del documento, asignar paginación, etcétera.

Las principales herramientas que permitían realizar esta tarea fueron:

- nroff

  Cuenta con un conjunto de marcas que permiten definir e identificar cada parte del documento y define el formato de presentación de la información. Para emplear esta herramienta se insertan las etiquetas en un documento de texto, después de emplea esta herramienta para procesarlo y presentarlo basado en el formato *WYSIWYG* "imprime lo usted ve en pantalla"[7].

  Ejemplo: la siguiente instrucción indica que la palabra *Título* será presentada con un tamaño de fuente H1.

  .H1 *Título*

- troff

  Esta herramienta extiende las funcionalidades de *nroff*, permite emplear diferentes tipos de fuentes para la presentación del documento. Una de las principales desventajas que presenta es que dificulta hacer cambios al texto formateado.

Con el surgimiento del sistema de ventanas, la presentación, contenido y estructura de los documentos se fusionan en una sola unidad, esto generó que el concepto *WYSIWYG*, adquiriera una mayor importancia y se siguiera usado hasta estas fechas.

1.3.2 Revisión.

Uno de los primeros programas que permitió realizar la revisión de documentos fue *Spell* desarrollado por Ralph G. en 1971 para una DEC-10, el cual comenzó a ser distribuido y adaptado a los sistemas operativos que fueron surgiendo con el tiempo.

En esta primer versión, la revisión se podía realizar por:

- **Token List**
  - Se tiene definida una lista de palabras válidas y un hash de palabras vacío.

- Para cada palabra del documento si no está registrada en el hash entonces se inserta, si ya está registrada en el hash entonces su valor se incrementa en 1.
- Una vez que se termina de procesar todas las palabras del documento se ordena el hash y cada palabra del hash se compara con la lista de palabras válidas.

Problemas que presenta: el texto es sensitivo, lista de palabras válidas estaba limitada.

- **Diccionario de palabras**
  - Se tiene definida una lista de palabras correctas(diccionario).
  - La aplicación construye un árbol con cada token del archivo.
  - Se revisa el árbol, identificando que cada nodo esté definido en el diccionario. Muestra los nodos que no estén bien definidos. Este método actualmente se emplea en UNIX.

Así poco a poco durante la evolución de las computadoras, diversas compañías comienzan a crear y mejorar las técnicas y métodos de revisión, corrección y edición documentos. Logrando así adaptar más funcionalidades a los editores para poder aplicar nuevas operaciones, por ejemplo:

- Minimizar los errores en los documentos.

  Al momento de detectar errores, generalmente eran presentados en una terminal, lo que limitaba presentar el documento y los errores de forma simultánea. Con el paso del tiempo se permitió que dentro del mismo editor se pudieran mostrar ambos y en algunos casos enfatizando la posición del error.

- Lenguaje natural.

  Si durante la edición o revisión de un documento se producía un error, el usuario tenía que conocer el código de error para encontrarlo en la documentación. Las mejoras que se comienzan a desarrollar es presentar los mensajes de error en lenguaje natural.

- La documentación debe ser parte del producto.

  Comienza a definirse la ayuda en línea, esto permite que se pueda acceder a la información más a detalle, así mismo esto permitía al usuario leer e imprimir sólo lo que necesitaba.

Esto generó que los editores comenzaran a ser una herramienta básica de todos los futuros sistemas de procesamiento de documentos, actualmente la mayoría de estas características son empleadas en la mayoría de los editores.

1.4 Herramientas de edición.

Hoy en día existe una gran cantidad de aplicaciones, conocidas como *editores*, que permiten generar documentos de forma rápida y fácil. Cada una de estas herramientas cuenta con su propio ambiente de trabajo y con el paso del tiempo han comenzado a ser dirigidas a tareas específicas de los usuarios y a la creación de tipos específicos de documentos, logrando así que sean más fáciles de usar, aprender y reducen los tiempos de elaboración de documentos.

1.4.1 ¿Qué es un editor?

Un editor es un programa que permite:

- Generar o acceder directamente al contenido de un documento.
- Modificar el contenido del documento.
- Almacenar los cambios realizados.

Cada editor cuenta con un área de edición formada por un buffer en el cual se almacena una copia del documento a procesar, todas las modificaciones se realizan sobre el buffer hasta que el usuario o el propio sistema solicita guardar los cambios, esto permite:

- Descartar los cambios realizados en el buffer y mantener el archivo original intacto.
- Guardar múltiples versiones del documento en archivos separados.

Un editor para guardar los cambios realizados, toma el contenido del buffer y lo almacena en un archivo. Los documentos que pueden ser editados con estas herramientas son conocidos como *documentos electrónicos*, cada uno de ellos puede ser almacenado en formato binario y en algunos casos como una secuencia de caracteres. El formato binario puede tener asociadas múltiples representaciones como son: sonido, imágenes, texto, etcétera. Esto permite que cada editor pueda soportar múltiples formatos para la edición y almacenamiento de su información.

1.4.2 Objetivo.

Un editor tiene como principal objetivo ser una herramienta efectiva para la creación, edición y almacenamiento de documentos.

1.4.3 Funcionalidad.

La funcionalidad de un editor está definida por el conjunto de características y operaciones que emplea para generar documentos, algunas de estas capacidades son comunes a la mayoría de los editores pero otras no. Cada herramienta entre mayor sea la funcionalidad que proporcione, representará una herramienta muy poderosa.

Un editor cuenta con las siguientes características:

- Ambiente de trabajo.
  - Interfaz
  - Una o más áreas de trabajo.
  - Conjunto de operaciones de edición.

Además, puede proporcionar algunas de las siguientes operaciones:

- Acceso a documentos

  - Crear
  - Abrir

- Edición de documentos

  - Borrar, buscar, copiar, cortar, insertar, pegar, remplazar, recorrer el documento, seleccionar, etcétera.

- Almacenamiento de documento

  - Puede soportar uno o más formatos distintos.
  - Permite renombrar.
  - Permite guardar el documento completo o sólo alguna parte en específico.

- Interactuar con otras herramientas

  - Interpretar.
  - Ejecutar programas.
  - Revisar documentos.
  - Detección y corrección de errores.
  - Presentar información.

Poco a poco han comenzando a integrarse nuevas características y funcionalidades a los editores, esto ha permitido que los editores sean más sofisticados durante el desarrollo de documentos.

1.4.4 Consideraciones de implementación.

Desarrollar un editor no es una tarea fácil, hay que tomar en cuenta diversos aspectos antes de su desarrollo, principalmente hay que considerar los siguientes puntos:

- Características de la información a manejar.
- Estrategias sobre cómo usar y acceder a la información.
- Definir cómo se va a representar el texto a ser editado.
- Aspectos de implementación:

  - Recuperar errores del usuario y del sistema.
  - Actualizar la pantalla.
  - Búsqueda de cadenas.

1.5 Futuro de las herramientas de edición.

Hoy en día los sistemas visuales de edición son los más usados debido a que están estructurados para aprovechar al máximo las capacidades de las personas, emplean colores, texto, sonido, imágenes, animaciones, etcétera, para poder describir y manipular los documentos. La mayoría de estas herramientas están asociadas a algún contexto específico y están dirigidas a los usuarios.

La tendencia actual en el desarrollo de herramientas para la edición de documentos, se enfoca principalmente en:

- Generar nuevas herramientas que soporten la edición de documentos de las tecnologías que van surgiendo con el paso del tiempo.
- Extender la funcionalidad de los editores que existen para soportar la edición de documentos de las nuevas tecnologías.

Capítulo 2

XML

2.1 Introducción.

XML es el acrónimo de *eXtensible Markup Languaje*, es un metalenguaje que permite representar información con un formato universal. Tiene definida una gramática, un conjunto de restricciones y reglas sintácticas para poder crear documentos, los cuales son conocidos como *documentos XML.*

Cada uno de estos documentos está formado de unidades de almacenamiento llamadas entidades, las cuales contienen la información y definen la estructura de los documentos.

Una aplicación para que pueda acceder al contenido de un documento XML, es necesario que cuente con un módulo de software llamado parser XML, actualmente existe una gran cantidad de herramientas como son: *Simple API for XML* (SAX), *Document Object Model* (DOM) entre otras, que permiten realizar esta tarea.

En este capítulo se presenta un panorama general sobre lo que es XML, aspectos históricos, objetivos, características, componentes que constituyen a los documentos XML, así como las reglas de sintaxis para construir documentos y verificar que sean correctos.

2.2 Aspectos Históricos.

Representar información siempre ha jugado un papel muy importante en las diversas áreas de investigación, cada una de ellas con el paso del tiempo creó su propia notación y forma de representar su información, por lo que el compartir, manejar y presentar información se convirtió en un problema muy serio que ha perdurado hasta nuestros días.

Esto generó como consecuencia que los formatos que se empleaban para representar la información varien mucho, entre diversas aplicaciones y en algunas ocasiones entre versiones de una misma aplicación.

Otro problema que surge es que generalmente durante el desarrollo de documentos se emplea la técnica 'cortar-copiar' para poder generar distintas presentaciones de un mismo documento, lo que genera que actualizar estos documentos se convierta en una tarea cada vez más compleja y tediosa.

Para solucionar estos problemas comenzó a desarrollarse un formato estándar de representación de información conocido como SGML.

2.2.1 SGML.

Standar Generalized Markup Language(SGML) es un metalenguaje para la definición de la estructura y contenido de diferentes tipos de documentos. Fue desarrollado en 1969, participaron investigadores de distintas instituciones como son: IBM, ANSI, Departamento de Defensa y otros. En 1986, esta herramienta comienza a ser un estándar (ISO 8879:1986) para la representación de la información.

SGML se caracteriza por separar la representación de la información de su contenido, el cual se define a través de un conjunto de etiquetas que son definidas por los mismos usuarios.

En SGML pueden definirse las reglas necesarias de las etiquetas que pueden ser empleadas para generar los documentos. Estas reglas generalmente son definidas en un documento, el cuál será descrito más adelante en la Sección 2.8, conocido como *Documento de Definición de Tipos (DTD)*. Esto permite que un usuario pueda definir un conjunto de etiquetas a emplear para representar su información y facilita que un mismo documento pueda tener asociadas múltiples presentaciones.

Actualmente se siguen desarrollando trabajos de investigación sobre SGML, como son la evaluación de cambios, nuevos proyectos y administración de documentación.

SGML representa una herramienta muy poderosa para representar información, pero debido al tamaño y a la complejidad de la versión completa de SGML, las herramientas desarrolladas para SGML tienden también a ser relativamente extensas, a raíz de esto comienzan a desarrollarse versiones más simplificadas y fáciles de usar.

La Figura 2.1 muestra las tecnologías que surgieron a partir del desarrollo de SGML.

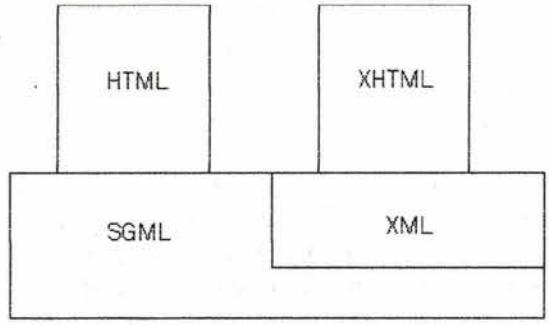


Figura 2.1: SGML.

2.2.2 HTML.

Como uno de los primeros intentos por simplificar SGML, en 1991 surge *Hypertext Markup Language (HTML)*[16] el cual es un lenguaje que está basado en las reglas y sintaxis definidas por SGML.

Para lograr este objetivo se define un conjunto de etiquetas con las cuales se define la estructura de un documento, pero con el gran crecimiento de Internet, intereses comerciales originaron que este lenguaje haya evolucionado muy rápido.

Este lenguaje permite representar documentos, publicar y acceder a información a través de la red, pero su estructura no indica que es lo que se esta representando, se preocupa principalmente en el formato o características del documento a presentar (colores, fuentes, etcétera).

Aunque HTML se convirtió en un lenguaje de marcas de gran popularidad, representa un lenguaje limitado para representar información, debido a su estructura rígida y limitada.

2.2.3 XHTML.

Extensible HyperText Markup Language(XHTML) es un lenguaje que define una familia de documentos que están basados en XML.

Características:

- Permite que nuevas etiquetas o elementos puedan ser incluidos sin necesidad de modificar todo el documento DTD.
- Los documentos que define están basados en la norma XML.
- Los documentos que define pueden ser presentados como tex/html o como text/xml, con las correspondientes hojas de estilo.

2.2.4 XML.

En 1996 comienza a definirse una nueva versión simplificada de SGML, la cual se enfoca más en cómo representar el contenido de la información y no tanto en su presentación.

En Noviembre de ese mismo año el consorcio *World Wide Web* (W3C) comienza a anunciar las primeras propuestas de *eXtensible Markup Language (XML)* este consorcio estaba formado por un grupo de compañías entre las que destacan: Microsoft, Netscape, IBM, Sun, Oracle y otras.

En esta versión se define un conjunto de etiquetas o marcas extendible y sus respectivos atributos o propiedades para representar la información, esto permitió solucionar el problema y limitaciones definidas en los lenguajes anteriores.

Con el paso del tiempo se fueron desarrollando nuevas versiones, y se genera una versión completa del estándar de XML, este documento esta formado principalmente de 3 componentes:

- XML-Lang : la definición del lenguaje
- XML-Link : define un conjunto de convenciones para enlazar documentos XML con otros recursos web.
- XSL *(Style Sheet Languaje)*: define hojas de presentación y estilo.

El desarrollo exitoso de XML, ha tenido grandes efectos en la mayoría de las aplicaciones comenzando así a emplear al máximo las capacidades que ofrece esta herramienta.

2.3 Impacto de XML

XML inició el próximo ciclo de la revolución de la información, ha tenido un gran impacto en la forma tradicional de pensar sobre los documentos, tradicionalmente elementos como: contenido, organización y despliegue estaban definidos en una sola unidad. La Figura 2.2 muestra la relación de los elementos de un documento tradicional y un documento no tradicional.

La estructura de un documento tradicional se empleaba desde las épocas antiguas en los códices y papiros, actualmente se maneja este concepto como *WYSIWYG (What you see is what you get)* y en la mayoría de los casos es oculta la forma de almacenamiento.[15]

Hoy en día se ha trabajado más en estos aspectos, logrando que el almacenamiento tenga una distinción entre: contenido, organización y presentación.

Figura 2.2: Diagrama tradicional de un documento.

Al separar cada uno de estos conceptos, es posible manejar cada uno de ellos de forma más efectiva y con el surgimiento de XML se logra realizar la separación de cada uno de estos componentes.[15]

Comenzar a separar estos elementos permitió:

- Independencia entre la representación y la presentación.
- Desempeño más eficiente.
- Manipulación más efectiva.
- Diversas presentaciones para un mismo conjunto de datos.
- Determinar que información se desea presentar.

XML está basado en esta filosofía, para fomentar en este mundo el intercambio de información. La presentación puede realizarse de varias formas a través de hojas de estilo: *Extensible Stylesheet Languaje(XSL)* y *Cascading Stylesheets(CSS).*

Actualmente muchas compañías han comenzado a anunciar aplicaciones basadas en XML debido a que el costo de emplear XML es mínimo porque el estándar de XML es libre y las herramientas para el manejo de XML también son libres, por lo que XML ha comenzado a ser soportado por más aplicaciones.[17]

2.4 ¿Qué es XML?

XML es el acrónimo de *eXtensible Markup Languaje*, es un metalenguaje que permite representar información con un formato universal. Representa la información a través de marcas o etiquetas, las cuales definen el contenido y la estructura de un documento, lo que permite tener un mayor control de la información y facilita la manipulación del documento.

XML representa un subconjunto de *SGML*, aunque no contiene toda la funcionalidad de esta herramienta cuenta con la mayoría de los beneficios proporcionados por *SGML*. XML se caracteriza por permitir desarrollar documentos que se puedan describir por si mismos y que sean válidos. Así mismo ofrece una combinación única de flexibilidad, simplicidad y legibilidad de documentos tanto para las personas como para las computadoras. [18]

2.5 Objetivos de XML.

Los principales objetivos de XML[15] son:

- XML fuera directamente utilizado en Internet.
- Sea soportado por una gran variedad de aplicaciones.
- Sea compatible con SGML.
- Sea sencillo escribir programas que procesarán XML.
- Los documentos sean legibles para las personas.
- Diseñar un documento XML sea fácil.
- Los documentos XML sean fáciles de crear.

2.6 Características de XML.

- Permite al usuario definir sus propios lenguajes de etiquetas.
- Maneja la independencia entre el contenido y la presentación del documento.
- Permite un alto grado de reutilización de documentos.
- Permite enviar información SGML sobre la WEB.
- Facilita la creación y procesamiento de documentos para generar nuevos documentos.
- Permite el empleo de otros tipos de aplicaciones sobre la WEB.

2.7 Documentos XML.

Un documento XML, es simplemente un documento de texto donde algunas cadenas de caracteres representan el contenido en sí del documento y otras representan la estructura a través de etiquetas que serán interpretadas por el procesador de XML.

Los documentos XML se basan en el principio de la independencia de los datos, es decir separa los datos de su presentación lo que permite tener un mejor control sobre la estructura de los documentos que se generan.

La sintaxis que emplean los documentos XML es similar a la de los archivos HTML, pero tiene las siguientes características:

- Etiquetas propias

  En los documentos XML no se emplean etiquetas predefinidas como en el caso de HTML, esto va a permitir poder definir nuestro propio lenguaje de etiquetas.

- Sintaxis estricta

  Hay diferencia entre el empleo de letras mayúsculas y letras minúsculas, restricciones para los nombres de las etiquetas y la declaración de los atributos.

- Empleo de un *Documento de Definición de Tipos (DTD).*

  Los documentos DTD se definen más adelante en la sección 2.8. Este tipo de documentos no es necesario emplearlos pero es muy recomendable debido a que permiten definir la estructura de los documentos y es muy útil en la elaboración y validación de los documentos XML.

Un documento XML tiene definido dos tipos de estructuras:

- **Física**

  El documento está compuesto por unidades llamadas entidades, cada una de ellas posee información que es flexible y representa una unidad de información, la cual a su vez puede estar formada de otras unidades. Esto permite definir niveles de abstracción, el nivel más alto es conocido como nivel raíz.

  Cada entidad está compuesta de 2 componentes:

- Contenido: representa la información a almacenar.
- Marcas: proveen información acerca del contenido.
  * Provee de un método para describir de forma jerárquica la información.
  * Proveen de una sintaxis simple y estándar para que el parser pueda interpretar y usar la información que describe.
  * Sus características son descritas mediante atributos.

- **Lógica**

  La estructura lógica describe el contenido de la información. En esta estructura el documento está compuesto de declaraciones, elementos, comentarios, referencias a caracteres e instrucciones de procesamiento.

2.7.1 Componentes de un documento XML

Un documento XML esta constituído de tres secciones genéricas: prólogo, opcionalmente la declaración de un tipo de documento *(Document Type Declaration)* y un elemento raíz. Cada una de estas secciones se divide en estructuras más detalladas. La Figura 2.3 [15] muestra los componentes que constituyen un documento XML.

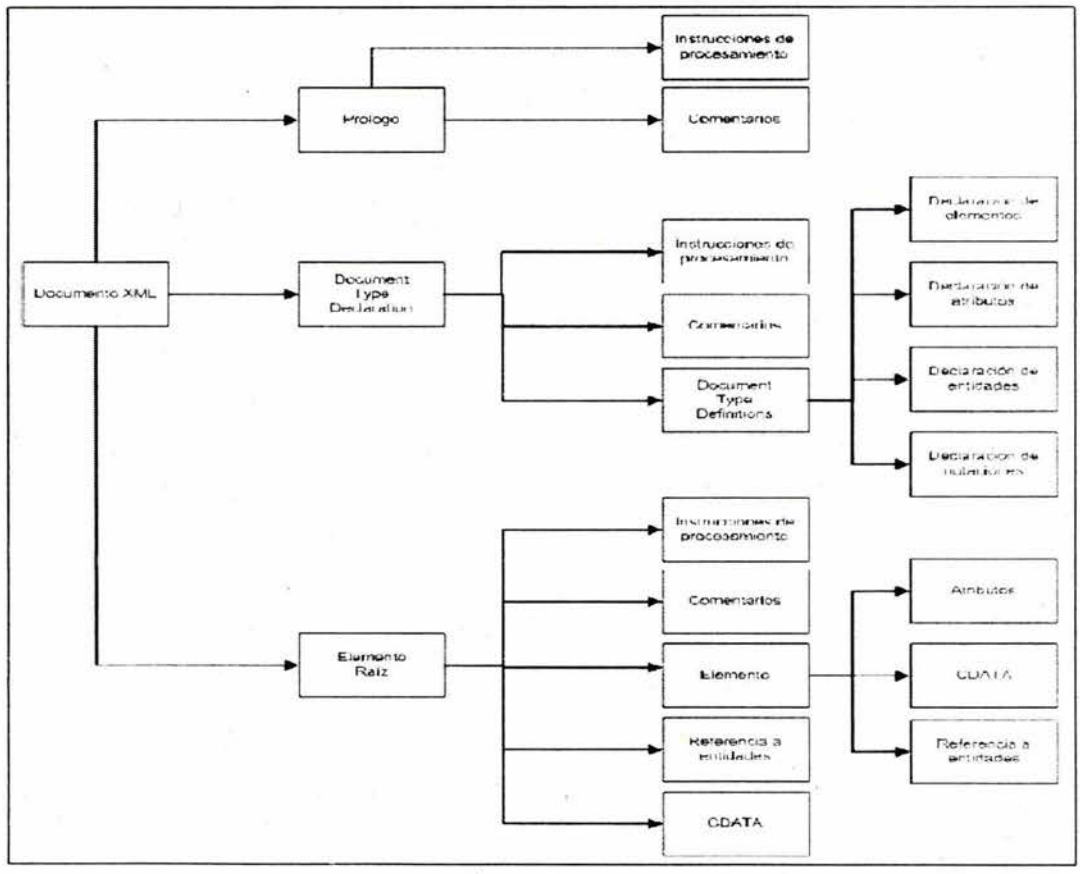


Figura 2.3: Componentes de un documento XML.

