VNIVERSIDAD NACIONAL
AVFNOMA DE
MEXICO

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

"ARQUITECTURA GENÉRICA PARA ALMACENAR DOCUMENTOS XML EN UNA BASE DE DATOS RELACIONAL"

T E S I S

QUE PARA OBTENER EL GRADO DE:

MAESTRO EN INGENIERÍA
(COMPUTACIÓN)

P R E S E N T A:

KAREN ROCIO TRILLO LARES

DIRECTOR DE TESIS: DRA. AMPARO LÓPEZ GAONA

México, D.F. 2005.

A mi Padres que forjaron mi carácter para salir adelante pese a las adversidades y tropiezos de la vida.

Agradezco el apoyo brindado en el desarrollo de esta tesis a la Dra. Amparo López Gaona.

Un especial agradecimiento a David, por enseñarme a creer en mí, a luchar por lo que se quiere y a no ponerme límites.

A los grandes amigos que tuve la fortuna de conocer y con los que pase muy buenos momentos durante la realización de esta maestría.

Y a todos mis seres queridos que me acompañaron a lo largo de esta gran etapa de mi vida.

Índice general

Índice de figuras

Índice de cuadros

Introducción

Antecedentes

La sociedad actual es la sociedad de la información y del conocimiento; la información es poder y los datos al convertirse en información pueden servir para la toma de decisiones de negocio.

Internet está provocando grandes cambios en las organizaciones y su gestión. Las organizaciones tienen que migrar a la Web, dando como resultado la aparición de nuevos tipos y formatos de datos para la publicación de información, tales como HTML o XML.

A finales del siglo XX aparece XML, un lenguaje de marcado que ofrece un formato para la descripción de datos estructurados, creado por el W3C (World Wide Web Consortium) y que sirve como estándar para la representación e intercambio de información en el entorno Web. Los datos XML no sólo se intercambian, sino también se procesan y se almacenan. Los documentos XML se pueden almacenar en bases de datos nativas XML o en bases de datos relacionales. Consecuentemente, el problema principal llega a ser almacenar datos XML con eficacia y eficiencia.

El mundo de las bases de datos también se ha visto afectado por los cambios. Por un lado, las tradicionales bases de datos relacionales siguen almacenando la mayor parte de información de una organización, pero las últimas versiones de los Sistemas Manejadores de Bases de Datos, se han adaptado a las nuevas necesidades, por ejemplo tratan y almacenan información en formato XML.

La importancia de XML en el medio Web por un lado, y por el otro lado la gran cantidad de información que continua siendo almacenada en las bases de datos relacionales, llevan a estudiar cómo tratar de forma conjunta datos procedentes de ambos entornos.

Problemática

La problemática que se presenta es ¿cómo almacenar dentro de una base de datos relacional documentos XML? para ello se debe definir una arquitectura que permita almacenar, verificar, consultar y manipular la información contenida en los documentos XML y sus respectivos esquemas.

Para el almacenamiento de documentos XML existen las bases de datos nativas XML, pero estas todavía forman un pequeño porcentaje de todas las bases de datos instaladas, además de que les falta implementar la seguridad, el respaldo, la recuperación y las herramientas administrativas que proporcionan las bases de datos relacionales existentes. Por otro lado las bases de datos relacionales existentes son infraestructura crítica en la mayoría de las organizaciones.

Otro punto que debe ser tratado es que antes de almacenar los documentos XML dentro de la base de datos, se deben analizar para verificar que sean documentos bien formados y en el caso de contener una DTD o un Esquema XML también se debe verificar que sean válidos y de esta forma garantizar la integridad en la información.

Objetivo

Crear una aplicación que basada en una arquitectura genérica permita almacenar y manipular documentos XML (válidos o bien formados, según sea el caso) dentro de una base de datos relacional.

El objetivo de esta tesis es realizar una aplicación para almacenar y manipular documentos XML dentro de una base de datos relacional. Entre las caracterísicas importantes de esta aplicación es que mapea la estructura del documento XML en vez de almacenar el documento completo en la base de datos relacional, además almacena cualquier documento XML, ya sea que contenga una DTD o un Esquema XML o incluso que no contenga ninguna de las 2 siempre y cuando sea un documento bien formado. Por último esta aplicación fue creada con software libre cuya arquitectura propuesta no crea ningun lazo con alguna base de datos comercial.

Estructura de la Tesis

La disertación de la tesis está conformada en siete capítulos, a través de los cuales se presenta el contexto del problema y la solución a la que se llegó. Los capítulos 2 y 3 exponen el panorama teórico alrededor del mundo XML mientras que el capítulo 4 expone la relación entre XML y las bases de datos.

En el capítulo 5, se presenta una arquitectura genérica para almacenar documentos XML dentro de una base de datos relacional; así como también los requerimientos, el análisis y el diseño de la aplicación realizada.

En el capítulo 6, se muestra los detalles de la implementación de la aplicación y mediante un ejemplo se observa el funcionamiento de ésta. Para llegar, finalmente, al capítulo 7 donde se encuentran las conclusiones de este trabajo.