- **Prólogo:**

  Forma parte de una declaración XML, esta sección esta definida por comentarios e instrucciones de procesamiento.

- **Comentarios:**

  Nos permiten documentar a los documentos que generamos, su sintaxis es exactamente similar a la empleada en HTML.

  Sintaxis.

```
<!-- Comentario -->
```

- **Instrucciones de procesamiento.**

  Las instrucciones de procesamiento(IP) se emplean para proporcionar información más a detalle sobre el procesamiento y contenido del documento. Pueden ser procesadas por distintos parser.

  Las instrucciones de procesamiento nos permiten definir:

  - Procesador de la instrucción(*xml, rtf u otro*).
  - Información sobre la versión de XML que estamos utilizando(*versión*).
  - Información sobre el tipo de codificación que se emplea dentro del documento: UTF-8, ASCII, UNICODE u otro (*encoding*).
  - Indica si el documento XML tiene asociado un documento de definición de tipos (*standalone*)

    standalone="no" : indica que el documento XML tiene asociado un documento DTD externo, el cual deberá ser leído para poder determinar las propiedades con las cuales se puede construir un documento XML.

    standalone="yes" : indica que no hace referencia a recursos externos.

  Sintaxis.

```
<?procesor specific_instruction="value"?>
```

  Ejemplos:

```
<?xml version="1.0" ?>
<?xml version="1.0" encoding="UTF-8"?>
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
```

- **Declaración de un tipo de documento.**

  La declaración de un tipo de documento (*Document Type Declaration*) es un enunciado que tiene como objetivo indicar la existencia y la localización de un documento de definición de tipos (*Document Type Definition DTD*), es decir este enunciado indica al parser si tiene que emplear un DTD para realizar la revisión y validación del documento XML.

  La diferencia entre un documento de declaración de tipos y un documento de definición de tipos, es que el primero es un enunciado que hace referencia al segundo mientras que el segundo es un conjunto de reglas que permiten definir la estructura de un documento XML.

  Sintaxis.

  La declaración de un tipo de documento permite asociar a un documento DTD con un documento XML de las siguientes formas:

  (a) Interna.

  La declaración del tipo de documento se realiza directamente en el documento XML. se indica cual es el nombre del elemento raíz del documento y las reglas del DTD.

```
<!DOCTYPE elemento_raiz [
<!-- Definicion de un DTD interno -->
]>
```

  (b) Externa:

  Se indica cuál es el nombre del elemento raíz del documento XML y se hace referencia al DTD el cual esta definido en un archivo externo.

```
<!DOCTYPE elemento_raiz SYSTEM "archivo" >
```

  (c) Combinación.

  No es muy recomendable emplear este tipo de declaración debido a que permitiría redefinir componentes del DTD externo y podrían obtener resultados no deseados.

```
<!DOCTYPE elemento_raiz SYSTEM "archivo.dtd" [
<!ATTLIST elemento nombreAtributo  CDATA #REQUIRED>
]>
```

  Ejemplos.

  Supongamos que se tiene definido un documento XML llamado *materias.xml* el cual tiene como elemento raíz CatalogoGrupos y hace referencia a un documento DTD, la forma de asociar estos documentos es:

a) Declaración externa:
Suponiendo que el archivo donde esta almacenado el documento DTD se llama *materias.dtd.*

```
<?xml  version="1.0" encoding="UTF-8"?>
<!DOCTYPE CatalogoGrupo SYSTEM "materias.dtd">
<CatalogoGrupos>
  <materia>
    <nombre> Algebra Superior I </nombre>
    <horario > 9:00 a 10:00 </horario>
  </materia>
</CatalogoGrupos>
```

(b) Declaración interna.

```
<?xml  version="1.0" encoding="UTF-8"?>
<!DOCTYPE CatalogoGrupo [
<!ELEMENT CatalogoGrupos(materia) >
<!ELEMENT materia (nombre, horario) >
<!ELEMENT nombre (#PCDATA) >
<!ELEMENT horario (#PCDATA) >
]>

<CatalogoGrupos>
  <materia>
    <nombre> Algebra Superior I </nombre>
    <horario > 9:00 a 10:00 </horario>
  </materia>
</CatalogoGrupos>
```

- **Elementos:**

Los elementos son unidades básicas que contienen la información del documento y permiten al usuario definir la estructura del documento XML que se desea generar, los símbolos que indican que se trata de una elemento son: <, >, /

Cada elemento puede ser:

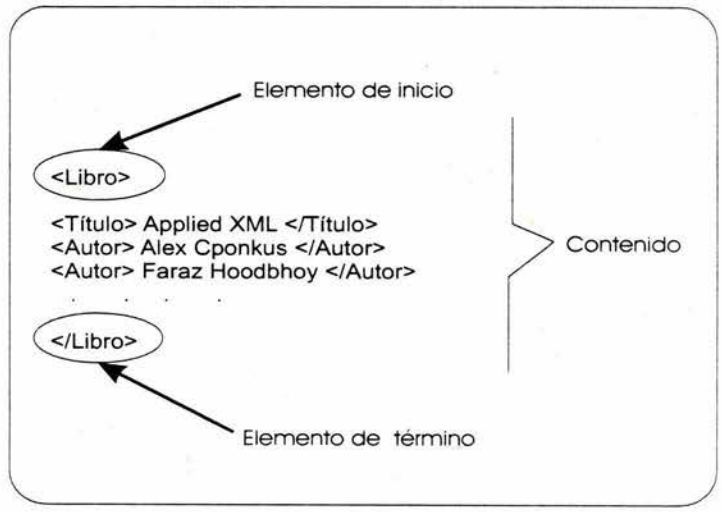
- **Elemento con contenido:**
  Este tipo de elementos están limitados por una etiqueta de inicio y una etiqueta de fin, se caracterizan por que pueden contener a su vez uno o más componentes.

  Sintaxis.
  Cada etiqueta de inicio consiste de una palabra limitada por < y >, mientras que la etiqueta de término esta formada por la palabra de la etiqueta de inicio limitada ahora por < / y >. La única restricción que se tiene es que la palabra no inicie con xml|XML|[0-9].

```
<nombre>  .....  contenido  ....  </nombre>
```

Hay que recordar que XML es sensitivo, por lo que el nombre de la etiqueta de inicio debe coincidir con el de la etiqueta de término.

Ejemplo: en la siguiente figura se muestra cómo los nombres del elemento de inicio y término coinciden, lo mismo ocurre con cada uno de los elementos que definen el contenido del elemento *Libro*.



- **Elemento vacío:**
  Este tipo de elementos se caracterizan porque no puede contener a otros componentes.

  Sintaxis.
  Los elementos vacíos estan definidos por una etiqueta, la cual consiste de una palabra limitada por < y / >, la cual no debe iniciar con xml|XML|[0-9].

```
<nombre/>
```

  Ejemplo:

```
<editorial  nombre="Wiley" />
```

Cada elemento pueden tener asociado un conjunto de *atributos* para proporcionar una mayor información. Son declarados dentro de los elementos de inicio.

Sintaxis.

La sintaxis para declara un atributo es:

```
nombre = "valor"
```

Ejemplo.

```
<materia clave="0132 creditos="100" horario="10:00 a 12:00>
  <nombre> Calculo Diferencial e Integral IV </nombre>
  <profesor> Miguel Lara Aparicio </profesor>
  <salon> P-212 </salon>
<materia>
```

- **CDATA:(Secciones de caracteres de datos.)**

Las secciones de caracteres de datos (*Character Data sections*) CDATA, permiten integrar texto en un documento en XML que no será interpretado, se emplea generalmente cuando se desea incluir caracteres reservados, esto permite que el parser no tenga conflictos.

Sintaxis.

```
<![DATA[
   texto que  no se interpretara
]]>
```

Ejemplo:

```
<ejemplo>
  <![DATA[
      A < B  && B < C entonces A < C
      A >0   && B > 0 entonces A+B > 0
  ]]>
</ejemplo>
```

- **Entidades de referencia.**

Las entidades de referencia son unidades de información que permiten hacer referencia a caracteres, texto y hasta documentos enteros, que de alguna forma podrían generar conflictos al parser.

Sintaxis.

```
&nombre_de_la_entidad;
```

Generalmente son definidas en el documento DTD. También se tiene definido un conjunto de entidades predefinidas:

- &*amp*; para el &
- &*lt*; para el <
- &*gt*; para el >

- &*apos*; para el '
- &*quot*; para el "

Ejemplo:

Para representar que la edad de Juan es menor a la edad de Pedro tendríamos:

```
<?xml version="1.0" encoding="UTF-8"?>
<comparacion_de_edad>
  Juan &lt; Pedro
</comparacion_de_edad>
```

A partir de los componentes vistos en esta sección pueden definirse dos tipos de documentos XML conocidos como:

2.8 Documentos XML Bien Formados.

Un documento XML Bien Formado se refiere a que este documento es sintácticamente correcto y su estructura es correcta. Para poder desarrollar documentos XML bien formados es necesario conocer la sintaxis de XML para tomar en cuenta las siguientes lineamientos:

- Hay exactamente un elemento raíz.
- Las etiquetas de inicio y término:
  - Están bien anidadas:
    Cada marca de inicio tiene asociada una de término y no se traslapa con otras etiquetas.
  - Tienen el mismo nombre.
    XML es sensible a la utilización de mayúsculas y minúsculas.
- Atributos.
  - El valor de los atributos van entre comillas.
  - Ningún atributo puede aparecer más de una vez en la misma etiqueta.

2.9 Documentos XML Válidos.

Los documentos XML válidos se caracterizan por:

- Ser documentos XML Bien Formados.
- Tener definida una estructura de acuerdo a una DTD.

2.10 Documentos DTD

Al desarrollar aplicaciones es necesario intercambiar información en base a una estructura particular, en XML la forma de definir esta estructura es empleando un template de SGML llamado DTD o a través de esquemas.

Los *DTD* juegan un papel muy importante debido a que nos permiten definir de forma explícita y completa la estructura lógica que puede tomar un documento XML a través de un conjunto de declaraciones. Los documentos que están construidos conforme a un DTD, son conocidos como documentos XML válidos.

Un parser emplean el DTD para validar la estructura del documento XML, para lograr esto el parser lee las reglas definidas en el DTD, las interpreta y revisa rigurosamente que la información contenida en el documento XML corresponda a lo definido en el DTD, quien le indica al parser que el documento XML tiene asociado un DTD es el *prólogo*.

2.10.1 Componentes de un documento DTD.

Un documento DTD esta formado básicamente de declaraciones las cuales pueden ser de la siguiente forma:

- **Notaciones:**

  Son declaraciones que indican el tipo de información que esta contenida en el documento.

  Sintaxis.

  ```
  <!NOTATION nombre SYSTEM "identificador">
  ```

  Ejemplo.

  ```
  <!NOTATION gif SYSTEM "image/gif">
  ```

- **Entidades:**

  Representan o definen abreviaciones de elementos que pueden ser empleados dentro del documento XML.

  Sintaxis.

  ```
  <!ENTITY nombre "contenido">
  ```

  Ejemplo.

  ```
  <!ENTITY H "Hidrogeno">
  ```

Y en el documento XML se emplearía como: &H;

- **Elementos:**

Define los elementos pueden ser empleados en el documento XML así, como la estructura que debe tener cada uno de ellos. Una declaración de elementos es un enunciado que provee el nombre que va a tener un elemento, el tipo de información que puede contener y los atributos que tiene asociados.

Sintaxis.

```
<!ELEMENT nombre (contenido)operador>
```

donde:

nombre: representa el nombre de la etiqueta a emplear dentro del documento XML, este nombre tiene como restricción no iniciar con xml|XML|[0-9].

operador: representa el operador más general que puede aplicarse a cada uno de los componentes definidos en el contenido, puede ser instanciado por alguno de los siguientes operadores:

? : indica 0 o 1 ocurrencia.
\+ : indica 1 o más ocurrencias.
* : indica 0 o más ocurrencias.

contenido:
Define que componentes debe contener un elemento, el orden y las ocurrencias en que pueden aparecer dentro del documento XML.

Hay varias formas de declarar el contenido de un elemento, generalmente puede estar formado por los siguientes componentes:

- **#PCDATA**
  Indica que el contenido sólo puede contener caracteres y no va a poder contener ningún otro elemento.

  Sintaxis.

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

  Ejemplo:

  ```
  <nombre> Ruiz Salinas Oscar </nombre>
  ```

- **Hijo único**

  Indica que un elemento solamente va a poder contener a otro elemento.

  En este caso indicamos que el elemento *nombreCompleto* contiene un solo elemento llamado *Apellido Paterno*, el cual contiene solo caracteres.

  Sintaxis.

```
<!ELEMENT nombreCompleto (apellidoPaterno)>
<!ELEMENT apellidoPaterno (#PCDATA)>
```

  Ejemplo:

```
<nombreCompleto>
   <apellidoPaterno> Ruiz  </apellidoPaterno>
</nombreCompleto>
```

- **Lista secuencial.**

  Representa una lista donde indicamos los múltiples hijos que puede tener un elemento, cada uno de estos elementos esta seperado por comas.

  En este tipo de declaración también se define el orden en que deben aparecer en el documento XML.

  Sintaxis.

```
<!ELEMENT nombreCompleto(apellidoPaterno , apellidoMaterno ,nombres)>
<!ELEMENT apellidoPaterno(#PCDATA)>
<!ELEMENT apellidoMaterno(#PCDATA)>
<!ELEMENT nombres(#PCDATA)>
```

  La declaración indica el elemento *nombreCompleto* contiene exactamente tres hijos, los cuales deberán aparecer en el orden *apellidoPaterno, apellidoMaterno y nombres.*

  Ejemplo.

```
<nombreCompleto>
  <apellidoPaterno> Ruiz  <apellidoPaterno>
  <apellidoMaterno > Salinas <apellidoMaterno>
  <nombres>  Oscar </nombres>
</nombreCompleto>
```

- **Lista optativa.**

  Indica que el contenido de un elemento es una lista de elementos de los cuales de los cuales solo puede ser seleccionado uno.

  Sintaxis.

```
<!ELEMENT postre(helado | gelatina)>
<!ELEMENT helado(#PCDATA)>
<!ELEMENT gelatina(#PCDATA)>
```

Ejemplo.

A partir de lo declarado podría obtenerse (a) o (b) pero no ambos.

a)

```
<postre>
    <helado> Vainilla </helado>
</postre>
```

b)

```
<postre>
  <gelatina> Fresa </gelatina>
</postre>
```

- **EMPTY**

  Permite definir elementos vacíos que pueden ser empeados en un documento XML.

  Sintaxis.

```
<!ELEMENT elemento_vacio EMPTY>
```

  Ejemplo.

```
<elemento_vacio/>
```

- **ANY**

  Indica que un elemento puede tener como contenido cualquier combinación de texto y elementos declarados en el DTD actual.

- **Operadores.**

  Los operadores nos permiten definir el número de ocurrencias que un elemento puede tener dentro de un documento XML, esto permite que la estructura de un documento XML pueda variar.

  Ejemplos

  ? Especifica que un elemento puede ocurrir 0 o 1 vez.
  Generalmente se emplea para definir datos que pueden o no ser proporcionados por el usuario.

```
<!ELEMENT usuario (nombre, rfc, mail?, telefono?)>
<!ELEMENT nombre  (#PCDATA)>
<!ELEMENT rfc     (#PCDATA)>
<!ELEMENT mail    (#PCDATA)>
<!ELEMENT telefono (#PCDATA)>
```

El parser identificaría que hay un elemento llamado usuario el cual tiene como contenido los elementos nombre, rfc y opcionalmente podría tener al elemento mail, teléfono o ambos.

+ Indica que un elemento puede contener un subelemento puede ocurrir 1 o más veces.

```
<!ELEMENT usuario (nombre,rfc,mail+,telefono+)>
<!ELEMENT nombre  (#PCDATA)>
<!ELEMENT rfc     (#PCDATA)>
<!ELEMENT mail    (#PCDATA)>
<!ELEMENT telefono (#PCDATA)>
```

Esto permitiría indicar que el elemento usuario tiene como contenido los elementos nombre, rfc y opcionalmente puede contener una o más direcciones de correo, seguidos de uno o más números telefónicos.

* Indica que un elemento puede contener un subelemento que puede ocurrir 0 o más veces.

```
<!ELEMENT usuario (nombre,rfc,mail*,telefono*)>
<!ELEMENT nombre  (#PCDATA)>
<!ELEMENT rfc     (#PCDATA)>
<!ELEMENT mail    (#PCDATA)>
<!ELEMENT telefono (#PCDATA)>
```

Esto permitiría indicar que hay un elemento usuario el cual tiene como contenido a los elementos nombre, rfc y opcionalmente podría o no tener cualquier cantidad de direcciones de correo, seguidos opcionalmente de cualquier cantidad de números telefónicos.

- **Atributos.**

Los atributos tienen como objetivo proporcionar información más detallada de un elemento. Dentro del DTD puede realizarse la definición de una lista de atributos que van a ser empleados para cada elementos declarado en el DTD al generar el documento XML.

Para que un parser pueda garantizar que los atributos empleados en el documento XML es necesario que también estén definidos en el DTD a través de una declaración de lista de atributos la cual nos describe:

- El elemento al que está asociado.
- El nombre del atributo.
- El tipo del atributos.
- El valor de default.

Sintaxis.

```
<!ATTLIST nombre_del_elemento
                    nombre_del_atributo1  tipo  default
                    nombre_del_atributo1  tipo  default
                      . . .
>
```

El tipo de un atributo puede tomar los siguientes valores:

- **CDATA**: indica que contiene cadenas de texto.

```
<!ELEMENT libro  (titulo)>
<!ELEMENT titulo (#PCDATA)>
<!ATTLIST libro
                  autor CDATA default
>
```

Ejemplo.

```
<libro autor="Andrew Tanenbaum">
 <titulo> Redes de Computadoras </titulo>
</libro>
```

- **Enumeration**: indica que el valor del atributo es obtenido de cualquier valor especificado de la lista de valores.

```
<!ELEMENT mail (#PCDATA)>
<!ATTLIST mail
                prioridad(urgente|normal|despacio) default
>
```

Ejemplo:

```
<mail prioridad="urgente">
   ... texto del mail....
</mail>
```

- **Entity y Entities**: ambos atributos permiten insertar entidades en los valores de los atributos. Para poder ser empleados previamente tiene que haber sido declarada la entidad en el DTD.

  Entity indica al parser que el valor del atributo tiene asociado una entidad, mientras que entities indica que puede tener asociada una o mas entidades definidas en el DTD.

```
<!ENTITY imagen mail prioridad="urgente">
   ... texto del mail....
</mail>
```

Ejemplo:

```
<portada src="&imagen;"/>
```

- **ID, IDREF e IDREFS.**

  ID permite definir un identificador único para cada elemento dentro del documento XML, esto permite identificar a cada elemento de manera única.

  IDREF representa un apuntador a un ID previamente declarado, esto permite incluir el contenido de ese elemento en este.

  IDREFS es similar a IDREF solo que ahora permite realizar múltiples referencias a ID's.

```
<!ELEMENT libro (capitulo+)>
<!ELEMENT capitulo (#PCDATA)>
<!ATTLIST capitulo
                    clave ID  default
>
```

Ejemplo:

```
<libro>
  <capitulo  clave=:"A91">
      Desarrollo de aplicaciones WEB
 </capitulo>
</libro>
```

- **NMTOKEN, NMTOKENS**

  NMTOKEN indica que el valor del atributo es una cadena que no contiene espacios, mientras que NMTOKENS indica que son varias cadenas las cuales están separadas por espacios.

```
<!ELEMENT carro EMPTY>
<!ATTLIST carro
                placas NMTOKEN  default
>
```

Ejemplo:

```
<carro placas="HDF658">
```

- **NOTATION**

  Esencialmente especifica una lista predeclarada de notaciones las cuales son definidas en el DTD. Esto va a permitir definir el valor de un atributo a través del valor de una notación.

```
<!NOTATION gif SYSTEM "image/gif">
<!ATTLIST aplicacion
                        tipo NOTATION(gif)
>
```

Ejemplo:

```
<aplicacion  tipo=gif>
```

- **Atributos de default.**

  Representan la ultima parte de la declaración de atributos en un DTD. puede tener asociado los siguientes tipos:

  - **Required:** indican al parser que en el documento XML debe contener un atributo y su respectivo valor para el atributo, si no tiene un valor asociado entonces genera error. Esto permite garantizar que se puede obtener un valor de este atributo.
    Sintaxis:

```
<!ELEMENT carro EMPTY>
<!ATTLIST carro
                licencia  CDATA #REQUIRED
>
```

  - **Implied:** indica que el valor del atributo no esta definido, por lo que la aplicación que desea manipular el documento deberá proporcionar este valor y decidir que hacer. Esto permite que la aplicación tenga una mayor flexibilidad y responsabilidad del manejo de la información.

    Sintaxis:

```
<!ELEMENT persona EMPTY>
<!ATTLIST persona
                  genero CDATA #IMPLIED
>
```

  - **Fixed:** indica que los valores de los atributos son definidos en la declaración del DTD y el valor no puede ser cambiado en el documento XML. Si el valor no es proporcionado entonces el parser asignará uno por default.