Capítulo 1

XML Extensible Markup Language

El lenguaje de marcado (*markup language*), también denominado lenguaje de anotaciones o de etiquetas[1], se define como un conjunto de reglas para estructurar y dar formato a un documento electrónico. Se suele utilizar etiquetas para definir el inicio y el final de un elemento: un párrafo, un título, un elemento subrayado, etc. Los lenguajes de marcado más utilizados son HTML y XML, ambos basados en SGML (Standard Generalized Markup Language)[6].

HTML (Hypertex Markup Language) se ha convertido en un lenguaje de marcado de inmensa popularidad durante los últimos años. Éste se ha encontrado con sus propias limitaciones, que algunas de ellas se han querido subsanar con la incrustación de scripts, javascripts, HTML dinámico, hojas de estilo en cascada (CSS). Todo esto es insuficiente para crear una arquitectura abierta de tipo cliente/servidor, por lo que el W3C (Word Wide Web Consortium), organismo que vela por el desarrollo de la Word Wide Web, se replanteó crear un nuevo estándar llamado XML (eXtensible Markup Language), que parte de las amplias especificaciones del SGML.

XML fue desarrollándose por el Grupo de Trabajo XML desde 1996 (en estos primeros años llamado SGML Editorial Review Board). La especificación XML 1.0 fue ratificada por el W3C el 10 de febrero de 1998, e interpretada como un sistema para definir, validar y compartir formatos de documentos en la Web.

XML es un lenguaje de marcado que ofrece un formato para la descripción de datos estructurados. Esto facilita tener declaraciones de contenido más precisas y resultados de búsquedas más significativos en varias plataformas.

[1] Etiqueta: es una marca; es texto que se caracteriza por estar delimitado por paréntesis angulares < >.

XML es una tecnología que tiene a su alrededor otras tecnologías que la complementan y la hacen mucho más grande y con posibilidades mucho mayores. XML junto con todas las tecnologías relacionadas, representa una manera distinta de hacer las cosas, cuya principal novedad consiste en permitir compartir los datos con los que se trabaja a todos los niveles, por todas las aplicaciones y soportes. Así pues, XML juega un papel importantísimo en este mundo actual, que tiende a la globalización y la compatibilidad entre los sistemas, ya que es la tecnología que permitirá compartir la información de una manera segura, fiable y fácil. Además, XML permite al programador dedicar sus esfuerzos a tareas más importantes cuando trabaja con los datos, ya que algunas tareas tediosas como la validación de los datos o el recorrido de las estructuras corre a cargo de este lenguaje y está especificado por el estándar, de modo que el programador no tiene que preocuparse por ello.

1.1. Historia de XML

XML proviene de un lenguaje que creó IBM por los años 70. El lenguaje de IBM se llama GML (General Markup Language) y surgió por la necesidad que tenían en la empresa de almacenar grandes canti0dades de información de temas diversos.

Este lenguaje gustó mucho a la gente de ISO, una entidad que se encarga de normalizar los procesos del mundo actual, de modo que para 1986 trabajaron para normalizar el lenguaje creando SGML, que no era más que GML pero estándar. SGML es un lenguaje muy trabajado, capaz de adaptarse a un gran abanico de problemas y a partir de él se han creado sistemas para almacenar información.

Por el año de 1989, para el ámbito de la red Internet, un usuario que había conocido el lenguaje de etiquetas (Markup) y los hiperenlaces creó un nuevo lenguaje llamado HTML, que fue utilizado para un nuevo servicio de Internet, la Web. Este lenguaje fue adoptado rápidamente por la comunidad y varias organizaciones comerciales crearon sus propios visores de HTML y riñeron entre ellos para hacer el visor más avanzado, inventándose etiquetas para sus necesidades. Desde 1996, el W3C (World Wide Web Consortium) ha tratado de poner orden en HTML y establecer sus reglas y etiquetas para que sea un estándar. Sin embargo, HTML creció de una manera descontrolada y no resolvió todos los problemas que planteaba la sociedad global de Internet[21].

En 1998, el mismo W3C empezó en el desarrollo de XML. En este lenguaje se ha pensado mucho más y muchas personas con grandes conocimientos en la materia han trabajado en su desarrollo. XML se creó para solucionar las carencias de HTML en lo que respecta al tratamiento de la información como:

- La mezcla entre el contenido y los estilos que se le quieren aplicar.

- No permitir compartir información con todos los dispositivos, como pueden ser computadoras o teléfonos móviles.
- Dependencia entre la presentación de la pantlla y el visor que se utiice.

1.2. Objetivos y Usos

XML se creó para que cumpliera con lo siguiente:

- Que fuera idéntico a HTML a la hora de servir, recibir y procesar la información, para aprovechar toda la tecnología implantada para éste.
- Que fuera formal y conciso desde el punto de vista de los datos y la manera de guardarlos.
- Que fuera extensible para poder utilizarse en todos los campos del conocimiento.
- Que fuera fácil de leer y editar.
- Que fuera fácil de implantar, programar y aplicar a los distintos sistemas.