    Sintaxis:

```
<!ELEMENT persona EMPTY>
<!ATTLIST persona
                  estado CDATA #FIXED "Feliz"
>
```

- **Supplied:** representa la declaración más común, proporciona el valor de default para un atributo cuando no es proporcionado algún valor en el documento XML.

Sintaxis:

```
<!ELEMENT colorFavorito EMPTY>
<!ATTLIST colorFavorito
                        color(rojo,verde,blanco,azul) "rojo"
>
```

2.11 Revisión de un documento XML.

XML es un herramienta que no esta asociada a algún lenguaje de programación o algún procesador de palabras en específico, para poder manipular los documentos XML, el consorcio W3C desarrollo un conjunto de interfaces: *Simple API for XML*(SAX) y *Document Object Model*(DOM).[15]

Ambas herramientas son un estándar que pueden ser implementadas en diferentes lenguajes de programación, en el caso del lenguaje de programación Java la implementación de estas interfaces es conocida como: *Java API for XML Parsing* (JAXP).

2.11.1 Herramientas

En *Java*, la implementación de la interfaz de DOM y SAX son proporcionadas en el paquete *JAXP*, esta herramienta tiene definido un conjunto de clases que permiten leer documentos XML, realizar la revisión y validación de estos documentos y los hace accesibles a aplicaciones Java que los requieran.

- **SAX**

  El API de SAX provee de un mecanismo para accesar a documentos XML, esta herramienta fue desarrollada por miembros del grupo de desarrollo XML-DEV. Este estándar consiste de un conjunto de interfaces que pueden ser implementadas en diferentes lenguajes de programación.

  El modelo de SAX permite realizar el análisis, revisión y manipulación de documentos XML, se caracteriza por generar un evento al momento de encontrar una marca durante el parseo del documento. La versión original de SAX fue liberada en Mayo 1998, la cual fue superada por SAX2.0 en Mayo del 2000.

  Para emplear esta herramienta en Java todo lo que hay que hacer es implementar algunos métodos de las siguientes interfaces:

  ContentHandler es la interfaz más importante de este conjunto de interfaces, contiene un conjunto de métodos para diferentes pasos durante el parseo del documento XML, pero en muchos casos solo estaremos interesados por algunos de estos métodos.

DefaultHandler es un clase que provee de una implementación vacía de todo el ContentHandler, los principales métodos que nos interesa rescribir, como son:

startElement()
Este método es el que hace el trabajo real, debido a que permite realizar validaciones sobre el nombre de la marca actual, permite accesar a los atributos a través de la clase Attribute para poder accesar al nombre y valor de los atributos.

endElement()
Este método se ejecuta cuando se localiza una marca de término en el documento XML.

La forma de declarar el parser depende de la versión de SAX que se desee emplear por ejemplo:

- SAX 1.0

```
SAXParserFactory spt = SAXParserFactory.newInstance();
SAXParser sp = spf.newSAXParser();
```

SAXParser no puede ser instanciado directamente, tiene que emplearse el método newSAXParser(), esto permite que diferentes implementaciones sean compatibles con los cambios.

- SAX 2.0

```
ParserAdapter pa = new ParserAdapter(sp.getParser())
```

ParserAdapter es complementario al parser de SAX 1.0. pero cuenta con una mayor funcionalidad.

- **DOM**

El API DOM está basado en un modelo completamente diferente al procesamiento del API de SAX, esta herramienta proporciona un conjunto de interfaces para la revisión y manipulación de los documentos XML.

Las principales interfaces que maneja son:

Document Es la clase de más alto nivel, representa un documento XML, esta clase contiene como datos un árbol de nodos.

Node Es un tipo básico que puede ser un elemento, atributo o algún otro tipo de contenido, así mismo provee de las operaciones básicas de un árbol, lo que manipular el documento.

Este modelo se caracteriza por que lee el documento XML una sola vez y construye un árbol asociado al documento, sobre el cual se hace la manipulación y recorrido del documento.

Las principales operaciones que pueden emplearse son:

getDocumentElement() nos permite accesar al elemento raíz del documento.

getElementByTagName(name) regresa una colección de nodos a partir del nodo que cuenta con este nombre.

getAttribute(name) regresa el valor del atributo name.

La forma de crear la instancia del parser de DOM es:

```
DocumentBuilderFactory dbf =
                        DocumentBuilderFactory.newInstance();
DocumentBuilder db = dbf.newDocumentBuilder();
Document doc  = db.parse( fileName);
```

La principal diferencia entre SAX y DOM es que uno lee el documento pieza por pieza. mientras que el otro lee el documento una sola vez.

2.11.2 Proceso de revisión.

El proceso de revisión se inicia cuando al parser se le indica que inicie la revisión de un documento XML. Básicamente lo que se realiza es que el parser lee el documento XML, va identificando cada uno de los componentes del documento y verifica que estén bien definidos. En la Figura 2.4 se presenta el diagrama completo del proceso de revisión de un documento XML.

Las principales tareas que realiza un parser son:

- Revisa rigurosamente que el documento XML sea bien formado.
- Identifica si el documento XML hace referencia o incluye un DTD entonces:
  a) No hace referencia a un DTD.
     * Se inicia el procesamiento futuro del documento como puede ser la manipulación del documento, etcétera.
  b) Hace referencia a un DTD.
     * Determina si el DTD es sintácticamente correcto.
     * Determina si el documento XML tiene una estructura en base a lo definido en el DTD, es decir determina si es válido.
     * Se inicia el procesamiento futuro del documento.

  Como habíamos visto en secciones anteriores emplear un DTD le va a indicar al parser el orden y la información que debe contener cada elemento del documento XML.
- En caso que no se cumpla con alguno de estos lineamientos, el parser muestra al usuario el error que detecto indicando también su localización.

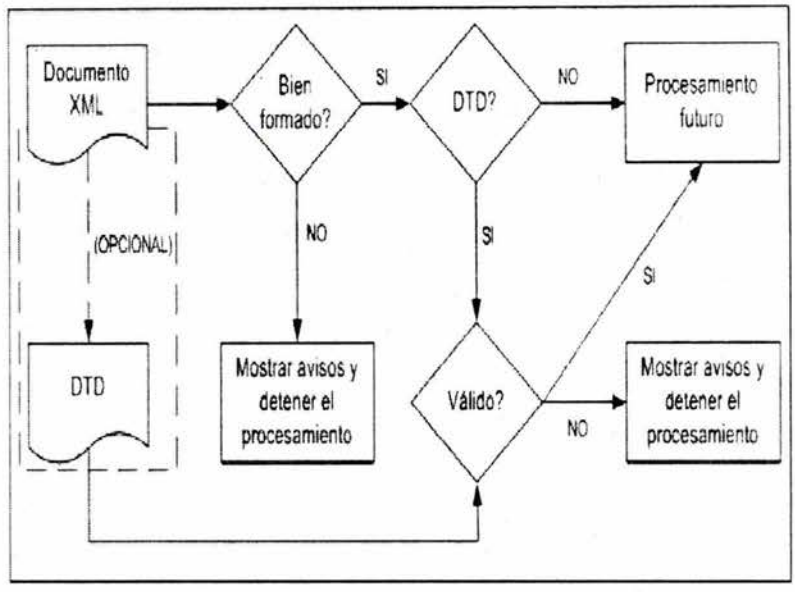


Figura 2.4: Cómo un parser intrepreta un documento XML.

2.12 Ventajas

XML al ser un estándar para la representación de la información cuenta con una gran cantidad de ventajas como son:

- Permite coleccionar e integrar información de múltiples fuentes.

  XML es muy flexible en cuanto a su estructura por lo que es fácil representar, intercambiar e interpretar los datos contenidos en el documento.

- Largos documentos son entendibles.

  Al manejar una estructura basada en componentes se puede accesar a los componentes necesarios y no a todo el documento.

- Conjunto de etiquetas extensible.

  Permite al usuario definir su propias etiquetas a emplear para representar su información.

- Es código abierto.

  Tanto el estándar de XML como la mayoría de las herramientas para procesar documentos XML son libres. Las herramientas facilitan la manipulación de la información y generalmente están desarrolladas en un lenguaje de alto nivel.

- Facilita el envío de datos sobre la web.

  Permitirá que los motores de búsqueda sean mucho más eficientes y esto facilitará el intercambio de información.

- Es independiente de plataforma.

  El estándar XML maneja documentos de texto.

- Múltiples vistas.

  El estándar cuenta con herramientas como son las hojas de estilo(StyleSheet). CSS para poder definir múltiples vistas, para un mismo conjunto de datos y permiten transformar documentos XML a otro formato.

2.13 Desventajas

Aunque XML es una herramienta muy poderosa que puede ser empleada en diversas áreas para representar y compartir información, es necesario definir un filosofía de cuando emplearlo y cuando no. En general XML no es adecuado para el manejo de datos rigurosamente estructurados, por ejemplo:

- Para representar documentos técnicos.
- Para representar expresiones matemáticas.

Debido a que en algunos casos es más fácil emplear otra notación en lugar de XML, por lo que decidir cuando emplear esta herramienta dependerá del tipo de aplicación y del contexto que se tiene.

También hay que tomar en cuenta que para poder lograr una mejor forma de compartir datos a través de XML, cada empresa debe comenzar un proceso de transformación, lo cual implica recursos, tiempo y dinero para cambiar los formatos que emplean actualmente por XML.

2.14 Aplicaciones.

XML puede ser empleado en una gran cantidad de áreas, debido facilita el intercambio y representación de información con un formato estándar. Actualmente XML ha comenzado a ser una de las principales herramientas para representar y compartir información, las principales áreas donde se emplea XML son:

- Aplicaciones WEB.
  - Comercio electrónico.
- Desarrollo de lenguaje de marcas
  - CML: lenguaje de marcas para química.
  - DFX: casa de cambio.
- Aplicaciones Gráficas.
  - Software comercial.
  - Software libre.

- Educación.
  - Bases de datos.

2.15 Futuro de XML

XML ha iniciado la revolución de la información, actualmente ha comenzado a ser una de las principales herramientas para representar la información.

Al comenzar a fusionarse algunos lenguajes de programación de alto nivel con XML, han comenzado a desarrollarse herramientas muy poderosas que capturan y aprovechar las ventajas que proporcionan ambos lenguajes.

XML como herramienta va a ir revolucionando el manejo de información, para seguir fomentando el intercambio de información de una forma inteligente, lo que permitirá que los procesos de manejo de información comiencen a ser más eficientes.

El efecto que XML ha comenzado a tener es muy grande debido a la flexibilidad que presenta al no depender de una o mas aplicaciones específicas (navegador, compilador o editor) lo que permitirá y facilitará el intercambio de información.[15]

Capítulo 3

Un editor para documentos XML

3.1 Introducción.

En este capítulo se presenta el desarrollo de una herramienta que tiene como objetivo facilitar la creación, edición y revisión de documentos XML. En la primer sección se presenta la *captura de requerimientos*, después se definen las etapas de *análisis* y *diseño*.

Finalmente en la etapa de *implementación* se presenta un panorama general sobre el desarrollo de la aplicación a nivel programación y sólo el punto de revisión de documentos se realiza con detalle.

3.2 Captura de Requerimientos.

Para desarrollar software como en cualquier otra actividad, se requiere contar un *proceso de desarrollo*, el cual permita tener un buen control y una mejor planeación de lo que se desea desarrollar. Durante todo este proyecto se empleará como proceso de desarrollo de software el *Proceso Unificado* [1], el cual proporiona las actividades necesarias para transformar los requerimientos de un usuario en software.

El *Proceso Unificado* se caracteriza por ser *iterativo e incremental*, emplea el *Lenguaje de Modelado Unificado (UML)* [2] como herramienta para realizar la modelación del software que se desea desarrollar.

La *Captura de requierimientos* representa la paso inicial del *Proceso Unificado*, el objetivo de esta etapa etapa es desarrollar una visión del producto final que se desea desarrollar.

3.2.1 Objetivo del sistema.

La aplicación que se desea desarrollar es un editor para documentos XML, el cual tiene como objetivo facilitar la creación, edición y revisión de documentos XML.

3.2.2 Descripción del sistema.

Se desea desarrollar una herramienta que permita editar documentos XML, básicamente se desea que esta herramienta cuente con una interfaz gráfica que facilite a los usuarios la

[1] Ver Anexo A *Proceso Unificado*
[2] Ver Anexo B *UML*

elaboración y trabajo con este tipo de documentos.

El editor debe contar con operaciones básicas para la edición de documentos de texto como son: abrir, guardar, buscar, seleccionar, remplazar, imprimir, etc. La generación de los documentos podrá ser realizada por edición libre o dirigida por sintaxis.

También debe contar con la funcionalidad necesaria que permita el desarrollo de las siguientes actividades:

- **Editar documentos XML bien formados. (.xml)**

  Este módulo básicamente debe garantizar que los documentos que genere cumplan con la característica de ser documentos XML bien formados, las principales operaciones que debe proporcionar son:

  1. Generar nuevos documentos XML bien formados.
  2. Abrir documentos XML bien formados.
  3. Guardar documentos XML bien formados.
  4. Editar documentos XML bien formados.
  5. Verificar que documentos XML sean bien formados.

- **Editar documentos DTD. (.dtd)**

  Este módulo permite editar documentos DTD y verifica que el contenido sea sintácticamente y semánticamente correcto, las principales operaciones que debe proporcionar son:

  1. Generar nuevos documentos DTD
  2. Abrir documentos DTD.
  3. Guardar documentos DTD.
  4. Editar documentos DTD.
  5. Realizar la verificación sintáctica y semántica del documento DTD.

- **Documentos XML Válidos. (.xml)**

  Este módulo garantiza que los documentos que genera sean documentos XML válidos, permite realizar las siguientes actividades:

  1. Soporta las operaciones de los documentos XML bien formados.
  2. Verifica que el documento DTD al que se hace referencia sea sintácticamente y semánticamente correcto.
  3. Verifica que la estructura del documento XML se realiza en base a las reglas definidas en el documento DTD.

- **Convertir documentos de texto a documentos XML válidos.**

  Este módulo tiene como objetivo generar documentos XML válidos a partir de documentos de texto, las principales actividades que debe realizar son:

1. Seleccionar el documento de texto a convertir.
2. Definir la lista de etiquetas a emplear para generar el documento XML válido donde:
   - Cada etiqueta tiene asociado un nombre y un color.
   - La lista puede ser:
     * Definida en ese instante por el usuario.
     * Puede ser obtenida de un archivo.
3. Realizar el proceso de conversión:
   - Se presenta al usuario un área de trabajo la cual está formada de:
     * Un área de edición que contiene al documento seleccionado por el usuario. (*área1*)
     * Un área de edición que contiene al documento XML válido que se va generando. (*área2*)
     * La lista de etiquetas.
   - En esta área de trabajo el usuario puede:
     * Seleccionar un color de la lista de etiquetas.
     * Seleccionar texto en el *área1*.
   - Esto permitirá al sistema:
     * Marcar el texto seleccionado, en el *área1*, con el color seleccionado por el usuario.
     * Marcar el texto seleccionado, en el *área2*, con la etiqueta asociada al color seleccionado por el usuario.
4. Guardar el documento XML válido obtenido.
5. Abrir un documento.

El editor podrá determinar que operación va a realizar de acuerdo al tipo de documento que tenga activo en ese momento. Durante todo el proceso de edición de documentos, el usuario siempre deberá tener el control de la aplicación y si llega a generarse alguna excepción, el sistema deberá proporcionar la información necesaria para que el usuario pueda localizar y corregir el error.

3.2.3 Lista de características deseadas.

A continuación se presenta la lista de características deseadas para el sistema:

1. Fácil de usar e intuitivo por los usuarios.
2. Permita la edición de documentos XML bien formados.
3. Permita la edición de documentos DTD.
4. Permita la edición de documentos XML válidos.
5. Permita la edición de documento de texto.
6. Permita la conversión de documentos de texto a documentos XML Válidos.
7. Cuente con la operaciones básicas de edición, administración y configuración.

8. Permita a los usuarios tener siempre el control de la aplicación.
9. Maneje las validaciones necesarias de acuerdo al tipo de documento.
10. Soporte excepciones.
11. Proporcione la información necesaria cuando se generen excepciones.

3.2.4 Requerimientos funcionales.

La funcionalidad requerida por el *Editor para documentos XML* es presentada en los *siguientes* diagramas de *Casos de uso.*

Diagramas generales

- En el diagrama de la Figura 3.1 se presentan los módulos soportados por el sistema.

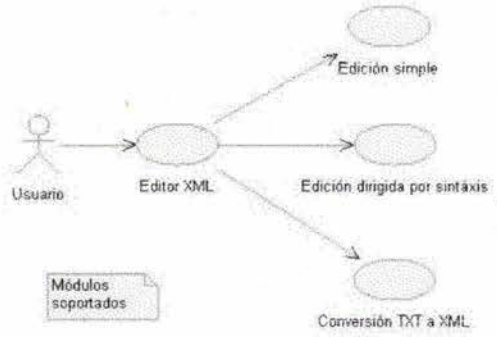


Figura 3.1: Módulos soportados por el editor.

- En el diagrama de la Figura 3.2 se presentan los documentos que pueden ser soportados por el sistema.

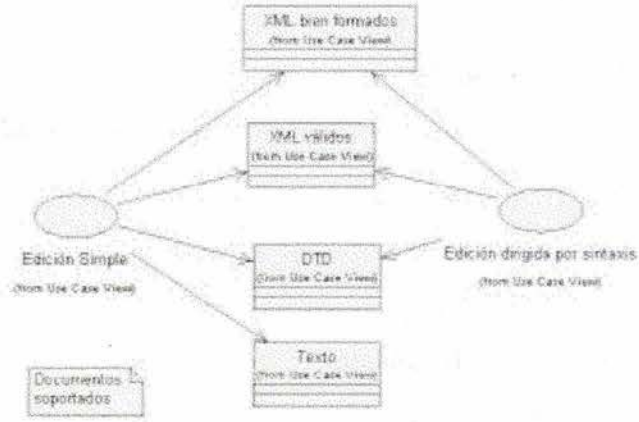


Figura 3.2: Documentos soportados.

- En el diagrama de la Figura 3.3 se presenta el alcance del sistema y las operaciones que un usuario puede realizar a través del editor, la funcionalidad de la operacion *revisión* va a depender del tipo de documento que esté editando el usuario en ese momento.

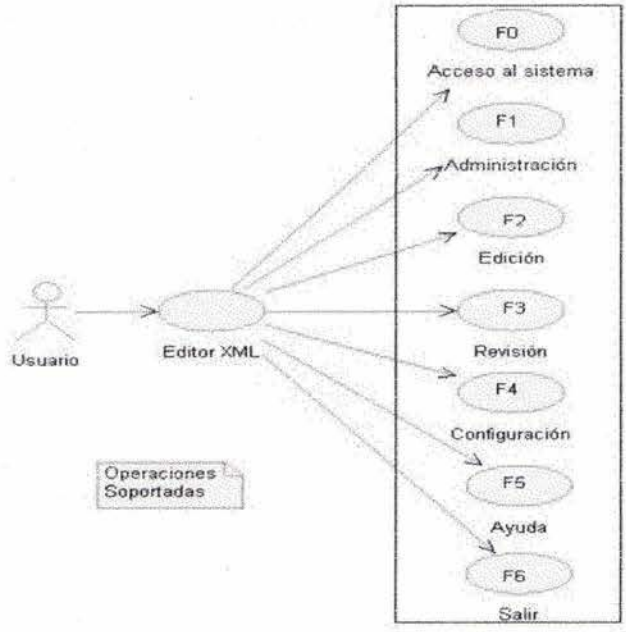


Figura 3.3: Operaciones soportadas.

3.2.5 Diagramas de casos de uso

En los siguientes diagramas se presentan las actividades que un usuario puede realizar en el *Editor para documentos XML.*

CASO DE USO:	Ingresar al sistema
IDENTIFICADOR:	F0
ACTORES:	Usuario, Editor XML
DESCRIPCIÓN:	Como se observa en la Figura 3.3 cualquier usuario puede ingresar para generar algún documento soportado por la aplicación.

FLUJO:

ACTOR		SISTEMA		
Paso	**Acción**	**Paso**	**Acción**	**Excepción**
1	Usuario ejecuta la aplicación	2	Sistema carga configuración inicial.	NA
		3	Se muestra la pantalla principal.	

CASO DE USO: Administración
IDENTIFICADOR: F1
ACTORES: Usuario, Editor XML
DESCRIPCIÓN: Operaciones de administración que un usuario puede emplear, en los documentos soportados por la aplicación.

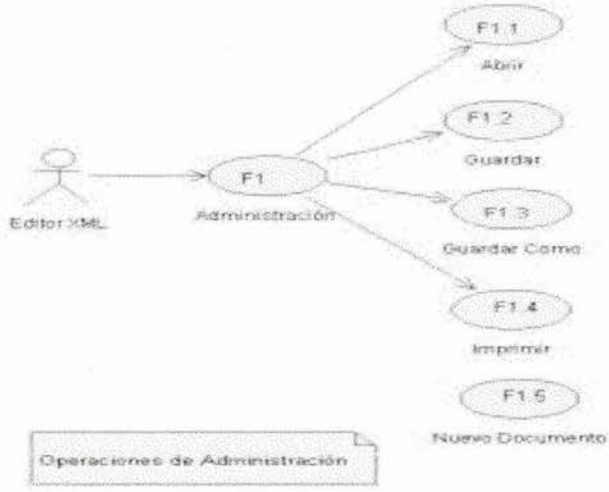


Figura 3.4: Operaciones de administración.

CASO DE USO: Abrir
IDENTIFICADORES: F1.1
ACTORES: Usuario, Editor XML
DESCRIPCIÓN: En la Figura 3.5 se presentan los documenos que pueden ser abiertos empleando la herramienta.