Específicamente algunos de los objetivos planteados por el Grupo de Trabajo XML y el W3C son[6]:

- XML debe ser directamente utilizable sobre Internet.
- XML debe soportar una amplia variedad de aplicaciones.
- XML debe ser compatible con SGML.
- Debe ser fácil la escritura de programas que procesen documentos XML.
- El número de características opcionales en XML debe ser absolutamente mínimo, idealmente cero.
- Los documentos XML deben ser legibles por los usuarios de este lenguaje y razonablemente claros.
- El diseño de XML debe ser formal, conciso y preparado rápidamente.
- Los documentos XML deben ser fácilmente creables.
- La longitud de las marcas XML es de mínima importancia.

A estos fines se le unen unos estándares como el Unicode e ISO/IEC 10646 para caracteres, el Internet RCF 1766 para identificación de lenguajes, ISO 639 para códigos de nombres de lenguajes y también el ISO 3166 para códigos de nombres de países, para la normal comprensión de esta versión de XML.

XML se puede usar para infinidad de trabajos y aporta muchas ventajas en amplios escenarios. Algunas ventajas de XML en algunos campos prácticos[21] son:

- Comunicación entre datos. Si la información se transfiere en XML, cualquier aplicación podría escribir un documento de texto plano con los datos que estaba manejando en formato XML y otra aplicación recibir esta información y trabajar con ella.

- Aplicaciones Web. Hasta ahora cada navegador interpreta la información a su manera y los programadores de la Web tienen que adoptar el programa en función del navegador del usuario. Con XML se tiene una sola aplicación que maneja los datos y para cada navegador o soporte se puede tener una hoja de estilo o similar para aplicarle el estilo adecuado. Si mañana la aplicación debe correr en WAP sólo se tiene que crear una nueva hoja de estilo o similar.

1.3. Tecnologías relacionadas con XML

XML es sencillo y tiene pocas normas para su sintaxis. Simplemente utiliza las etiquetas que se necesitan, abriendo y cerrando de manera parecida a como se hace en HTML.

La sencillez de XML radica en que a su alrededor hay un mundo de tecnologías que son las encargadas de la programación o del acceso a la base de datos, de manejar la sintaxis y de la manera de aplicar estilos[2].

Tecnologías relacionadas con los procesos de:

- Contenidos: DTD o Esquema XML.
- Diseño: CSS o XSL.
- Programación: SAX o DOM.

1.3.1. Contenidos: DTD o Esquema XML

Para especificar el contenido de un documento XML, es decir qué etiquetas se pueden o deben encontrar en los documentos, en qué orden, dentro de qué otras, además de especificar los atributos que pueden o deben tener cada una de las etiquetas existen dos metalenguajes[2] que permiten definirlo, la DTD y el Esquema XML.

La DTD (Definition Type Document o Definición de Tipo de Documento), tiene una sintaxis especial, distinta a la de XML, que es sencilla aunque un poco rara si nunca se ha visto un documento similar.

Para evitar utilizar una DTD, ya que tiene una sintaxis muy especial, se intentó encontrar una manera de escribir en XML la definición del contenido de

[2]metalenguaje: lenguaje que define otro lenguaje.

un documento XML definiendo entonces el Esquema XML y de esta forma se ahorra el tener que aprender una sintaxis particular como la DTD.

Un detalle importante de señalar a la hora de hablar de las DTDs o de los Esquemas XML es que estos lenguajes también permiten comprobar la integridad de los datos en cualquier momento. Los metalenguajes de XML sirven para tomar un documento XML y comprobar que los datos que en éste se incluyen sean válidos, es decir, si lo que se tiene en XML concuerda con lo que se debe de tener. Eso se puede hacer al leer el documento, si no son válidos se genera un mensaje de error y se detiene el proceso de validación del documento. Si son válidos se hace lo que toque sin tener que preocuparse por la integridad de los datos.

1.3.2. Diseño: CSS o XSL

Para cada documento XML que se desee presentar en pantalla dando formato de la manera deseada se tiene que escribir una hoja de estilos o similar. También se tienen dos posibles lenguajes para dar formato a los textos de un documento XML y poder verlo por pantalla. La primera posibilidad es el CSS y la segunda opción es el XSL, bastante más avanzada.

CSS (Cascading Style Sheets o hojas de estilo en cascada) no es nada nuevo, ya se podía utilizar con HTML y se creó en un intento de separar la forma del contenido en HTML. En XML se utilizan las CSS, de una manera muy similar a cómo se utilizan en HTML. Los atributos de estilo que se pueden aplicar son los mismos y sus posibles valores también.

XSL, que son las siglas de XML Style Language, es el segundo lenguaje con el que se puede trabajar en XML. Este lenguaje no se limita a definir qué estilo aplicar a cada elemento del documento XML; además se pueden realizar pequeñas instrucciones típicas de los lenguajes de programación y la salida no tiene porque ser un documento HTML, sino que además podría ser de otros tipos, cualquiera que se pueda necesitar como un documento escrito en WML (para WAP), un documento de texto plano u otro documento XML.