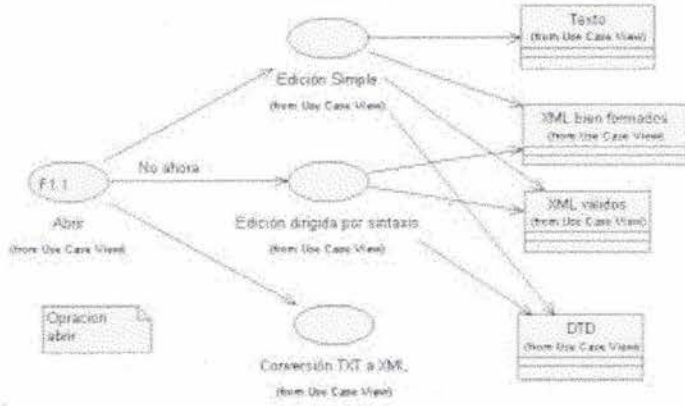


Figura 3.5: Operación abrir documento.

FLUJO

ACTOR		SISTEMA		
Paso	**Acción**	**Paso**	**Acción**	**Excepción**
1	Usuario selecciona opción abrir documento.	2	Se muestra diálogo de abrir archivo.	
3	Usuario selecciona extensión del tipo de documento. a abrir.	4	Sistema muestra los archivos que cuentan con esta extensión	
5	Usuario selecciona archivo.			
6	Usuario selecciona la opción aceptar	7	Sistema abre el archivo seleccionado por el usuario.	E1
		8	Sistema inicializa el área de edición.	
		9	Sistema muestra el documento	

EXCEPCIONES:

ID	NOMBRE	ACCIÓN
E1	Error al abrir el documento	Mostrar aviso al usuario

CASO DE USO: Guardar, Guardar como.
IDENTIFICADORES: F1.2 y 1.3
ACTORES: Usuario, Editor XML
DESCRIPCIÓN: En la Figura 3.6 se presentan los tipos de documentos que pueden ser almacenados.

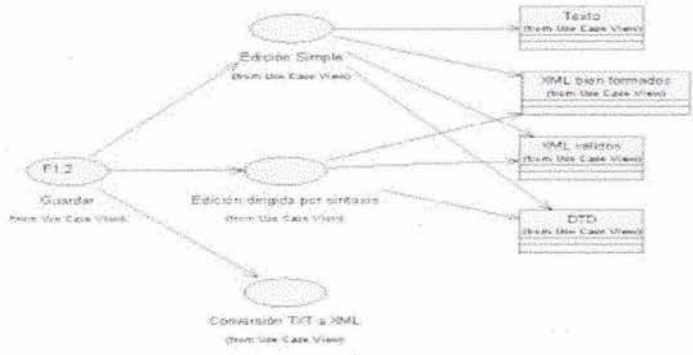


Figura 3.6: Operación guardar documento.

FLUJO:

ACTOR		SISTEMA		
Paso	**Acción**	**Paso**	**Acción**	**Excepción**
1	Usuario selecciona opción edición está activa.	2	Sistema identifica que área de texto está activa.	
		3A	Sistema identifica que el documento. tiene asociado un nombre.	
		4A	Sistema obtiene texto del área de edición. activa.	
		5A	Sistema crea el archivo y guarda la información.	E1
		3B	Sistema identifica que el documento. no tiene asociado un nombre.	
		4B	Sistema muestra diálogo guardar como.	
5B	Usuario selecciona directorio			
6B	Usuario proporciona nombre.			
7B	Usuario selecciona opción aceptar	8B	Sistema crea el archivo y guarda	E1

EXCEPCIONES:

ID	NOMBRE	ACCIÓN
E1	Error al guardar documento	Mostrar aviso al usuario

CASO DE USO: Imprimir
IDENTIFICADOR: F1.4
ACTORES: Usuario, Editor XML
DESCRIPCIÓN: La Figura 3.7 presenta los tipos de documentos que pueden ser impresos.

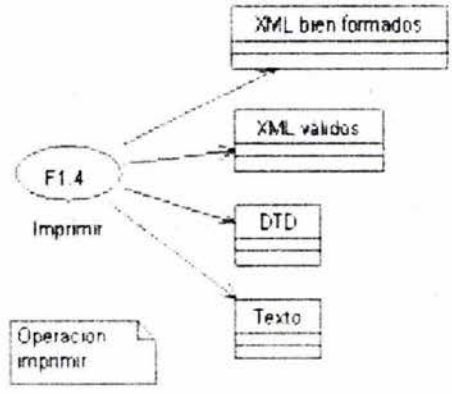


Figura 3.7: Operación imprimir documento.

FLUJO:

ACTOR		SISTEMA		
Paso	**Acción**	**Paso**	**Acción**	**Excepción**
1	Usuario selecciona opción imprimir documento.	2	Sistema muestra diálogo de imprimir.	
3	Usuario selecciona opción aceptar.			
		4	Sistema identifica área de edición activa.	
		5	Sistema imprime el documento	E1

EXCEPCIONES:

ID	NOMBRE	ACCIÓN
E1	Error al imprimir el documento	Mostrar aviso al usuario

CASO DE USO: Crear Nuevo Documento
IDENTIFICADOR: F1.5
ACTORES: Usuario, Editor XML
DESCRIPCIÓN: La Figura 3.8 presenta los documentos que pueden ser generados.

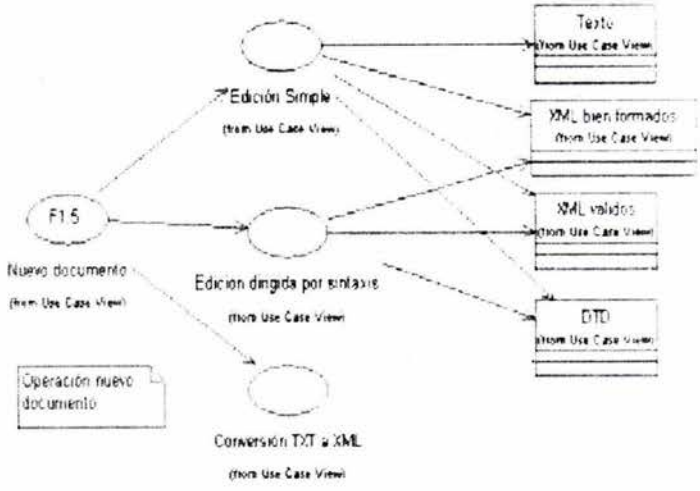


Figura 3.8: Operación crear nuevo documento.

FLUJO:

ACTOR		SISTEMA		
Paso	**Acción**	**Paso**	**Acción**	**Excepción**
1	Usuario selecciona opción crear nuevo documento.	2	Sistema muestra diálogo de tipos de documentos.	
3	Usuario selecciona tipo de documento a crear.			
4	Usuario selecciona aceptar			
		5	Sistema inicia proceso de creación de nuevo documento. (Ver casos F1.5.*)	

EXCEPCIONES:

ID	NOMBRE	ACCIÓN

CASO DE USO: Crear nuevo documento
IDENTIFICADORES: F1.5.1, F1.5.2, F1.5.3
ACTORES: Usuario, Editor XML
DESCRIPCIÓN: Operación nuevo documento.

FLUJO: F1.5.1

ACTOR		SISTEMA		
Paso	**Acción**	**Paso**	**Acción**	**Excepción**
		1	Sistema inicializa un área de edición.	
		2 2	Sistema muestra al usuario el área de edición.	

FLUJO: F1.5.2

ACTOR		SISTEMA		
Paso	**Acción**	**Paso**	**Acción**	**Excepción**
		1	Sistema solicita información para generar componentes básicos.	
2	Usuario proporciona información	3	Sistema genera componentes básicos.	
		4	Sistema inicializa área de edición.	
		5	Sistema presenta el área de edición.	

FLUJO: F1.5.3

ACTOR		SISTEMA		
Paso	**Acción**	**Paso**	**Acción**	**Excepción**
		1	Sistema solicita archivo de texto	
		2	Sistema identifica lista de etiquetas.	
		3	Sistema genera documento XML y lo asigna a un área de edición.	
			Sistema asigna documento de texto a un área de edición.	

EXCEPCIONES:

ID	NOMBRE	ACCIÓN

CASO DE USO: Edición
IDENTIFICADOR: F2
ACTORES: Usuario, Editor XML
DESCRIPCIÓN: La Figura 3.9 las operaciones de edición que son soportadas por el editor.

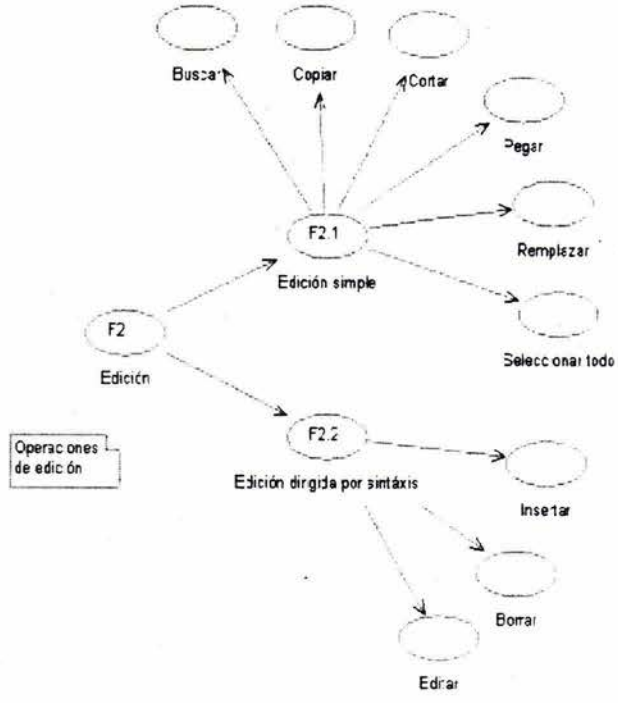


Figura 3.9: Operaciones de edición.

CASO DE USO: Edición
IDENTIFICADOR: F2.1, F2.2
ACTORES: Usuario, Editor XML
DESCRIPCIÓN: Operaciones para edición simple y dirigida por sintaxis.

FLUJO: F2.1

ACTOR		SISTEMA		
Paso	**Acción**	**Paso**	**Acción**	**Excepción**
1	Usuario selecciona operación a realizar.	2	Sistema aplica operación al documento actual.	
		3	Presenta cambios realizados.	

FLUJO: F2.2

ACTOR		SISTEMA		
Paso	**Acción**	**Paso**	**Acción**	**Excepción**
1	Usuario selecciona componente.			
2	Selecciona operación a realizar.			
		3	Sistema aplica operación.	
		4	Presenta cambios realizados.	

EXCEPCIONES:

ID	NOMBRE	ACCIÓN

CASO DE USO: Revisión
IDENTIFICADOR: F3
ACTORES: Editor XML
DESCRIPCIÓN: La Figura 3.10 presenta los documentos que pueden ser revisados con el editor.

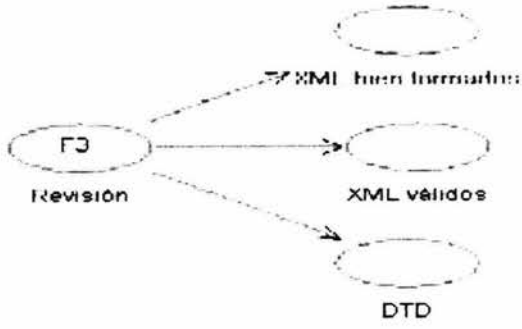


Figura 3.10: Operaciones de revisión.

FLUJO:

ACTOR		SISTEMA		
Paso	**Acción**	**Paso**	**Acción**	**Excepción**
1	Usuario selecciona opción revisar documento.	2	Sistema identifica área de edición.	
		3	Sistema obtiene texto del documento actual	
		4	Sistema crea instancia del parser y asigna texto.	
		5	Sistema realiza el parser del documento.	E1
		6A	Sistema muestra diálogo revisión exitosa.	

EXCEPCIONES:

ID	NOMBRE	ACCIÓN
E1	Erro al revisar el documento	Mostrar mensaje al usuario

CASO DE USO: Configuración
IDENTIFICADOR: F4
ACTORES: Editor XML
DESCRIPCIÓN: La Figura 3.11 presenta los elementos que pueden ser modificados para definir la forma en que será presentado el editor al usuario.

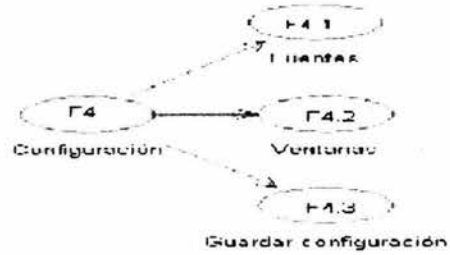


Figura 3.11: Operaciones de configuración.

FLUJO: F4.1

ACTOR		SISTEMA		
Paso	**Acción**	**Paso**	**Acción**	**Excepción**
1	Usuario selecciona opción configurar fuentes.	2	Sistema muestra menú con posibles tamaños, tipos y estilos a usar.	
3	Usuario selecciona opción.	4	Sistema identifica opción y aplica propiedad a documentos actuales.	

FLUJO: F4.2

ACTOR		SISTEMA		
Paso	**Acción**	**Paso**	**Acción**	**Excepción**
1	Usuario selecciona opción configurar ventanas.	2	Sistema muestra menú con propiedades alinear, background	
3A	Usuario selecciona opción alinear.	4A	Sistema alinea ventanas.	
3B	Usuario selecciona opción background.	4B	Sistema muestra diálogo configurar background.	
5B	Usuario selecciona color.			
6B	Usuario selecciona aceptar	7B	Sistema aplica color a ventanas actuales.	

FLUJO: F4.3

ACTOR		SISTEMA		
Paso	**Acción**	**Paso**	**Acción**	**Excepción**
1	Usuario selecciona opción guardar configuración.	2	Sistema crear archivo XML válido con los datos de configuración actual	
		3	Sistema almacena documento	E1
		4A	Sistema muestra diálogo almacenamiento exitóso	

EXCEPCIONES:

ID	NOMBRE	ACCIÓN
E1	Error alamacenar el documento.	Mostrar mensaje el usuario.

CASO DE USO: Ayuda
IDENTIFICADOR: F5
ACTORES: Editor XML
DESCRIPCIÓN: La Figura 3.12 presenta la información a la que tiene acceso el usuario al seleccionar la operaciones de ayuda.

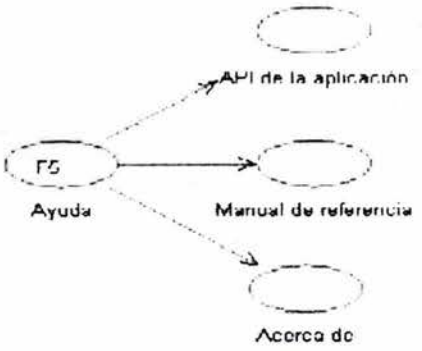


Figura 3.12: Operaciones de ayuda.

FLUJOS

ACTOR		SISTEMA		
Paso	**Acción**	**Paso**	**Acción**	**Excepción**
1 1	Usuario selecciona opción ayuda.	2 2	Sistema muestra menú de ayuda.	
3A	Usuario selecciona opción Manual de referencia.	4A	Sistema muestra diálogo. manual de referencia.	
3B	Usuario selecciona opción API de la aplicación.	4B	Sistema muestra dialogo API de la aplicación.	
3C 3C	Usuario selecciona opción Acerca de.	4C 4C	Sistema muestra dialogo Acerca de.	
6 6	Usuario selecciona cerrar diálogo.	4B 4B	Sistema oculta diálogo. Sistema oculta diálogo.	

EXCEPCIONES:

ID	NOMBRE	ACCIÓN

CASO DE USO: Salir del sistema.
IDENTIFICADOR: F6
ACTORES: Usuario, Editor XML
DESCRIPCIÓN: El usuario solicita salir de la aplicación

FLUJO:

ACTOR		SISTEMA		
Paso	**Acción**	**Paso**	**Acción**	**Excepción**
1	Usuario selecciona opcion salir	2	Sistema identifica áreas de edición activas.	
		3	Sistema guarda cambios.	
		4	Sistema termina.	

EXCEPCIONES:

ID	NOMBRE	ACCIÓN

3.2.6 Requerimientos no funcionales.

Básicamente se desea que el funcionamiento de la aplicación sea correcto, eficiente y que facilite a los usuarios desarrollar documentos XML. Esta herramienta será desarrollada en un ambiente gráfico y su funcionamiento deberá ser independiente de la plataforma en la que se trabaje.

Los requerimientos necesarios para que un cliente pueda ejecutar la aplicación con un buen desempeño son:

- Hardware
  - Computadora Pentium a 140 Mhz.
  - 64 MB de memoria RAM.
- Software
  - JDK o JRE versión 1.3 o superior.

3.2.7 Plan de pruebas.

El plan de pruebas fue desarrollado en base a los casos de uso defnidos durante la etapa de captura de requerimientos. En la tabla de la siguiente página se presentan las actividades a desarrollar para verficar la funcionalidad del editor.

CASO DE USO	CASO DE PRUEBA	RESULTADO
F1.1 Válido.	Abrir un documento soportado por el sistema.	Se muestra un área de edición con documento.
F1.1 Inválido.	Abrir un documento que no existe.	Se muestra diálgo con mensaje: "No exite el archivo"
F1.2 Válido	Modificar documento de F1.1 y guardar cambios	Documento almacenado con las modificaciones realizadas.
F1.3	Abrir un documento soportado por la aplicación y renombrarlo.	Documento almacenado.
F1.4 Válido	Imprimir un documento soportado por la aplicación.	Documento impreso.
F1.5	Crear un documento soportado por la aplicación.	Area de edición se presenta al usuario.
F2.1	Buscar texto en un área de edición área de edición.	Texto seleccionado en el
F2.2	Copiar texto de un área de edición aplicación.	Texto almacenado en el buffer de la
F2.3	Cortar texto de un área de edición de la aplicación.	Texto cortado almacenado en el buffer
F2.4	Pegar texto un área de edición en posición actual del cursor.	Se inserta texto contenido en el buffer
F2.5	Remplazar texto un área de edición de edición.	Palabras remplazadas en el área
F2.6	Seleccionar todo el texto de un área de edición	Texto seleccionado
F2.7	Seleccionar opción ir a linea. de línea proporcionada por el usuario.	Cursos se posiciona en el número
F3.1 Válido.	Revisar documento XML Bien Formado.	Mensaje: "Revisión completa."
F3.1 Inválido.	Revisar documento XML Bien Formado.	Se presenta mensaje indicando el error detectado.
F3.2 Válido.	Revisar documento DTD.	Mensaje: "Revisión completa."
F3.2 Válido.	Revisar documento DTD. el error detectado.	Se presenta mensaje indicando
F3.3 Válido.	Revisar documento XML Válido.	Mensaje: "Revisión completa."
F3.3 Válido.	Revisar documento XML Válido.	Se presenta mensaje indicando el error detectado.
F4.1	Configurar propiedades de fuentes	Configuración aplicada a documentos actuales.
F4.2	Configurar propiedades de Background.	Configuración aplicada a documentos actuales.
F4.4	Seleccionar opción guardar configuración	Configuración actual almacenada.
F5.1 F5.2 F5.3	Seleccionar opción ayuda Seleccionar opción API Seleccionar opción Acerca de	Se presenta diálogo de ayuda. Se presenta diálogo API. Se presenta diálogo Acerca de.

3.2.8 Prototipo.

A continuación se presenta el prototipo de algunas pantallas a emplear en el sistema. Estas pantallas fueron desarrolladas empleando la herramienta *Sun ONE Studio*[3].

- Pantalla principal.

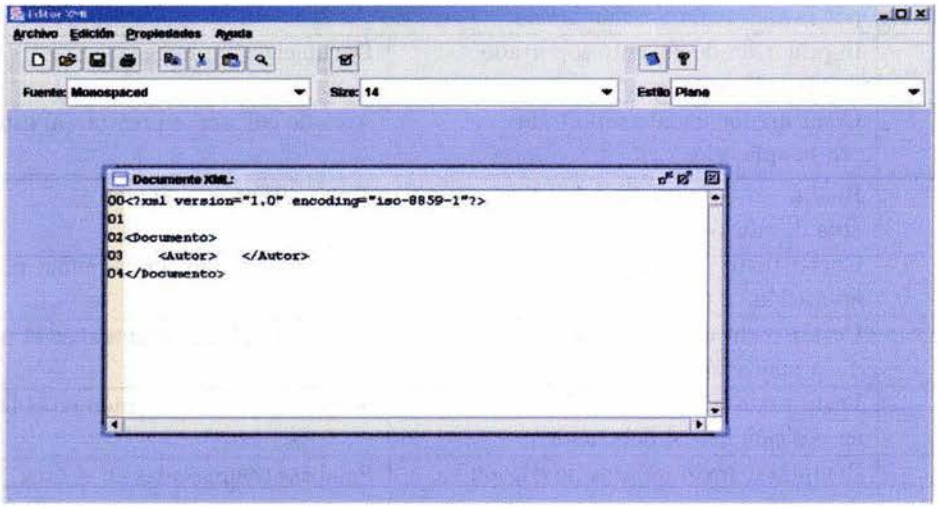


Figura 3.13: Pantalla de edición simple.

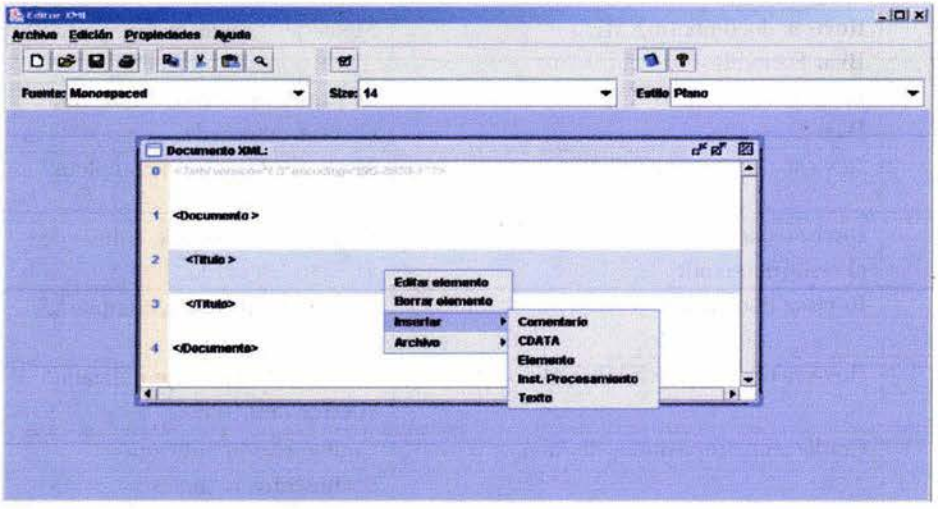


Figura 3.14: Pantalla de edición dirigida por sintaxis.

[3]Ver Anexo C Herramientas

- Los principales componentes que forman parte del editor son:
  - Área de edición.
  - Barra de herramientas gráfica.
  - Barra de herramientas de texto.

  Las barras de herramientas contienen las opciones de las tareas que pueden realizarse, estas opciones se habilitan o deshabilitan de acuerdo al tipo de documento que se está editando en ese momento.

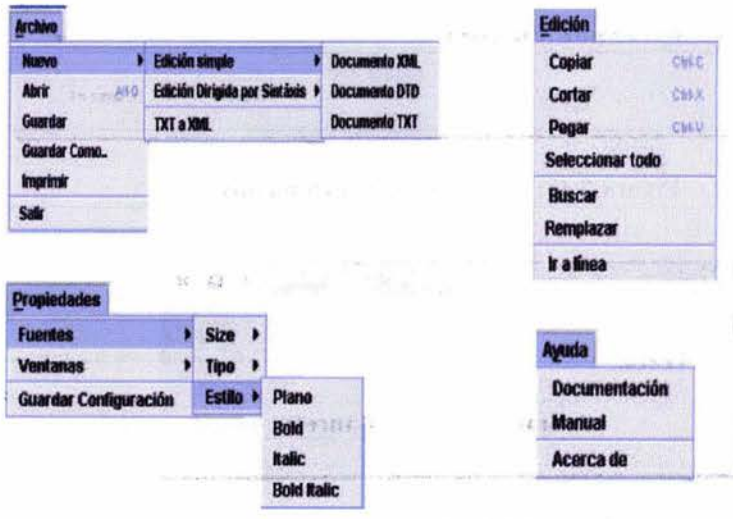


Figura 3.15: Barra de herramientas.

- Diálogos

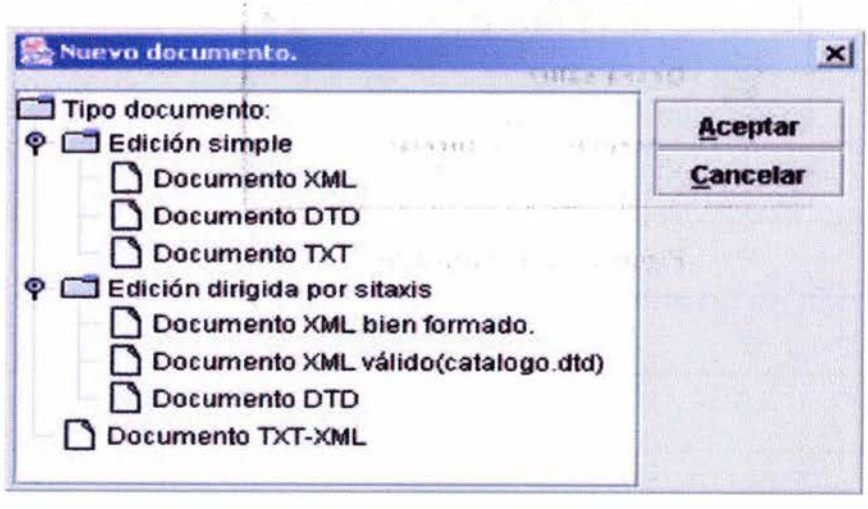


Figura 3.16: Diálogo nuevo documento.

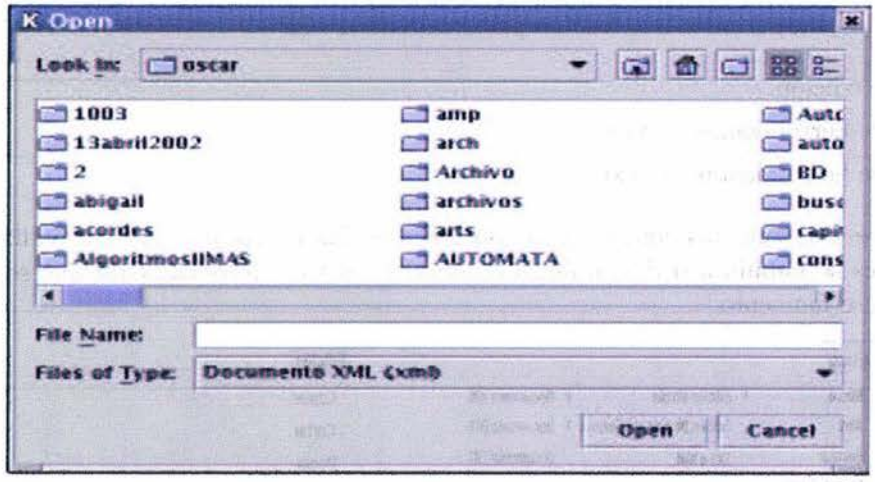


Figura 3.17: Diálogo abrir documento.

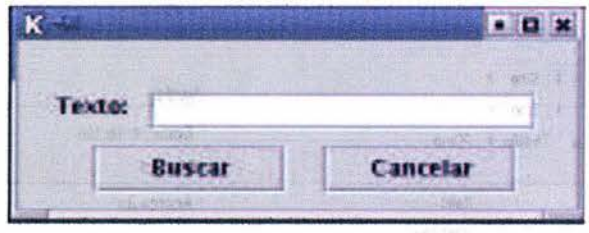


Figura 3.18: Diálogo buscar.

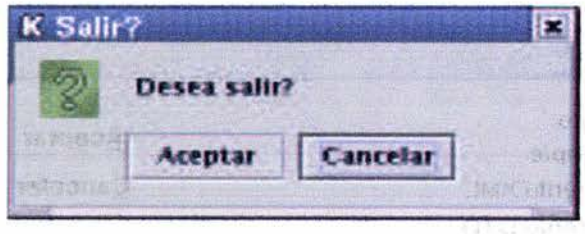


Figura 3.19: Diálogo salir.

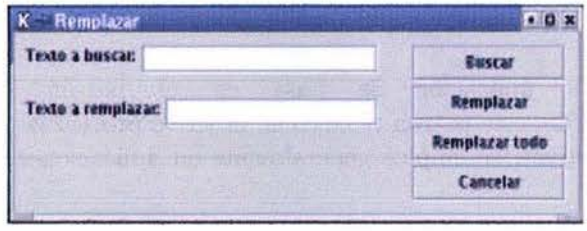


Figura 3.20: Diálogo remplazar.

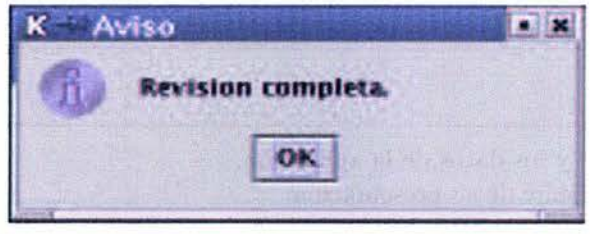


Figura 3.21: Diálogo de aviso.

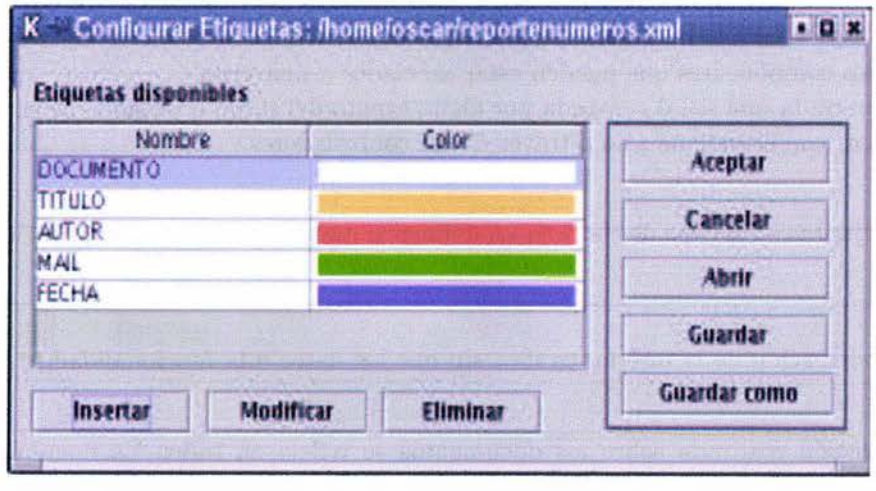


Figura 3.22: Diálogo configuración etiquetas.

3.3 Análisis.

El desarrollo de esta aplicación se basa en el patrón de arquitectura *Modelo-Vista-Controlador (MVC)*, este patrón tiene como objetivo proveer de una *organización* estructural de sistemas interactivos, se emplea generalmente en aplicaciones que cuentan con una interfaz gráfica.

Esto permite definir la estructura de la aplicación en subsistemas, especificar responsabilidades, reglas, lineamientos para definir y organizar cada uno de ellos.

MVC representa uno de los mejores patrónes de arquitectura se caracteriza por dividir una aplicación en tres componentes:

- Modelo:

  Contiene la funcionalidad y los datos de la aplicación.
  Su definición es independiente de su presentación.

- Vista:

  Se encarga de presentar la información al usuario.
  Obtiene los datos a presentar de la capa de modelo.

- Controlador:

  Representan los componentes que pueden estar asociados a una vista.
  Recibe como entrada una señal generada por algún evento del ratón o teclado. El usuario puede interactar con el sistema sólo a través de los controladores.

La capa de vista junto con la capa de controlador definen la interfáz gráfica de las aplicaciones.

Ventajas:

- Facilita la modificación de la interfaz gráfica sin que los datos o la funcionalidad se vean afectados.
- Toda configuración realizada sobre los documentos se refleja en todos los documentos inmediatamente.
- Al contar con la separación del procesamiento y los controladores permite generar múltiples vistas para un mismo modelo.

3.3.1 Diagrama de paquetes.

Una vez aplicado el patrón MVC a este caso de estudio la organización de los elementos quedó definida de la siguiente forma: la Figura 3.23 presenta los elementos que forman parte de la capa de *Modelo*, en la Figura 3.24 se presentan los componentes que definen la capa de *Vista*. finalmente la Figura 3.25 presenta los elementos que constituyen la capa *Controlador*.

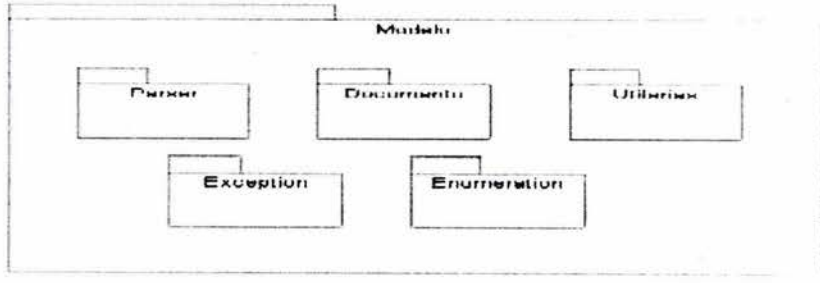


Figura 3.23: Capa de Modelo.

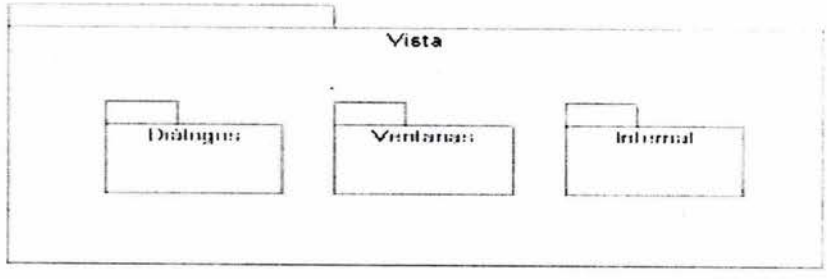


Figura 3.24: Capa de Vista.

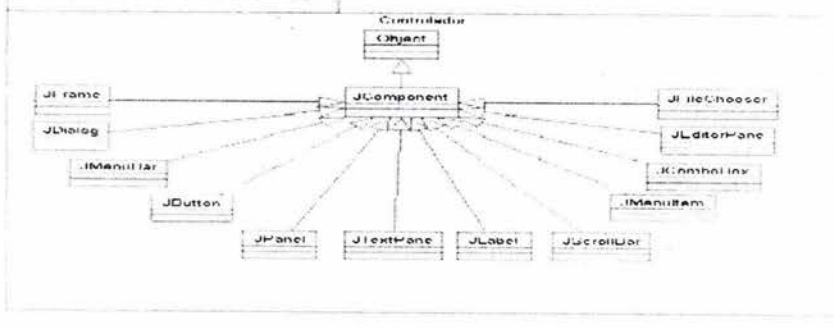


Figura 3.25: Capa de Controlador.

Cada uno de elementos definidos en el paquete de la capa de *Modelo* están formados por:

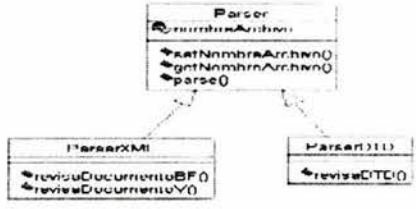


Figura 3.26: Paquete parser.

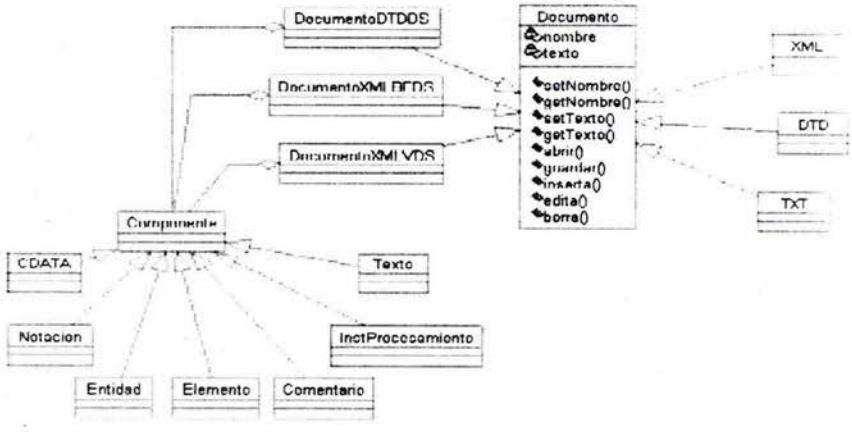


Figura 3.27: Paquete documento.

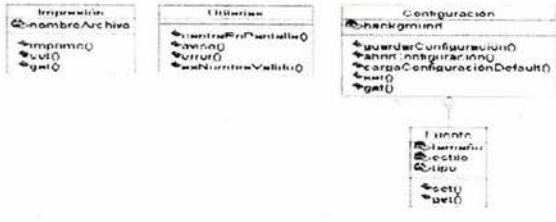


Figura 3.28: Paquete Utilerias.

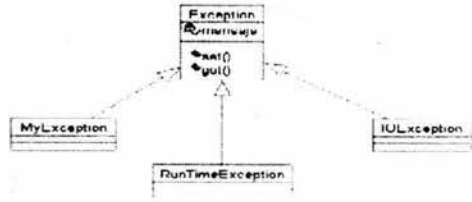


Figura 3.29: Paquete Excepciones.

Cada uno de elementos definidos en el paquete de la capa de *Vista* están formados por:

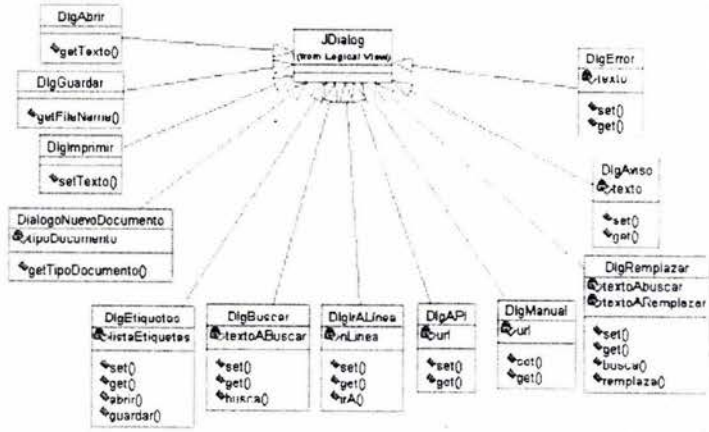


Figura 3.30: Paquete Diálogos.

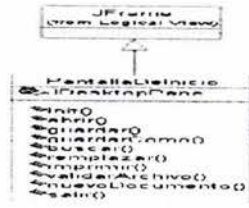


Figura 3.31: Paquete Ventanas.

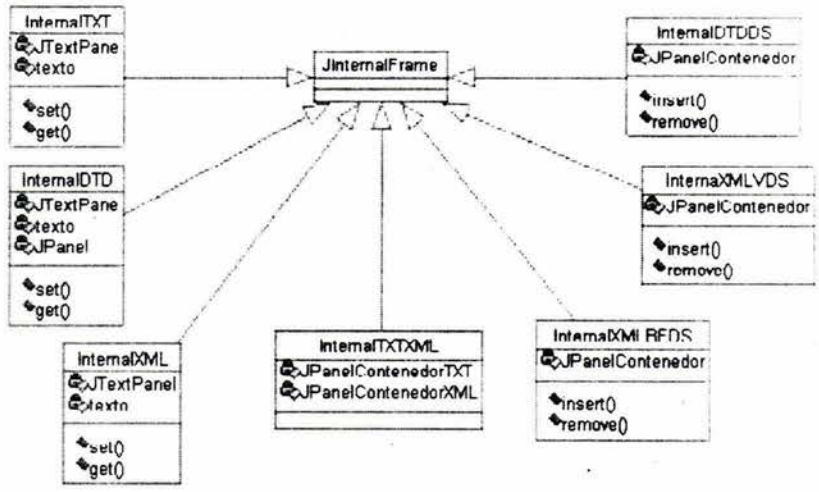


Figura 3.32: Paquete Internal.

3.3.2 Diagrama de clases.

La Figura 3.33 presenta la línea base para el desarrollo de este proyecto. En este diagrama se definen las clases a desarrollar así como la relación que hay entre ellas.

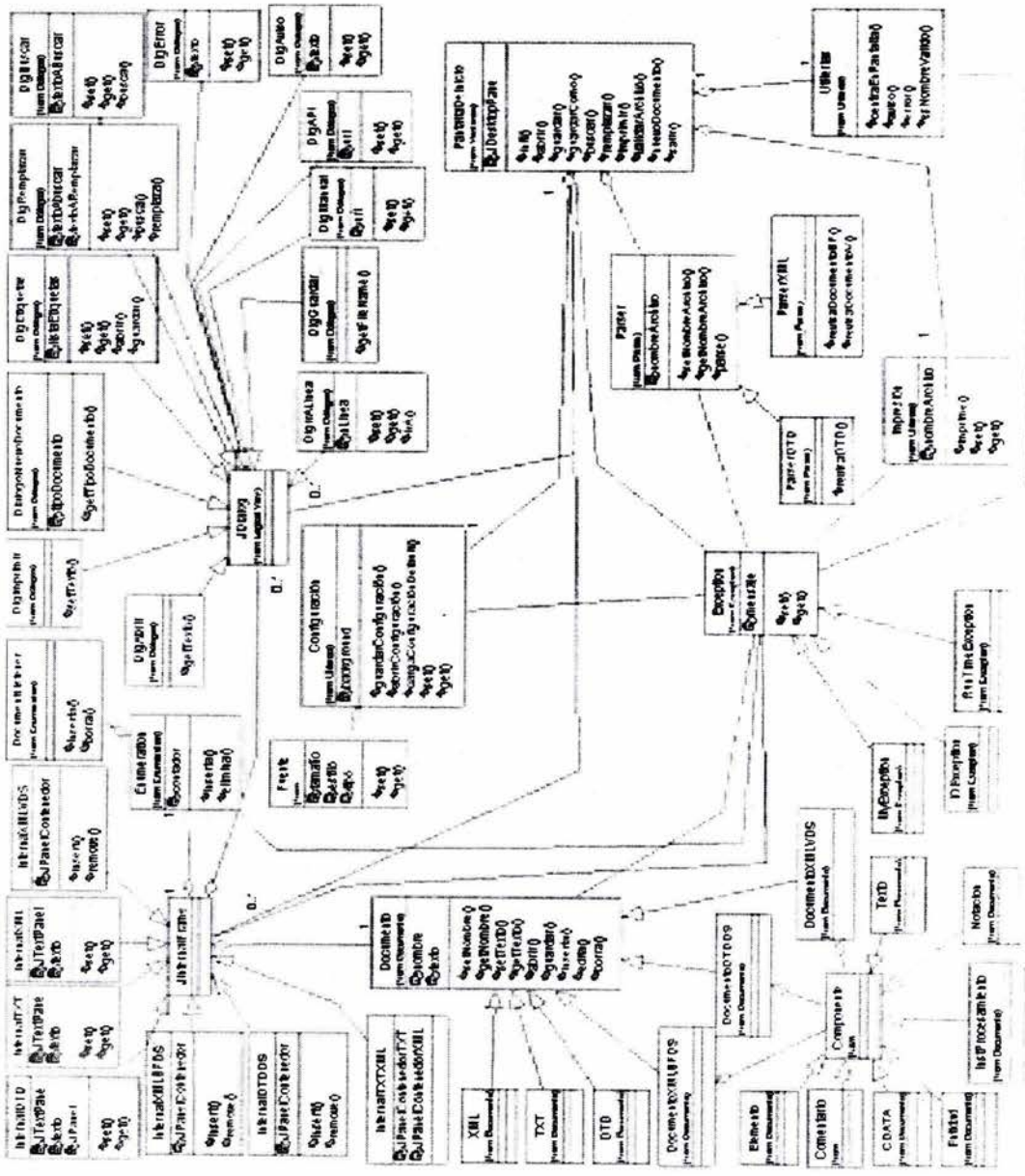


Figura 3.33: Diagrama de clases.