1.3.3. Programación: SAX o DOM

El W3C ha especificado dos mecanismos para acceder a documentos XML y trabajar con ellos. Se trata de unas normas que indican a los desarrolladores la manera de acceder a los documentos. Estas normas incluyen una jerarquía de objetos que tienen unos métodos y atributos con los que se tienen que trabajar y que simplifican las tareas relativas al recorrido y acceso a las partes del documento. Estos dos mecanismos se denominan SAX y DOM.

SAX se utiliza para hacer un recorrido secuencial de los elementos del documento XML y DOM implica la creación de un árbol en memoria que contiene el

documento XML, y con él en memoria se puede hacer cualquier tipo de recorrido y acciones con los elementos que se quieran.

Se puede programar con el lenguaje de programación que se desee para acceder a un documento XML. Los creadores del lenguaje son los responsables de crear unas API[3] que cumplan las especificaciones de XML para que luego los desarrolladores de cada lenguaje las encuentren y puedan trabajar con ellas. Un lenguaje típico para trabajar con XML es Java y en este caso es SUN Microsystems el encargado de proveer el API que ha especificado el W3C y por lo tanto, los desarrolladores en Java cuentan con unas clases especiales que ha creado SUN para programar con XML.

1.4. Sintaxis de XML

Un documento XML es texto por naturaleza. Por ser XML, el documento consiste en datos de caracteres y marcado; los cuales se representan mediante texto.

Básicamente, lo que interesa son los datos de caracteres, pues representan la información. No obstante el marcado es importante ya que es lo que graba la estructura del documento.

En XML existe una gran variedad de construcciones de marcado; sin embargo reconocer dicho marcado es fácil porque siempre está encerrado entre paréntesis angulares (< >) y se le conoce como *etiqueta*. Obviamente, el marcado es lo que diferencia el documento XML de un texto plano.

1.4.1. Elementos

El pilar de XML es el *elemento*, pues es de lo que está constituido un documento XML. Cada elemento tiene un nombre y contenido.

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

El contenido de un elemento está delimitado por marcado especial al que se le reconoce como *etiqueta de apertura* y *etiqueta de cierre*[11].

La etiqueta de apertura es el nombre del elemento (*nombre* en este ejemplo) que está entre paréntesis angulares; la etiqueta de cierre contiene una diagonal extra antes del nombre.

En XML son necesarias ambas etiquetas. Lo siguiente no es correcto en XML:

[3]API: (Interfaz de Programación de Aplicaciones) es un conjunto de especificaciones de comunicación entre componentes software. Representa un método para conseguir abstracción en la programación, generalmente, aunque no necesariamente, entre los niveles o capas inferiores y los superiores del software.

```
<nombre>contenido
```

XML no define elementos, es un estándar flexible que proporciona una sintaxis común para almacenar información de acuerdo a una estructura.

Los nombres de elemento deben seguir ciertas normas. En XML los nombres deben comenzar con una letra o con el caracter de subrayado ("_"). El resto del nombre debe estar compuesto por letras, dígitos, el caracter de subrayado, el punto (".") o un guión ("−"). Los nombres no admiten espacios. Por último, los nombres no pueden comenzar con la cadena "xml", pues está reservada para la especificación XML misma.

Los siguientes son ejemplos de nombres de elementos válidos en XML:

```
<información-copyright>
<p>
<base64>
<direccion.cliente>
<nombre>
```

Los siguientes son nombres de elementos no validos en XML:

```
<123>
<direccion cliente>
<tom&jerry>
```

En XML los nombres son sensibles a mayúsculas y minúsculas. Por lo tanto, los siguientes nombres son diferentes:

```
<nombre>
<NOMBRE>
<Nombre>
```

Por convención, en XML los elementos HTML siempre van en mayúsculas y los elementos XML se escriben en minúsculas. Cuando un nombre está compuesto por varias palabras, éstas por lo regular se separan mediante un guión o caracter de subrayado.

```
<direccion_cliente>
<direccion-cliente>
```

Otra convención popular es poner en mayúsculas la primera letra de cada palabra y no utilizar ningún carácter de separación como:

```
<DireccionCliente>
```

1.4.2. Atributos

Es posible agregar información adicional a los elementos mediante *atributos*. Estos tienen un nombre y un valor. Los nombres de dichos atributos siguen las mismas normas que los de elementos.

Los elementos pueden tener uno o más atributos en la etiqueta de apertura, y el nombre del atributo se separa del valor mediante el símbolo igual (=). Además, el valor del atributo se encierra entre comillas simples o dobles.[11]

Por ejemplo, el elemento *nombre* puede tener el atributo *nombre_atributo* con valor *valor_atributo*:

```
<nombre nombre_atributo = "valor_atributo"> contenido </nombre>
```

XML requiere las comillas. El procesador XML rechazaría lo siguiente:

```
<nombre nombre_atributo = valor_atributo>contenido</nombre>
```

Las comillas pueden ser sencillas o dobles. Esto depende de si tiene que insertar comillas sencillas o dobles en el valor del atributo, como en el siguiente ejemplo en inglés en el que el valor del atributo contiene una comilla sencilla:

```
<confidencialidad nivel = "I don't know">
Este documento no es confidencial.
</confidencialidad>
```

o este, en el que el valor del atributo contiene comillas dobles:

```
<confidencialidad nivel = 'approved' "for your eyes only">
Este documento es secreto.
</confidencialidad>
```

1.4.3. Elementos vacíos

Los elementos que no tienen información se conocen como *elementos vacíos*. Por lo general, se incluyen en el documento por el valor de sus atributos.