3.3.3 Diagramas de interacción.

En las Figuras 3.34, 3.35, 3.36 y 3.37 se presentan los *Diagramas de Interacción* asociados con las operaciones *abrir, guardar, imprimir y revisar.* Este tipo de diagramas permite obtener los métodos que debe tener asociada una clase para poder interacturar con otras.

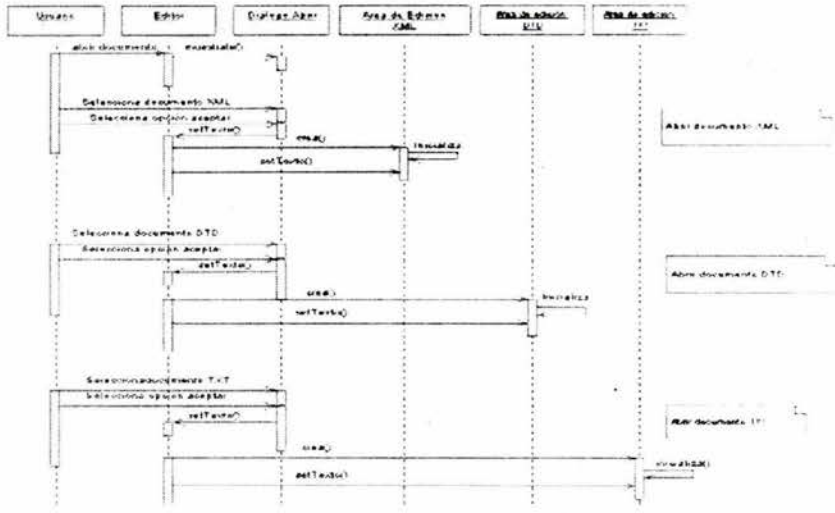


Figura 3.34: Diagrama de interacción abrir documento.

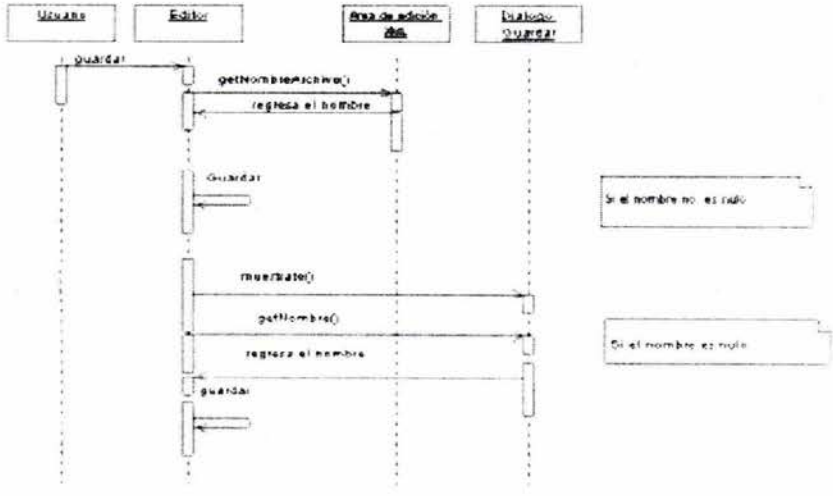


Figura 3.35: Diagrama de interacción guardar documento.

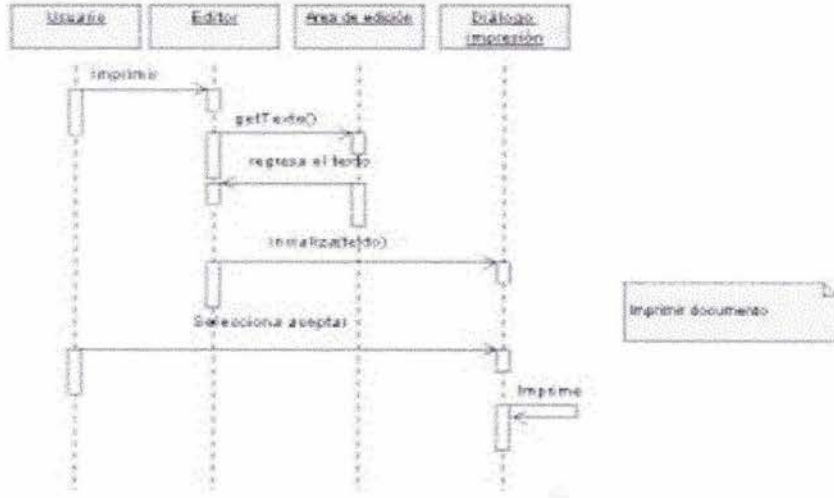


Figura 3.36: Diagrama de interacción imprimir documento.

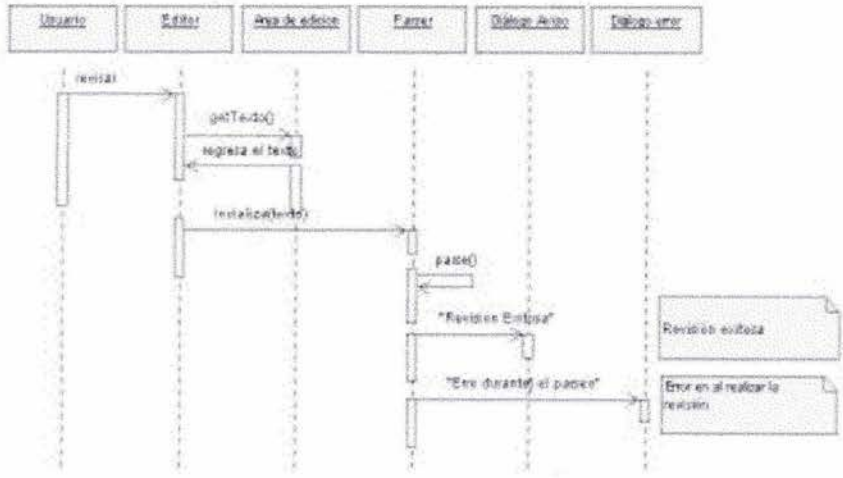


Figura 3.37: Diagrama de interacción revisar documento.

3.3.4 Diagramas de actividades.

En las Figuras 3.38, 3.39 y 3.40 se presentan los *Diagramas de Actividades* asociados con las operaciones *abrir, buscar e imprimir*. Este tipo de diagramas permiten conocer los posibles escenarios que pueden ocurrir un caso de uso.

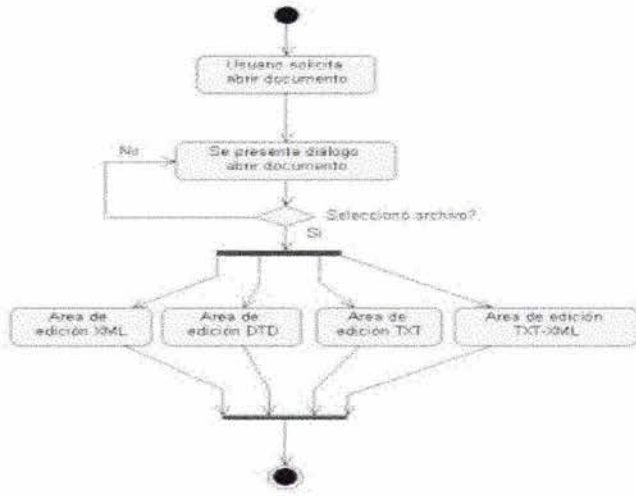


Figura 3.38: Diagrama de estado abrir documento.

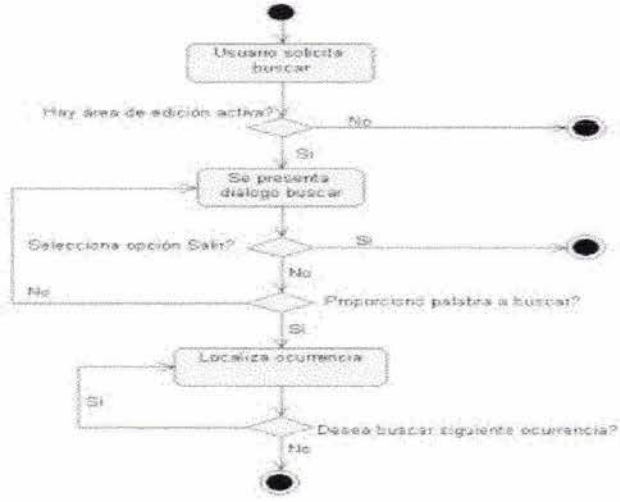


Figura 3.39: Diagrama de estado buscar.

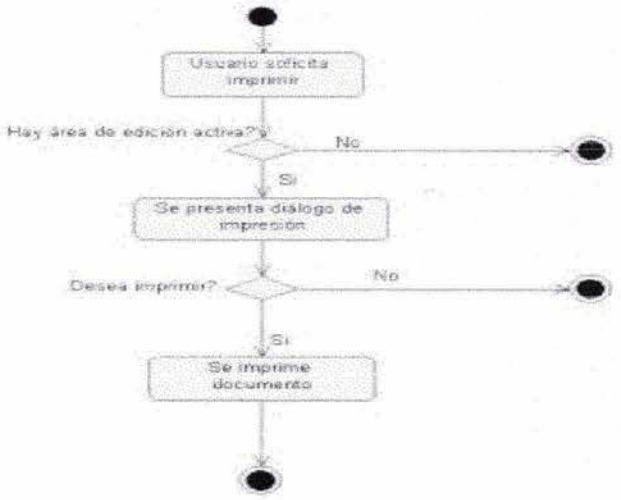


Figura 3.40: Diagrama de estado imprimir documento.

3.4 Diseño.

Una vez realizadas la etapas de *captura de requerimientos y análisis*, se determinó que es necesario tomar en cuenta las siguientes consideraciones.

3.4.1 Ambiente de implementación.

- El lenguaje de programación que se usará para la implementación de este proyecto es Java en su versión JSDK 1.4.
- La herramienta que se empleará para el desarrollo de la interfáz gráfica es Sun ONE Studio 4 CE.
- La herramienta para realizar la revisión de documentos XML es Java XML Pack (JAXP).

Para mayor información de estas herramientas ver el Anexo *C Herramientas.*

3.4.2 Restricciones.

Para esta versión se definieron las siguientes restricciones:

- La operación abrir documento esta restringida a documentos de edición libre.
- La estructura para la generación de documentos dirigidos por sintaxis, está definida en la Figura 2.3 *Componentes de un documento XML* del Capítulo 2.
- La generación de documentos XML Válidos dirigidos por sintaxis, esta limitada a la siguiente DTD:

```
<!ELEMENT Documento (Título, Autor+, Contenido, Bibliografía? >
<!ELEMENT Título (#PCDATA)¿ <!ELEMENT Autor (Nombre+, Apellido+, Dirección?)>
<!ELEMENT Nombre (#PCDATA)>
<!ELEMENT Apellido (#PCDATA)>
<!ELEMENT Dirección (#PCDATA)>
<!ELEMENT Contenido (Texto | Sección)+>
<!ELEMENT Texto (#PCDATA | Importante | Excluir | Puntualización)*>
<!ELEMENT Sección (Tema, Objetivo?, Contenido)*>
<!ELEMENT Tema (#PCDATA)>
<!ELEMENT Objetivo (#PCDATA)>
<!ELEMENT Importante (#PCDATA | Excluir | Puntualización)*>
<!ELEMENT Excluir (#PCDATA | Importante | Puntualización)*>
<!ELEMENT Puntualización (Punto+)>
<!ELEMENT Punto (#PCDATA | Importante | Excluir | Puntualización)*>
<!ELEMENT Bibliografía (#PCDATA)>
```

3.4.3 Arquitectura del sistema.

Como se definió en la etapa de análisis la arquitectura de la aplicación estará basada en el uso del patrón *Modelo-Vista-Controlador*, por lo que la arquitectura está formada de las siguientes capas:

Capa de modelo: Contiene los datos y la funcionalidad a emplear en la aplicación.

Capa de Vista: Se encarga de presentar la información obtenida de la capa de modelo al usuario.

Cada controlador: Contiene los elementos que interactúan directamente con el usuario.

3.4.4 Diagrama de instalación.

La herramienta lograda es distribuida en un archivo llamado *UnEditorParaDocumentosXML.tgz*, el cual contiene los archivos fuentes y los archivos ejecutables de la aplicación. La Figura 3.41 presenta como empleando alguna versión del JSDK o JRE puede ser ejecutada la aplicación en distintas plataformas.

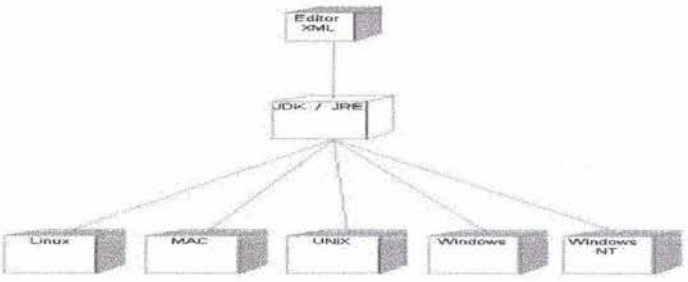


Figura 3.41: Diagrama de instalación.

3.5 Implementación.

Una vez definidas las estapas de captura de requerimientos, análisis y diseño, a continuación se presenta los puntos más importantes de la implementación del proyecto:

3.5.1 Procesamiento de un documento XML

El documento a editar se llama *Configuracion.xml*, contiene la configuración básica del editor y a este documento se accede cada vez que un usuario ejecuta la aplicación.

Los componentes que forman este documento son:

- **Fuente.**

  Nos permite definir las propiedades de la fuente a emplear para editar los documentos XML. Estas propiedades son: nombre, estilo y tamaño de la fuente.

- **Background.**

  Define el color que se empleará como fondo en las áreas de edición.

La estructura del documento *Configuración.xml* esta definida de la siguiente forma:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE CONFIGURACION [
  <!ELEMENT CONFIGURACION
            FUENTE,BACKGROUND)>
  <!ELEMENT FUENTE   EMPTY>
  <!ATTLIST FUENTE   nombre CDATA #REQUIRED
                     estilo   CDATA #REQUIRED
                     tamanio CDATA #REQUIRED >
  <!ELEMENT BACKGROUND  EMPTY>
  <!ATTLIST BACKGROUND red    CDATA #REQUIRED
                       green CDATA #REQUIRED
                       blue CDATA #REQUIRED >
]>