Una nota importante acerca de los elementos vacíos: las etiquetas de apertura y final se mezclan y la diagonal de la etiqueta de cierre se agrega al final de la etiqueta de apertura.

Para XML, los siguientes elementos son idénticos:

```
<correo-e ref.="jdiaz@email.com"/>
<correo-e ref.="jdiaz@email.com"></correo-e>
```

1.4.4. Anidamiento de elementos

El contenido de los elementos no está limitado a sólo texto; los elementos pueden contener otros elementos, que a su vez pueden contener texto u otros elementos, y así sucesivamente.

Un documento XML es un árbol de elementos. No hay límite para la profundidad del árbol, además de que los elementos pueden repetirse.

Al elemento que está dentro de otro se le conoce como *hijo*. El elemento en el que éste está contenido se conoce como *padre*. En el siguiente ejemplo, *nombre* tiene dos hijos: *nombre de pila* y *apellido*. El elemento nombre es el padre de estos dos elementos.

```
<nombre>
<nombre-pila>Jesús</nombre-pila>
<apellido>Sandoval</apellido>
</nombre>
```

Las etiquetas de apertura y de cierre siempre deben estar equilibradas y los hijos deben estar contenidos dentro de sus padres, en líneas separadas. En otras palabras, no es posible que la etiqueta de cierre de un hijo aparezca después de la etiqueta de cierre de su padre. Por lo tanto, lo siguiente es ilegal:

```
<nombre><nombre-pila>Jesús</nombre-pila><apellido>Sandoval</nombre></apellido>
```

1.4.5. Elemento Raíz

El documento debe tener un solo elemento raíz. En otras palabras, todos los elementos del documento deben ser hijos de un solo elemento. El siguiente ejemplo no es válido porque hay dos elementos *entrada* que no están contenidos en un elemento principal:

```
<?xml version="1.0"?>
<entrada>
<nombre>Juan Díaz</nombre>
<correo-e ref.="jdias@email.com"/>
</entrada>
<entrada>
<nombre>Jesús Sandoval</nombre>
<correo-e ref.="jsandoval@email.com"/>
</entrada>
```

Basta con introducir una nueva raíz para sea válido:

```
<?xml version="1.0"?>
<libreta-direcciones>
<entrada>
```

```
<nombre>Juan Díaz</nombre>
<correo-e ref.="jdiaz@email.com"/>
</entrada>
<entrada>
<nombre>Jesús Sandoval</nombre>
<correo-e ref.="jsandoval@email.com"/>
</entrada>
</libreta-direcciones>
```

1.4.6. Declaración XML

La *declaración XML* es la primera línea del documento. Ésta lo identifica como documento XML. También indica la versión de XML que se utiliza en el documento[11].

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

La declaración puede contener otros atributos para soportar otras características como la codificación del conjunto de caracteres.

La declaración XML es opcional. Si se desea incluir la declaración, ésta debe ser en la primera línea del documento. La recomendación XML sugiere que se incluya en todos sus documentos XML.

1.4.7. Comentarios

Para insertar *comentarios* en un documento, se encierran entre los caracteres <!- - y - - >. Por lo general, los comentarios se utilizan para notas, indicación de propiedad, etcétera. Están hechos para las personas y son ignorados por el procesador XML. Un ejemplo:

```
<!-- Este es un comentario -->
```

Los comentarios no pueden insertarse en el marcado; deben colocarse antes o después de él.

1.4.8. Unicode

Los caracteres de los documentos XML siguen el estándar Unicode, el cual es una extensión del conjunto de caracteres ASCII. El Consorcio Unicode es responsable de la publicación y el mantenimiento del estándar Unicode. La ISO publica el mismo estándar como ISO/IEC 10646.

Unicode soporta todos los caracteres de los idiomas hablados, así como todos los símbolos matemáticos, entre otros. Soporta el inglés, cirílico, japonés, chino, etcétera.

El soporte para Unicode es un gran paso hacia adelante en la internacionalización de Web.

Los caracteres Unicode ocupan lo doble que su equivalente en Latín-1; lógicamente, los documentos XML pesan lo doble que los archivos de texto normales. Por fortuna hay una opción. En la mayoría de los casos, no se necesitan 16 bits y se pueden codificar documentos XML con un conjunto de caracteres de 8 bits.

El procesador XML debe reconocer las codificaciones UTF-8 y UTF-16. Como su nombre lo indica, UTF-8 utiliza 8 bits para caracteres. La mayoría de los procesadores soportan otras codificaciones. En particular soportan la codificación ISO 8859-1 (el nombre oficial de Latín-1) para las lenguas europeas del oeste.