<!-- Archivo de configuracion -->
<CONFIGURACION>
   <FUENTE nombre="Monospaced" estilo="Plano" tamanio="4"/>
   <BACKGROUND red="255" green="255" blue="255"/>
   <DOCUMENTOETIQUETAS file="demo.xml"/>
   <ESTADOTXTAXML estado="1"/>
</CONFIGURACION>
```

Para acceder al contenido del documento *Configuracion.xml*, se creó una clase llamada *Configuracion.java*, el código completo de esta clase está definido en el Anexo D *Procesamiento de un documento XML*. En esta clase se definieron los siguientes métodos:

- set / get :

  Permiten accesar a los atributos que estan almacenados en el documento XML.

- String generaDocumentoDeConfiguracion()

  Permite generar una cadena con la información actual del sistema. Esta cadena será almacenada en el documento XML.

- almacenaConfiguracion()

  En esta función llama a la función generaDocumentoDeConfiguracion() para generar el texto del documento XML Válido y lo almacena en el archivo Configuracion.xml

- cargarConfiguracion()

  Esta función permite obtener la información del documento *Configuracion.xml*. Para realizar esta tarea crear una instancia del parser definido en JAXP, para realizar la revisión del documento obtenido.

  Si la revisión es correcta se genera un árbol con la estructura del documento, finalmente se hace un recorrido sobre el árbol obtenido y se obtienen los valores de cada elemento.

Durante todo este proceso, pueden ser generadas diversas excepciones por lo que en cada función se define *try... catch(...)*, lo permitirá capturar las excepciones generadas.

3.5.2 Desarrollo del editor para documentos XML.

Empleando la herramienta de Sun One Studio[4] se logra desarrollar la *interfaz* gráfica de la aplicación y los elementos definidos en el *diagrama de clases* presentado en la Figura 3.33. En la Sección *3.2.8 Protipo* se presentaron algunas de las pantallas desarrolladas con esta herramienta.

La *Pantalla principal* esta formada de los siguientes componentes gráficos:

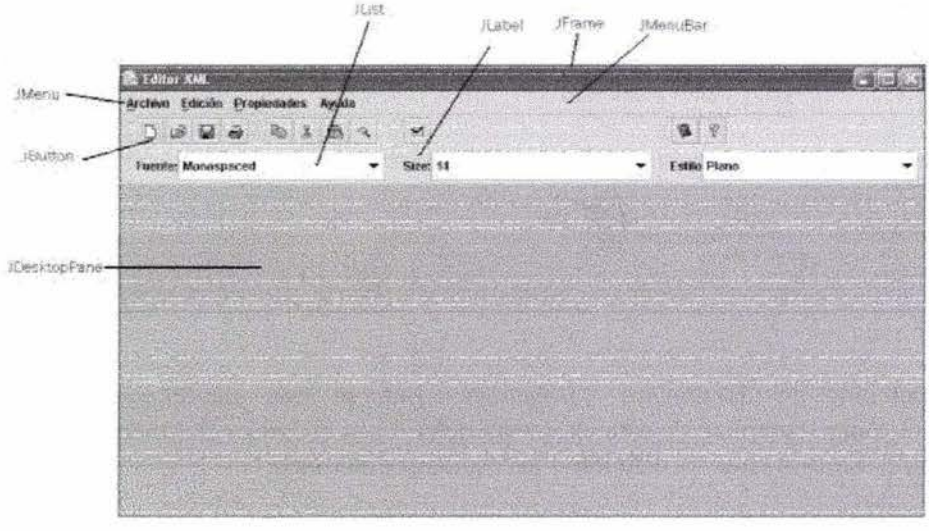


Figura 3.42: Componentes gráficos de la pantalla principal.

El desarrollo de las áreas para la *edición simple* y *dirigida por sintaxis* son presentadas en el Anexo E *Áreas de edición.*

[4]Ver Anexo C Herramientas

3.6 Resumen de la herramienta obtenida.

El producto final obtenido permite llevar a buen fin el objetivo planteado en esta tesis. todos los elementos que constituyen este producto se localizan en el CD anexo a este documento.

Los principales elementos que constituyen el producto final son:

- **Aplicación:**

  Esta formado por los archivos fuentes, los archivos ejecutables y el API de la aplicación.

- **Manual de usuario:**

  Documento que presenta cómo instalar y hacer uso de la funcionalidad que proporciona la aplicación. Este documento es presentado en el Anexo F *Manual de usuario.*

- **Documento de desarrollo:** Este documento.

Conclusiones.

XML es un estándar para representar e intercambiar información, permite separar la estructura, contenido y presentación de los documentos, logrando así manejar a cada uno de ellos de forma más efectiva. Es importante conocer los beneficios, limitaciones y la gran variedad de herramientas definidas para XML, para poder determinar si es más conveniente emplear esta tecnología o un formato propio para representar la información que se desea manejar.

El objetivo principal de este trabajo fue plantear una forma más fácil y eficiente de crear, editar y revisar documentos XML, esta herramienta se logra desarrollar empleando un conjunto de tecnologías como son: *Proceso unificado de desarrollo de software*, *UML*, *Java* y otras.

La herramienta lograda permite llevar a buen fin el objetivo planteado y está dirigida a personas que cuentan con poca experiencia en el manejo de documentos XML, las principales actividades que permite realizar esta herramienta son:

- Generar documentos XML bien formados.
- Generar documentos XML válidos.
- Generar documentos DTD.
- Generar documentos de texto.

Finalmente este *Editor para documentos XML*, puede ser valorado como una herramienta factible en uso para el desarrollo de proyectos futuros.

Apéndices.

Apéndice A

Proceso Unificado

El *Proceso Unificado* es un conjunto de actividades necesarias para transformar los requerimientos de un usuario en software[20], proporciona un marco genérico de procesos que pueden ser adaptados a diversos tipos de aplicaciones, áreas, organizaciones, niveles de competencia y tamaños de proyectos. Se caracteriza por permitir definir *quién* hace *qué*, *cuándo* y *cómo*.

El *Proceso Unificado* es un proceso guiado por los casos de uso, centrado en la arquitectura, iterativo e incremental, define el orden de las actividades a realizar, los productos que se deben generar, las tareas de cada participante y los criterios para poder monitorear el desarrollo de las actividades y productos. Emplea el *Lenguage de modelado unificado (UML)* para realizar la modelación de los sistemas de software.

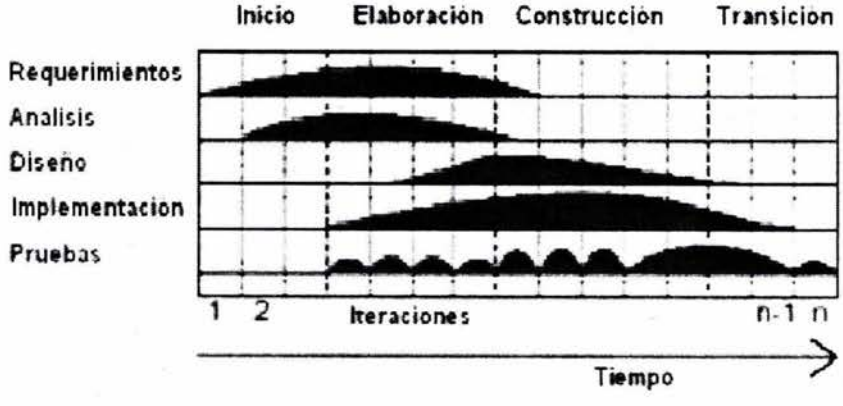


Figura A.1: Proceso Unificado.

La Figura A.1 presenta las cuatro *fases* que constituyen al Proceso Unificado *Inicio*, *Elaboración*, *Construcción y Transición*, cada una de estas fases tienen asociado un *flujo de trabajo* formado por *Captura de requerimientos*, *Análisis*, *Diseño*, *Implementación y Pruebas*. Cada elemento del *flujo de trabajo* está formado por un conjunto de actividades que producen como resultado unidades de información o *artefactos* como por ejemplo: modelos, código, documentos, etcétera.

Apéndice B

UML

UML es el acrónimo de *Unified Modeling Language*, es un lenguaje de modelación visual que se emplea para *especificar, visualizar, construir y documentar artefactos* de un sistema de software[19]. Tiene definido un conjunto de conceptos semánticos, notaciones y lineamientos. surge como un intento para unificar experiencias de la modelación de software.

Este lenguaje está diseñado para que pueda ser empleado en cualquier proceso de desarrollo de software, durante cualquier etapa de este y no importa el dominio de la aplicación. Tiene definido un conjunto de *diagramas* los cuales permiten generar distintas *vistas* de los aspectos *estáticos* y *dinámicos* del sistema. A continuación se presentan los principales diagramas de *UML*:

- **Diagrama de casos de uso.**

  Un diagrama de *casos de uso* permite identificar *quién hace qué en el sistema*, es decir permite modelar la interacción que hay entre un *actor* y el *sistema*. La Figura B.1 presenta el ejemplo de un diagrama de *casos de uso*, los principales componentes en este tipo de diagramas son:

  Caso de uso: representa alguna acción que el usuario puede desarrollar.

  Actores: puede ser un usuario o algún sistema.

  Asociaciones: permiten definir la participación de un actor en un caso de uso.

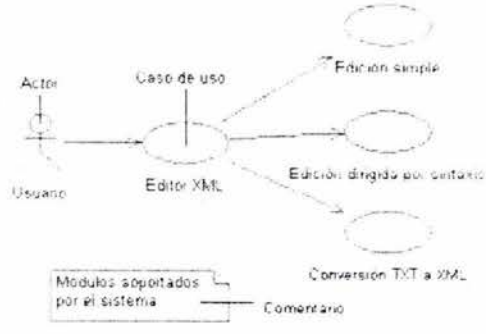


Figura B.1: Ejemplo de un diagrama de casos de uso.

- **Diagrama de clases.**

  Los *diagramas de clases* permiten modelar la estructura, contenido y relación que hay entre las clases que forman parte del sistema. Permiten definir la perspectiva *conceptual*, de especificación e *implementación* las cuales juegan un papel muy importante durante el desarrollo del software. La Figura B.2 presenta el ejemplo de un diagrama de clases, los principales componentes en este tipo de diagramas son:

  Clases: tiene definido un *nombre*, una lista de *atributos* y una lista *operaciones*.

  Asociaciones: definen una relación entre las clases, pueden ser dirigidas e indicar la multiplicidad que tienen asociada.

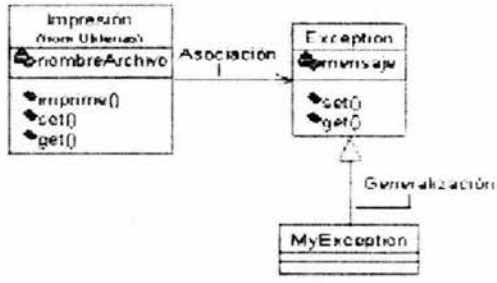


Figura B.2: Ejemplo de un diagrama de clases.

- **Diagrama de secuencia.**

  Un diagrama de *secuencia* permite modelar la interacción de los objetos que participan en un *caso de uso* durante un periodo de tiempo. La Figura B.3 presenta el ejemplo de un *Diagrama de secuencia*, los principales elementos que forman este tipo de diagramas son:

  Objetos: representan los elementos que están interactuando.

  Línea del tiempo: representa el tiempo en el cual un objeto está activo para poder enviar y recibir mensajes.

  Mensajes: son representados con flechas, tienen asociado la acción que se solicita y establecen un enlace entre las líneas del tiempo.

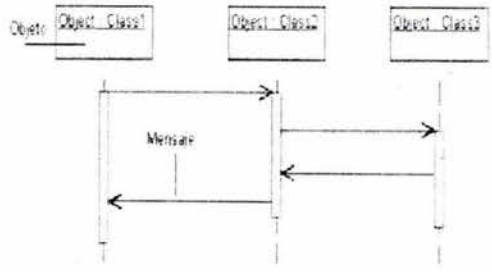


Figura B.3: Ejemplo de un diagrama de secuencia.

- **Diagrama de actividades.**

  Un diagrama de *actividades* permiten modelar los distintos escenarios que pueden ser presentados en un *caso de uso*. La Figura B.4 presenta el ejemplo de un diagrama de *actividades*, los principales elementos que forman este tipo de diagramas son:

  Actividades: representan las acciones a realizar.

  Asociaciones: indica que hay una dependencia secuencial.

  Fork: es empleado cuando distintas actividades pueden acurren al mismo tiempo sin importar el orden.

  Join: indica las actividades que deben estar completadas para pasar a la siguiente actividad.

  Branch: permite tomar decisiones para determinar la siguiente actividad a realizar.

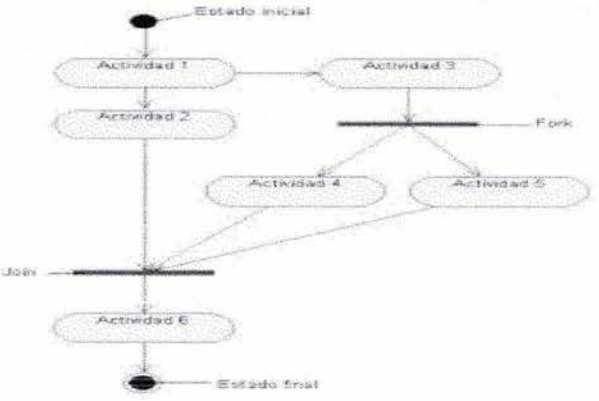


Figura B.4: Ejemplo de un diagrama de actividades.

- **Diagramas de estados.**

  Los diagramas de *estados* muestran un conjunto de acciones (*estados*) conectados por transiciones *(eventos)* que ocurren en un periodo de tiempo durante la vida de un objeto. Cuando un evento ocurre permite realizar el cambio de un estado a otro. La Figura B.5 presenta el ejemplo de un diagrama de *estados*, los principales elementos que forman este tipo de diagramas son:

  - *Estados:* los posibles estados que puede tener asociado un objeto.
  - *Transiciones:* representan los los eventos que puedan ocurrir.

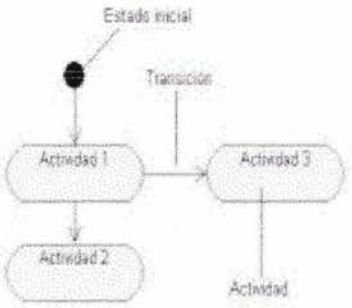


Figura B.5: Ejemplo de un diagrama de estados.

- **Diagrama de paquetes.**

  Los diagramas de *paquetes* proporcionan una forma de agrupar elementos como pueden ser: *clases*, *componentes de software*, *hardware*, *paquetes* y cualquier otro elemento que sea relevante. La Figura B.6 presenta el ejemplo de un diagrama de *paquetes*.

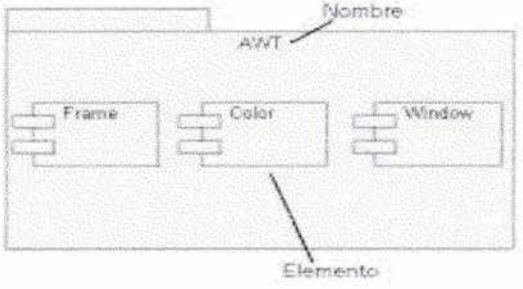


Figura B.6: Ejemplo de un diagrama de paquetes.

- **Diagramas de componentes.**

  Los diagramas de *componentes* permite modelar las *unidades de software* y la organización que tienen definida. La Figura B.7 presenta el ejemplo de un diagrama de *componentes*.

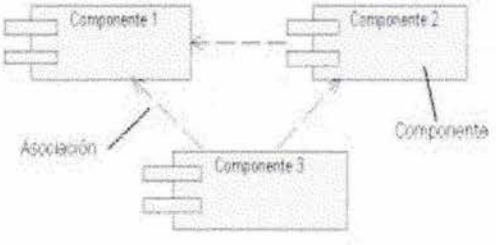


Figura B.7: Ejemplo de un diagrama de componentes.

- **Diagramas de instalación.**

  Un diagrama de *instalación* permite modelar la estructura de los componentes de hardware, los protocolo de comunicación que son empleados, la distribución y la asignación de los recursos con los que se cuenta. La Figura B.8 presenta el ejemplo de un diagrama de *componentes*. Los principales elementos de este tipo de diagramas son:

  - *Nodos:* representa una unidad de hardware.
  - *Asociaciones:* representa un enlace o un canal de comunicación.

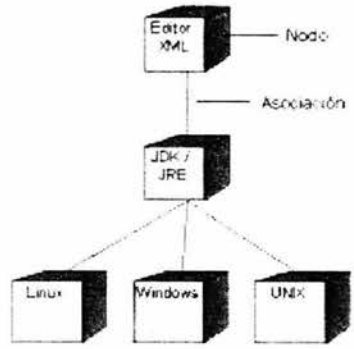


Figura B.8: Ejemplo de un diagrama de instalación.

Apéndice C

Herramientas

Sun ONE Studio 4 Community Edition.
Representa una de las herramientas más poderosas para el desarrollo de aplicaciones en Java, como son: aplicaciones web, gráficas y texto. Esta herramienta fue desarrollada por *Java Sun*, para uso académico es software libre, mientras que para uso comercial se tiene que pagar una licencia. El sitio donde puede obtenerse esta herramienta es: *http://www.java.sun.com*

JAXP
Esta herramienta forma parte del paquete *JavaTM XML Pack Spring 02 Dev Bundle*, desarrollada por *Java Sun.*

Java API for XML Processing (JAXP), representa uno de los mejores programas que realizan el parser de documentos XML, emplea *Xerces 2* como parser de XML, *Xalan* para el procesamiento de XSLT y para accesar a los datos del documento XML soporta SAX y DOM. El sitio donde puede obtenerse esta herramienta es:*http://www.java.sun.com*

JLEX y CUP
Ambas constituyen un herramienta muy poderosa para realizar el análisis sintáctico y reconocer expresiones, esta herramienta está basada en las utilerias lex y yacc de UNIX. El sitio donde puede adquirirse esta herramienta es:
http://www.cs.princeton.edu/ appel/modern/java/

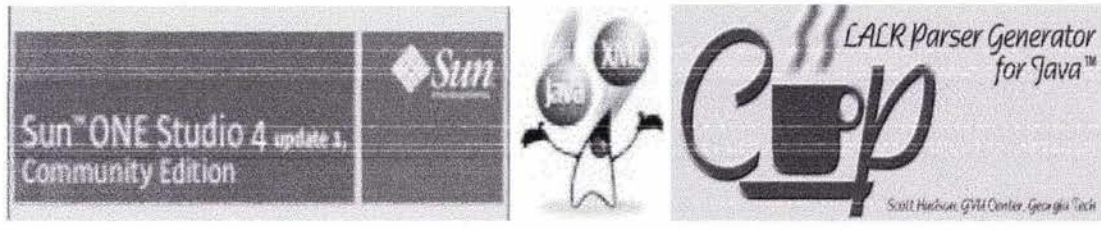


Figura C.1: Herramientas.

Apéndice D

Procesamiento de un documento XML

En esta sección se presenta el código completo de la clase *Configuracion.java* descrito en la Sección 3.5.1 del Capítulo 3. Este programa permite realizar las siguientes actividades:

- Generar un documento XML.
- Almacenar un documento XML.
- Cargar la información almacenada de un documento XML.
- Realizar la revisión de un documento XML.

```
1  //————————————————————————————|| Configuracion.java ||————————————————————————————//
2  //— Programo: Ruiz Salinas Oscar.
3  package src.PUtilerias;
4
5  import java.io.File;
6  import java.io.FileReader;
7  import java.io.BufferedReader;
8  import java.io.FileOutputStream;
9  import java.io.FileInputStream;
10 import javax.swing.*;
11 import java.util.*;
12 import java.awt.Font;
13 import java.awt.Color;
14 import src.PUtilerias.*;
15 import org.xml.sax.*;  // JAXP packages
16 import javax.xml.parsers.*;
17 import org.xml.sax.*;
18 import org.xml.sax.helpers.*;
19 import org.w3c.dom.*;
20 import javax.xml.parsers.SAXParserFactory;
21 import javax.xml.parsers.SAXParser;
22 import javax.xml.parsers.ParserConfigurationException;
23 import src.ParserXML.*;
24
25 /**
26    Esta clase contiene las propiedades de configuración de la aplicación,
27    tiene acceso al archivo src/PUtilerias/Configuracion.xml, en el cual
28    se almacenan y obtienen las propiedades de la aplicación.
29
```

```
30     Los componentes de esta clase son: fuente, nombre, estilo, tamanio y
31     background.
32     */
33  public class Configuracion{
34      Mensajes msj = new Mensajes(new JFrame());//Para mostar mensaje
35      /**Estas son las  propiedades de configuracion*/
36      String estilo="";
37      String nombre="";
38      int tamanio=0;
39      Color background=null;
40
41      public Configuracion(){ cargaConfiguracionDefault(); }
42
43      /**Estos  son los metodos para acceder a los atributos.*/
44      public void setEstilo(String  t){this.estilo=t;}
45      public void setNombre(String t){this.nombre=t;}
46      public void setTamanio(int t){this.tamanio=t;}
47      public void setBackground(Color c){this.background=c;}
48      public String getEstilo(){return this.estilo;}
49      public String getNombre(){return this.nombre;}
50      public int getTamanio(){return this.tamanio;}
51      public Color getBackground(){ return this.background;}
52
53      /**Esta función nos permite generar el documento  XML de la configuración*/
54      String generaDocumentoDeConfiguracion() {
55          String prologo= "<?xml version=\"1.0\" encoding=\"iso-8859-1\"?>\n";
56
57          String dtd="<!DOCTYPE CONFIGURACION [ \n"+
58                  "\n\t<!ELEMENT CONFIGURACION  (FUENTE,BACKGROUND)> \n"+
59                  "\t<!ELEMENT FUENTE  EMPTY>     \n"+
60                  "\t<!ATTLIST FUENTE           \n\t"+ "\t nombre CDATA #REQUIRED    \n\t"+
61                  "\t estilo  CDATA #REQUIRED   \n\t"+ "\t tamanio CDATA #REQUIRED > \n"+
62                  "\n\t<!ELEMENT BACKGROUND  EMPTY> \n"+
63                  "\t<!ATTLIST BACKGROUND   \n\t"+ "\t red   CDATA #REQUIRED \n\t"+
64                  "\t green CDATA #REQUIRED \n\t"+ "\t blue CDATA #REQUIRED >\n"+ "]> \n";
65
66          String comentario="<!-- Archivo de configuracion -->\n";
67          //Generamos el documento
68          String xml="<CONFIGURACION>\n";
69          xml+="\t<FUENTE nombre=\""+this.nombre+"\""+
70                  " estilo=\""+this.estilo+"\""+
71                  " tamanio=\""+this.tamanio+"\"/> \n";
72          xml+="\t<BACKGROUND red=\""+this.background.getRed()+"\""+
73                  " green=\""+this.background.getGreen()+"\""+
74                  " blue=\"" +this.background.getBlue()+"\"/> \n";
75          xml+="</CONFIGURACION>";
76          String documentoXML= prologo+"\n"+dtd+"\n"+comentario+"\n"+xml;
77
78          return documentoXML;
79      }
80
81      /**Esta función permite almacenar los valores de la configuración en el
82       archivo src/PUtilerias/Configuracion.xml*/
83      public void almacenaConfiguracion(){
84          String rta = getClass().getResource("/src/PFrames/imagenes/open.gif").toString();
85          StringTokenizer str = new StringTokenizer(rta,"/");
86          String path ="";
87          str.nextToken();
88
89          int total= str.countTokens()-3;
90          for(int i=0; i< total; i++){
```

```
91              path+=str.nextToken()+"/";
92          }
93
94          path+="PUtilerias/Configuracion.xml";
95
96          //DELIMITADOR
97          path.replace('/','\\');
98          System.out.println(path);
99          try{
100             File file =new File(path);
101             FileOutputStream fun = new FileOutputStream(file);
102             String text = generaDocumentoDeConfiguracion();
103             int textsize= text.length();
104             byte data[];
105             data =new  byte[textsize];
106             text.getBytes(0,textsize,data,0);
107             fun.write(data);
108             fun.close();
109             System.out.println("Documento almacenado\n"+ text);
110             msj.aviso("Aviso","Documento de configuracion almacenado");
111         }
112         catch(Exception e2){
113             msj.error("Error","No se pudo almacenar el archivo de configuracion");
114             System.out.println("Error al almacenar configuracion!");
115         }
116     }
117
118     /**Esta función permite obtener los valores de configuración almacenados en
119        el archivo src/PUtilerias/Configuracion.xml*/
120     public void cargaConfiguracion(){
121         String rta = getClass().getResource("/src/PFrames/imagenes/open.gif").toString();
122         StringTokenizer str = new StringTokenizer(rta,"/");
123         String path ="";
124         str.nextToken();
125
126         int total= str.countTokens()-3;
127         for(int i=0; i< total; i++){
128             path+=str.nextToken()+"/";
129         }
130
131         path+="PUtilerias/Configuracion.xml";
132         //DELIMITADOR
133         path.replace('/','\\');
134         System.out.println(path);
135
136         try{
137             File file= new File(path);
138             ParserConfiguracion demo = new ParserConfiguracion();
139             demo.setEstatusMensaje(false);
140             demo.procesa(file);
141             Document doc = demo.getDocumento();
142
143             //Obtenemos a cada uno de los elementos.....
144             if(doc != null){
145                 System.out.println("Documento no nulo .... ");
146                 Element elem = doc.getDocumentElement();
147
148                 //Obtenemos FUENTE
149                 NodeList nl = elem.getElementsByTagName("FUENTE");
150                 Element item =(Element) nl.item(0);
151                 if(item != null){
```