Los documentos que utilizan una codificación diferente de UTF-8 o UTF-16 deben comenzar con una declaración XML. Dicha declaración debe contener un atributo que indique la codificación utilizada.

Por ejemplo, un documento escrito en Latin-1 podría contener la siguiente declaración:

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

1.4.9. Entidades

Por lo general, los documentos complejos están divididos en varios archivos: el texto, los gráficos acompañantes, etcétera.

Sin embargo, XML no se organiza en términos de archivos. En su lugar, organiza físicamente los documentos en *entidades*. En algunos casos, las entidades son equivalentes a los archivos, pero en otras, no es así.

Para insertar una entidad en un documento, se utilizan *referencias de entidad*, representadas por el nombre de la entidad entre un caracter ampersand '&'y un carácter punto y coma ';'. En la aplicación, la referencia de entidad se reemplaza con el contenido de la entidad. Si se supone que se define una entidad *eu*, la cual tiene el valor *Estados Unidos*, las siguientes dos líneas son equivalentes:

```
<pais>&eu;</pais>
<pais>Estados Unidos</pais>
```

XML predefine entidades para los caracteres utilizados en el marcado (paréntesis angulares, comillas, etcétera). Las entidades se utilizan para representar el contenido (los caracteres a insertar) de elementos o atributos. Algunas entidades son:

- <, que se utiliza para insertar el paréntesis angular izquierdo "<".
- &, que se utiliza para insertar el ampersand "&".
- >, que se utiliza para insertar el paréntesis angular derecho ">".
- ', que se utiliza para insertar las comillas simples ', esencialmente en valores de parámetros.
- ", que se utiliza para insertar las comillas dobles ", esencialmente en valores de parámetros.

Lo siguiente no es válido pues el ampersand podría confundir al procesador XML:

```
<empresa>Mark & Spencer</empresa>
```

Este código debe reescribirse de manera que inserte el ampersand mediante la entidad &:

```
<empresa>Mark &amp; Spencer</empresa>
```

XML también soporta *referencias de carácter* en donde una letra se reemplaza con un carácter equivalente en Unicode.

Las referencias de carácter que comienzan con &#x proporcionan una representación hexadecimal del código de caracteres. Las que comienzan con &# proporcionan una representación decimal del código de caracteres.

1.4.10. Atributos especiales

XML define dos atributos:

- xml:space, para aquellas aplicaciones que eliminan espacios duplicados como los exploradores Web que eliminan los espacios innecesarios en HTML. Este atributo controla si la aplicación puede eliminar espacios. Si se establece a *preserve*, la aplicación deberá preservar todos los espacios de este elemento y de sus hijos. Si se establece a *default*, la aplicación puede utilizar su manejo de espacios predeterminado.
- xml:lang en publicación electrónica, por lo general se utiliza para saber en que idioma está escrito el contenido. Este atributo puede utilizarse para indicar el idioma del contenido del elemento. Por ejemplo:

```
<p xml:lang=" sp-MX">¿Qué color es?</p>
<p xml:lang="en-US">What color is it?</p>
```

1.4.11. Instrucciones de procesamiento

Las PIs (*instrucciones de procesamiento*) son un mecanismo para insertar instrucciones que no son de XML en forma de secuencias de comandos en el documento.

A primera vista, las instrucciones de procesamiento van en contra del concepto de que el procesamiento siempre se deriva de la estructura. En SGML y XML, el procesamiento se deriva de la estructura del documento. No debería existir la necesidad de insertar instrucciones específicas en un documento. Esta es una de las mejoras de SGML en comparación con los primeros lenguajes de marcado.

Esa es la teoría. En la práctica, hay casos en donde es más fácil insertar instrucciones de procesamiento que definir estructuras que son más complejas.

Una declaración XML es una instrucción de procesamiento:

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

Finalmente, las instrucciones de procesamiento son utilizadas por aplicaciones específicas. Por ejemplo, XMetaL (un editor XML) las utiliza para crear plantillas. La siguiente instrucción de procesamiento es específica de XMetaL:

```
<?xml-texto_reemplazo {Haga clic aquí para escribir el nombre}?>
```

La instrucción de procesamiento está encerrada entre <? y ?>. El primer nombre es el destino e identifica la aplicación o el dispositivo al que están dirigidas las instrucciones. El resto de las instrucciones de procesamiento están en un formato específico de la aplicación que no necesariamente debe ser XML.

1.4.12. Secciones CDATA

Los caracteres de marcado (el paréntesis angular izquierdo, el ampersand, etcétera) que aparecen en el contenido de un elemento deben insertarse mediante una entidad.

Para algunas aplicaciones es difícil insertar caracteres de marcado, si utilizan demasiados. Las ecuaciones matemáticas pueden utilizar muchos paréntesis angulares. Incluir un lenguaje de secuencia de comandos en un documento es muy difícil, así como insertar los paréntesis angulares y el ampersand. Además incluir un documento XML en otro documento XML también es muy difícil.