```
152                     this.nombre  = item.getAttribute("nombre");
153                     this.estilo  =  item.getAttribute("estilo");
154                     this.tamanio =  (new Integer(item.getAttribute("tamanio"))).intValue();
155                     System.out.println("Atributos > "+ nombre +" "+estilo+
156                                        " "+this.tamanio);
157                     }
158                     else{
159                         this.msj.error("Error","El documento de configuracion"+
160                                        " no tiene el formato correcto. \n"+
161                                        " se cargarán los valores por default");
162                         cargaConfiguracionDefault();
163                     }
164
165                 //Obtenemos BACKGROUND
166                 nl = elem.getElementsByTagName("BACKGROUND");
167                 item =(Element) nl.item(0);
168                 if(item != null){
169                     int red   = (new Integer(item.getAttribute("red"))).intValue();
170                     int green = (new Integer(item.getAttribute("green"))).intValue();
171                     int blue  = (new Integer(item.getAttribute("blue"))).intValue();
172                     System.out.println("BACKGROUND > "+ red+" "+green +" "+blue);
173                     this.background= new Color(red,green,blue);
174                 }
175                 else{
176                     this.msj.error("Error","El documento de configuracion"+
177                                    " no tiene el formato correcto. \n"+
178                                    " se cargarán los valores por default");
179                     cargaConfiguracionDefault();
180                 }
181             }
182         }
183         catch(Exception e){
184             this.msj.error("Error","El documento no tiene el formato correcto.");
185             e.printStackTrace();
186             cargaConfiguracionDefault();
187         }
188     }
189
190     /** Esta función permite definir una configuración de default */
191     void cargaConfiguracionDefault(){
192         this.nombre="Dialog";
193         this.tamanio=10;
194         this.estilo="Plano";
195         this.background = new Color(255,255,255);
196     }
197
198
199     public static void main(String args[]){
200         Configuracion c = new Configuracion();
201         Color cl = new Color(255,255,255);
202         c.setBackground(cl);
203         c.almacenaConfiguracion();
204         c.cargaConfiguracion();
205     }
206 }
207 //————————————————————————|| Fin de Archivo ||————————————————————————//
```

Apéndice E

Áreas de edición

Un *área de edición* representa un área de trabajo en la cual se puede generar y manipular distintos tipos de documentos, Java proporciona un conjunto de componentes como son: *JTextArea, JEditorPane, JTextPane, etcétera*, los cuales tienen definido un conjunto de atributos e implementan la funcionalidad de un área de edición.

Para el desarrollo de esta tesis se implementaron 2 componentes uno para la *edición simple* y otro para la *edición dirigida por sintaxis*, estos elementos extienden la funcionalidad de los componentes definidos en Java y proporcionan información necesaria sobre el número de líneas que constituyen al documento y el número de línea en la que se está trabajando.

La Figura E.1 (a) presenta el ejemplo de una área de edición (*JTextArea*) definida por Java, mientras que la Figura E.1 (b) presenta el ejemplo de unos de los componentes que se desarrollaron para la edición simple de documentos.

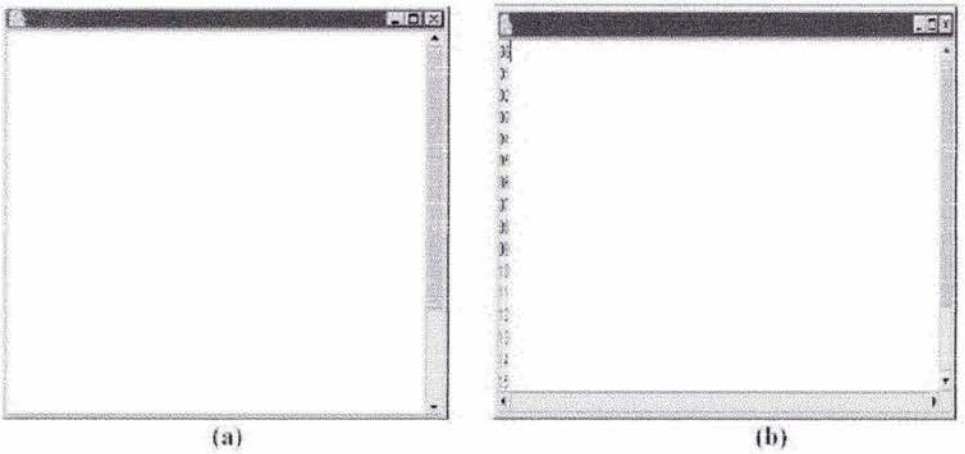


Figura E.1: Áreas de edición: a) Proporcionado por Java b) Componente desarrollado.

- **Edición simple.**

  Para realizar la edición de documentos que no son dirigidos por sintaxis, se generó el componente *InternalTXT*, la Figura E.2 presenta los componentes gráficos que se emplearon para realizar la implementación de este componente.

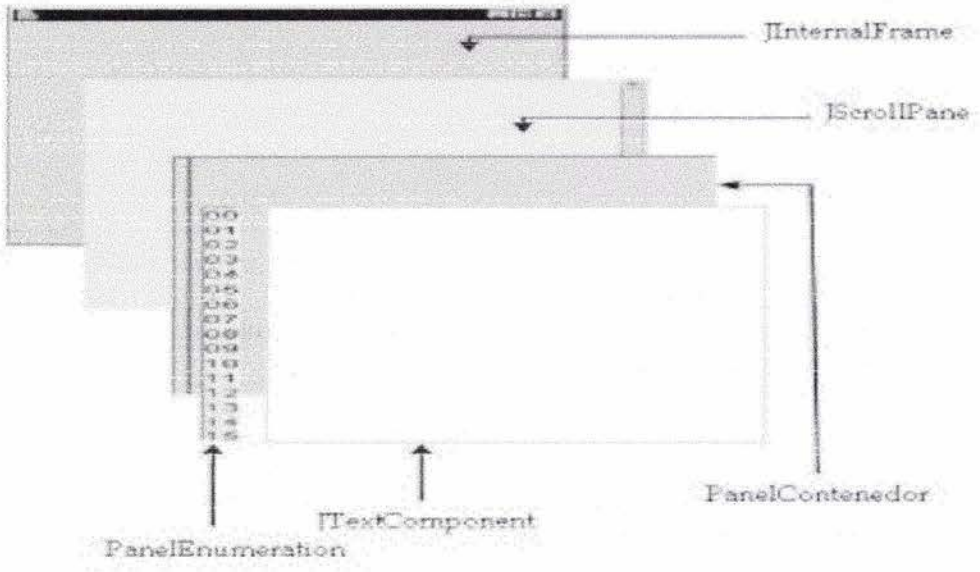


Figura E.2: Áreas de edición simple.

InternalTXT emplea un *JTextPane* como área de edición y cuenta con un componente llamado *PanelEnumeration* el cual permite llevar el control sobre el número de líneas que constituyen al documento, este componente se activa con los eventos generados por el área de edición.

- **Edición dirigida por sintaxis.**

  Para la edición de documentos dirigidos por sintaxis se desarrollaron los componentes *InternalXMLBF, InternalXMLV e InternalDTD* cada uno de ellos permite desarrollar documentos específicos. La Figura E.3 presenta la estructura de los elementos desarrollados y los componente gráficos empleados.

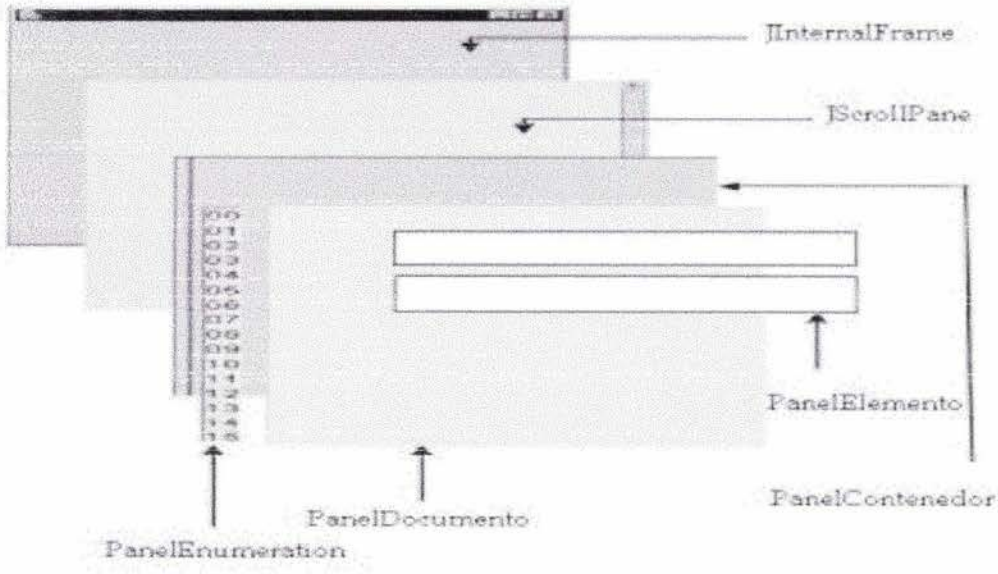


Figura E.3: Áreas de edición dirigida por sintáxis.

Cada uno de estos elementos cuenta con un *PanelEnumeration* para tener el control del número de líneas que constituyen al documento y el área de edición está formada por el componente *PanelDocumento* el cual contiene uno o más componentes del tipo *PanelElemento.*

Apéndice F

Manual de usuario

Un editor para documentos XML es un programa desarrollado en Java, el cual tiene como objetivo facilitar la creación, edición y almacenamiento de documentos XML. En el CD anexo a este documento se tiene la versión compilada y los archivos fuentes de la aplicación.

Requerimientos:

Para poder ejecutar la aplicación se requiere tener instalado alguno de los siguientes programas:

- Java 2 Software Development Kit (J2SDK) en su versión 1.3 o superior.
- Java Runtime Enviroment (JRE) 1.3 o superior.

En cuanto a hardware se requiere almenos un equipo Pentium con 64MB de memoria RAM.

Instalación:

Para instalar la versión compilada basta con copiar la carpeta *UnEditorParaDocumentosXML/* a la unidad donde se desea instalar la aplicación. Mientras que para realizar la instalación empleando los archivos fuentes se deberán ejecutar las siguientes actividades:

1. Copiar el archivo *UnEditorParaDocumentosXML.tgz*, a la unidad donde desea instalar la aplicación.

2. Descomprimir el archivo.

   Linux:

   ```
   $ tar zxvf UnEditorParaDocumentosXML.tgz
   ```

   Windows: se emplea Winzip para descomprimir el archivo.

   Un vez realizada la actividad anterior se obtiene la estructura de archivos y directorios definida en la Tabla F.

3. Compilar los archivos.

Linux:

```
$cd EditorParaDocumentosXML
$make
```

Windows:

```
$c:\ cd EditorParaDocumentosXML
$c:\EditorParaDocumentosXML\>compila.bat
```

docs/	Contiene la API y el manual de la aplicación.
ejemplos/	Ejemplos de archivos que pueden ser generados con la herramienta.
java_cup/	Contiene una aplicación desarrollada en Java, la cual permite realizar el reconoimiento de expresiones.
JLex/	Contiene una aplicación desarrollada en Java, la cual permite realizar el reconocimiento de tokens.
src/	Contiene los archivos fuentes de la aplicación.
compila.bat:	Permite realizar la compilación de los archivos fuente desde Windows.
Makefile:	Permite realizar la compilación de los archivos fuente y la ejecución de la aplicación desde Linux.
runEditor.bat:	Permite realizar la ejecución de la aplicación en Windows.
README:	Proporciona la información necesaria para realizar la instalación y ejecución de la aplicación.

Tabla F.1: Estructura de archivos

Uso de la aplicación:

Para inciar la aplicación se deberá ejecutar la siguiente instrucción desde la línea de comandos:

Linux:

```
$make  run
```

Windows:

```
c:\EditorParaDocumentosXML\>runEditor.bat
```

La Figura F.1 muestra la pantalla obtenida después de ejecutar la insrucción desde la línea de comandos. Esta pantalla esta compuesta principalmente de una barra de herramientas, una barra de herramientas gráfica y un escritorio.

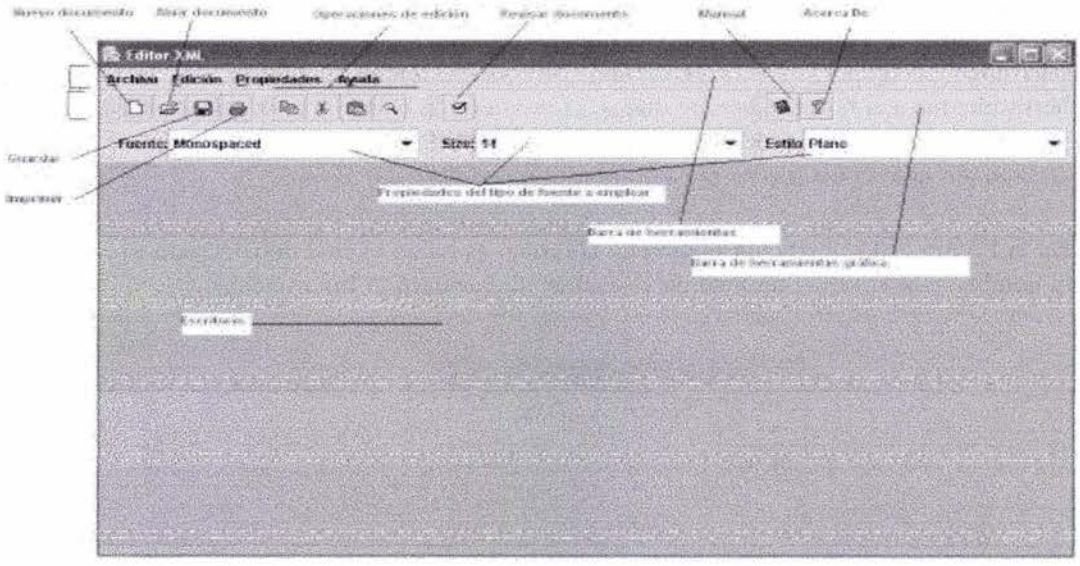


Figura F.1: Pantalla principal.

Nuevo documento:

Para generar un nuevo documento seleccione la opción *Archivo/Nuevo* en la barra de herramientas o presione el botón *Nuevo documento* localizado en la barra de herramientas gráfica.

Una vez realizada esta actividad se presenta el diálogo *Nuevo documento*, el cual permite seleccionar el tipo de documento a generar. En la Figura F.2 se presenta la figura del diálogo *Nuevo documento*, una vez seleccionado el tipo de documento seleccione la opción *Aceptar* y el editor presentará en el escritorio el área de edición del documento solicitado.

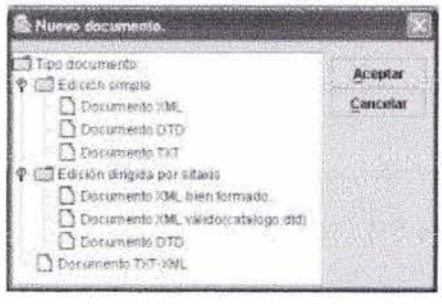


Figura F.2: Nuevo documento.

Guardar documento:

Para realizar el almacenamiento del documento que se está editando actualmente, seleccione la opción *Archivo/Guardar*, *Archivo/Guardar Como* en la barra de herramientas o el botón *Guardar* que se presenta en la barra de herramientas gráfica. Si el documento tiene asociado un nombre, la información será almacenada en este archivo, en caso contrario se presentará el diálogo *Guardar como.*

En la Figura F.3 se presenta el diálogo *Guardar como* en el cual deberá proporcionar el nombre y la ruta del archivo en el cual se almacenará la información.

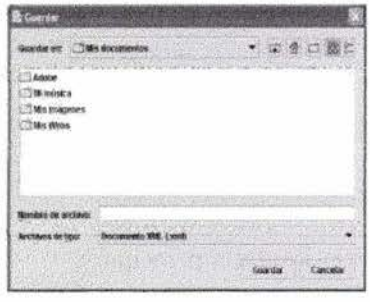


Figura F.3: Diálogo Guardar como.

Abrir:

Para abrir un documento, seleccione la opción *Archivo/Abrir* o seleccione el botón *Aabrir* de la barra de herramientas. Una vez realizada esta actividad se presenta el diálogo *Abrir documento.* En la Figura F.4 se presenta el diálogo *Abrir documento* en el cual de deberá proporcionar la ruta del archivo a abrir.

Figura F.4: Abrir documento.

Imprimir:

Para realizar la impresión de un documento debe seleccionar la opción *Archivo/Imprimir* de la barra de herramientas o el botón *Imprimir* que se localiza en la barra de herramientas gráfica, una vez realizada esta acción se presenta el diálogo *Imprimir.* En la figura F.5 se presenta el diálogo *Imprimir* en el cual deberá proporcionar las características de la impresión a realizar y el dispositivo a emplear.

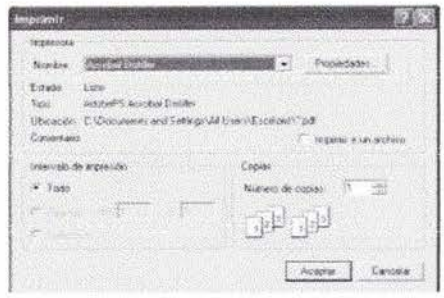


Figura F.5: Diálogo Imprimir documento.

Revisión:

Está operación solo está disponible para los documentos DTD y XML. Para realizar esta operación seleccione la opción *Edición/Revisar* de la barra de herramientas o el botón *Revisar* en la barra de herramientas gráfica. Una vez realizada esta acción el sistema presentará el diálogo *Resultado de revisión.*

La figura F.6 presenta el diálogo *Resultado de revisión*, el cual le indicará si la revisión del documento fue correcta o el error que se encontró durante la revisión.

Figura F.6: Diálogo Revisar documento.

Operaciones de edición:

Las operaciones de edición pueden aplicarse seleccionado los botones del menu *Edición* en la barra de herramientas o los botones de la barra de herramientas gráfica.

Figura F.7: Operaciones de edición.

Manual:

Para acceder al manual de usuario seleccione la opción *Ayuda/Manual* o el botón *Manual* de la barra de herramientas gráfica. Una vez realizada esta acción se presentará el dialgo Manual. La Figura F.8 presenta el diálogo *Manual.*

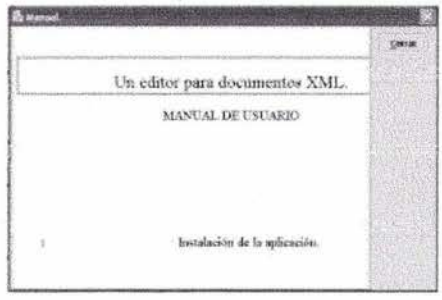


Figura F.8: Diálogo Manual de usuario.

Acerca De:

Para acceder a la información acerca de seleccione la opción *Ayuda/Acerca de* o el botón *Acerca de* en la barra de herramientas gráfica. La Figura F.9 presenta el diálogo obtenido una vez realizada esta acción.

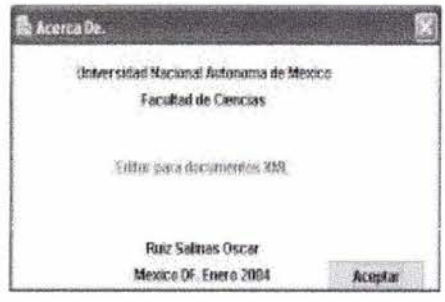


Figura F.9: Diálogo Acerca de.

Salir:

Para salir de la aplicación seleccione la opción *Archivo/Salir* o el botón *Salir* el cual se localiza en la esquina superior derecha de la pantalla de la aplicación.

Figura F.10: Diálogo Salir.

Glosario.

ASCII : Es un estándar para representar texto, se caracteriza por emplear 8 bits para representar los caracteres.

DTD: Document Type Definition. Representa un documento de definición de datos el contiene las reglas válidas para construir documentos XML.

HTML: (Hypertext Markup Language) es un lenguaje de marcado que permite hacer la representación de documentos para que puedan ser accesados a través de la red.

MVC: (Model-View-Controller) es un patrón de arquitectura, se caracteriza por dividir a una aplicación en 3 componentes: modelo, vista y controlador. Esto permite que la funcionalidad de la aplicación sea independiente de la interfaz de usuario logrando así que un cambio en la interfaz no impacte de forma grave a la funcionalidad interna.

Patrón de arquitectura: representa un patrón con un alto nivel de abstracción. Permite especificar la estrucutra fundamental de una aplicación.

RTF: RTF (Rich Text Format) representa un formato en cual emplea código ASCII para representar marcas.

SGML: (Standar Generalized Markup Language) es un metalenguaje para la definición y estructura de diferenes tipos de documentos.

UNICODE: Es un estándar para representar texto, emplea una representación de 16 bits para representar los caracteres.

Usuario: representa a las personas que pueden emplear el sistema para generar un documento.

XML: es el acrónimo de e*X*tensible *M*arkup *L*anguaje
es un lenguaje estándar para representar información.

Bibliografía

[1] An Introduction to Text Processing
Peter D. Smith.
Editorial The MIT Press.
London England, 1990.

[2] ACM Computing Surveys
Volumen 14, Número 3.
Palo Alto CA, Septiembre 1982.
Editorial: Computing Surveys Asociate
Artículo: *Interactive Editing Systems Part I.*
Artículo: *Interactive Editing Systems Part II.*
Norman Meyrowitz.
Andries van Dam.

[3] The society of text.
Edward Barrett.
Editorial: The MIT Press.
London Inglaterra,1993.
Artículo: *Online Information, Hypermedia, and the Idea of Literacy.*
Philip Rubens.

[4] The evaluation of text editors: methodology and empirical results.
Roberts, T. L. & Moran, T. P.
Editorial: Comm. ACM, 26 (4), 265-283.
E.U. 1983.

[5] Lecture Notes in Computer Science 1614
Third International Conference, Visual 1999.
Ed. Springer.
Amsterdam, 1999
Articulo: *Visual Information and Informatic Systems.*
Jou-Hwee Lim.

[6] Lecture Notes in Artificial Intelligence 546
Editorial: Springer
Berlín, 1991 **Artículo:** *Text Understanding in LILOG*
Jochen Dörre Rollnger

[7] UNIX Text Processing
Dale Dougherty y Tim O'Reilly
Editorial: Hayden Books
E.U. 1997.

[8] Computer Text Recognition and Error Correction
Sargur N. Srihari
Editorial: IEEE Computer Society Press
1984.

Artículo: Computer Programs for Detecting and Correction Spelling Errors
James L. Peterson.

Artículo: Automaic Recognition of Print and Script
Autor: Leon D. Harmon.

[9] Text and Context: Document Processing and Storage
Susan Jones.
Editorial: Springer-Verlang.
Alemania, 1991

[10] Visual Language Theory
Kim. Marriot y Bernd Meyer
Ed. Springer
New York, 1998

[11] JDC Teach Tips : Introduction to XML.
Junio 27 2002.
http://developer.java.sun.com

[12] Wold Wide Web Consortium.
http://www.w3.org
http://www.w3.org/TR/1998/REC-xml-19980210.

[13] Extensible markup languaje(XML).
http://www.xml.com/xml/pub/

[14] Inside XML DTD's
St. Laurent and Nigsjar
Mac Graw Hill.
E.U. 1999

[15] Applied XML, a Toolkit for Programmers
Alex Ceponkus and Faraz Hoodbhoy
Wiley Computer Publishing
E.U. 1999

[16] XML: A Primer
Simon St. Laurent
Hungry Minds, Inc.
Enero 1998.

[17] Manual de XML.
Charles F. GoldFarb, Paul Prescod.
Ed. Prentice Hall
Junio 1999

[18] Presenting XML.
Richard Ligth.
Macmillan Computer Publishing, 1997.

[19] The Unified Modeling Language. User Guide
Grady Booch, James Rumbaugh, Ivan Jacobson.
Addison-Wesley, USA, primera edición, 1998.

[20] The Unified Software Process Development
Grady Booch, James Rumbaugh, Ivan Jacobson.
Addison-Wesley, USA, primera edición, 1999.