Las *secciones CDATA* están hechas para estos casos. Dichas secciones están delimitadas por '<[CDATA['y ']]>'. El procesador XML ignora todo el marcado excepto]]> (lo que significa que no es posible incluir una sección CDATA en otra sección CDATA).

1.5. Estructura de XML

Un documento XML tiene dos estructuras, una lógica y otra física. Físicamente, el documento está compuesto por unidades llamadas entidades. Una entidad puede hacer referencia a otra entidad causando que ésta se incluya en el documento. Cada documento comienza con una entidad documento, también llamada raíz. Lógicamente, el documento está compuesto de declaraciones, elementos, comentarios, referencias a caracteres e instrucciones de procesamiento, todos los cuales están indicados por una marca explícita. Las estructuras lógica y física deben encajar de manera adecuada.

Los documentos XML se dividen en dos grupos, documentos bien formados y documentos válidos.

Un objeto textual o documento XML se dice que está bien formado si, considerándolo como conjunto, encaja con las especificaciones XML de producción, lo que implica:

- Contiene uno o más elementos.
- Hay exáctamente un elemento, llamado raíz o elemento documento, de forma que ninguna parte del mismo aparece en el contenido de ningún otro elemento. Para todos los demás elementos, si la etiqueta inicial está en el contenido de otro elemento, la etiqueta final forma parte del contenido del mismo elemento. Dicho de forma más clara, los elementos, delimitados por una etiqueta de inicio y otra de final, se encuentran anidados correctamente.
- Cada una de sus partes procesadas está bien formada.
- Todas las etiquetas deben estar balanceadas, esto es, todos los elementos que contengan datos de tipo caracter deben tener etiquetas de principio y fin (no está permitida la omisión excepto para los elementos vacíos.).
- El valor de cada atributo debe ir entrecomillado. El caracter comilla simple puede utilizarse si el valor contiene caracteres de comillas dobles, y viceversa. Si se necesitan ambos tipos de comillas, se utiliza ' y &quot.
- Cualquier elemento vacío debe terminar con /> o se debe hacer no vacío añadiéndole una etiqueta de fin. Por ejemplo,
 se convertirá en
 o en
</BR>.
- No debe haber etiquetas aisladas en el texto. Por ejemplo < ó & debe darse como < y &amp, y la secuencia]]> debe darse como]]> si no ocurre esto como final de una sección marcada como CDATA.
- Los elementos deben anidar dentro de sí sus propiedades, es decir no se deben sobreponer etiquetas, como en el resto de SGML.

- Los documentos bien formados sin DTD pueden utilizar atributos en sus elementos, pero éstos deben ser todos del tipo CDATA, por defecto.

Un documento bien formado se dice además que es válido, si tiene una DTD como el resto de aplicaciones SGML o un esquema XML. Si se utiliza una DTD el documento XML válido comienza como cualquier otro documento SGML, es decir con una Declaración de Tipo de Documento (DTD) y cumple todas las restricciones que proporciona su especificación a través de la DTD:

```
<?xml version="1.0"?>
<!DOCTYPE anuncio SYSTEM "http://www.foo.org/ad.dtd">
<anuncio>
<titulo>...<foto/>...</titulo>
<texto>...</texto>
</anuncio>
```

Dado que XML está diseñado para ser un subconjunto de SGML, cualquier documento XML válido debe ser también un documento SGML válido.

1.5.1. Estructura lógica

Cada documento XML contiene uno o más elementos cuyos límites están delimitados por etiquetas de comienzo y de final o, en el caso de elementos vacíos, por una etiqueta de elemento vacío.

Cada elemento tiene un tipo, identificado por un nombre, denominado identificador genérico, y puede tener un conjunto de especificaciones de atributos.

Cada especificación de atributo tiene un nombre y un valor. Estas especificaciones no tienen restricciones de nombre y tipo con excepción de que los nombres que comienzan con XML están reservados para etiquetas o atributos del estándar XML.

1.5.2. Estructuras físicas

Un documento XML puede consistir de una o más unidades de almacenamiento virtual, llamadas entidades. Todas estas unidades tienen contenido y todas ellas (excepto la entidad documento y el subconjunto externo de la DTD) están identificadas por un nombre. Cada documento XML contiene una entidad, llamada entidad documento, que sirve como punto de comienzo para el procesador XML y puede contener el documento completo.

Las entidades pueden ser analizadas o no analizadas (también llamadas procesadas o sin procesar). El contenido de una entidad analizada se conoce también como texto de reemplazo, y es parte integrante del documento. Las entidades no analizadas son recursos, tales como enlaces, cuyo contenido puede o no ser texto, o en caso de que sea texto que no sea XML. Cada entidad no

asociada tiene una notación asociada, identificada por un nombre. Aparte de obligar al procesador XML a hacer accesible a la aplicación el nombre de esta notación y sus identificadores asociados, XML no proporciona ninguna otra restricción sobre el contenido de estas entidades. La forma de invocar ambos tipos de entidades es a través de su nombre, en el caso de las analizadas a través de su referencia a entidad y en el de las no analizadas a través de sus atributos de entidad.

Las entidades generales son entidades analizadas que se usan en el interior del documento. Las entidades parametrizadas son entidades analizadas que se usan en el ámbito de la DTD. Estos dos tipos de entidades usan distintos tipos de referencias y se reconocen en contextos distintos.

1.6. Ventajas de XML

- Los autores y proveedores pueden diseñar sus propios tipos de documentos usando XML. Los tipos de documentos pueden ser 'hechos a la medida de una audiencia'. Autores y diseñadores serán libres de inventar sus propias etiquetas.
- La información contenida puede ser más 'rica'y fácil de usar, porque las habilidades hipertextuales de XML son mayores.
- XML puede dar más y mejores facilidades para la representación en los visualizadores.
- Elimina muchas de las complejidades de SGML en favor de la flexibilidad del modelo, con lo que la escritura de programas para manejar XML será más sencilla que haciendo el mismo trabajo en SGML.
- La información será más accesible y reutilizable porque la flexibilidad de las etiquetas de XML pueden utilizarse sin tener que amoldarse a reglas específicas de un fabricante.

Capítulo 2

Analizadores

Un parser o analizador sintáctico lee el documento XML y verifica que esté - bien formado; algunos analizadores también comprueban que el código XML sea válido.

El analizador es la herramienta principal de cualquier aplicación XML. Mediante el analizador no solamente se puede comprobar si los documentos son bien formados o válidos, sino que también se puede incorporar a las aplicaciones, de manera que estas puedan manipular y trabajar con documentos XML[20].

Un analizador es un componente de software ubicado entre la aplicación y el documento XML. Su objetivo es aislar al desarrollador de las complejidades de la sintaxis de XML.

2.1. Analizadores de validación

Los documentos XML pueden ser bien formados o válidos. Los documentos bien formados respetan las normas sintácticas. Los documentos válidos no sólo respetan dichas normas, sino que también se apegan a una estructura descrita en una DTD o en un Esquema XML.

De igual manera, existen analizadores de validación y analizadores que no son de validación. Ambos analizadores hacen valer las normas sintácticas, pero sólo los analizadores de validación saben como validar documentos con sus DTDs y sus Esquemas XML.

Aunque es posible verificar manualmente un documento, éste no es un método práctico, sobre todo cuando se manejan documentos extensos y elaborados por distintas aplicaciones o usuarios. Lógicamente, es mucho más cómodo y eficiente utilizar un analizador de validación, no obstante, necesita conocer las reglas que rigen el documento o de lo contrario no sabrá si es o no válido.

Los analizadores que no son de validación pueden leer documentos válidos

pero no los validan. De manera similar, un analizador de validación acepta documentos bien formados pero se comporta como un analizador que no es de validación.

2.2. DTD (Definition Type Document)

Como antes se comentó, los documentos XML pueden ser válidos o bien formados. En cuanto a los válidos, ya se sabe que su gramática se puede definir en las DTD.

Físicamente, una DTD es un documento con estructura similar a un documento XML. El contenido, sin embargo, no son datos propiamente dichos, como en el caso de XML, sino indicaciones acerca de la estructura de un determinado tipo de documento.

A pesar de que una DTD puede parecer relativamente compleja, no es más que una enumeración de los elementos que podrán existir en un documento XML, indicando el tipo y orden de cada uno de ellos.

Una DTD especifica los tipos de etiquetas que se pueden incluir en un documento XML y el contenido de dichas etiquetas. Se puede usar la DTD para asegurar que no se crea una estructura XML inválida. También se puede usar para asegurar que la estructura XML que se esta leyendo, o que va a ser enviada por la red, es válida. Desafortunadamente, es difícil especificar una DTD para un documento complejo de forma que evite todas las combinaciones inválidas y permita las válidas.

2.2.1. Referencias a las DTDs

Hay dos modos de referenciar una DTD en un documento XML[5]:

- Incluir dentro del documento una referencia al documento DTD en forma de URI[1] (Universal Resource Identifier, o identificador universal de recursos) y mediante la siguiente sintaxis:

  <!DOCTYPE ficha SYSTEM "http://www.dat.etsit.upm.es/ãbarbero/D'

  En este caso la palabra SYSTEM indica que la DTD se obtendrá a partir de un elemento externo al documento e indicado por el URI que lo sigue por supuesto entrecomillado.

[1] URI: (acrónimo: Uniform Resource Identifier, literalmente "Identifador Uniforme de Recurso"). Conjunto de caracteres que identifica un recurso de red, mediante un nombre o utilizando sus datos de localización.

- O bien incluir dentro del propio documento la DTD de este modo:

```
<?xml version="1.0"?>
<!DOCTYPE ficha [
<!ELEMENT ficha (nombre+, apellido+, direccion+, foto?)>
<!ELEMENT nombre (#PCDATA)>
<!ATTLIST nombre sexo (masculino|femenino) #IMPLIED>
<!ELEMENT apellido (#PCDATA)>
<!ELEMENT direccion (#PCDATA)>
<!ELEMENT foto EMPTY>
]>
<ficha>
<nombre>Angel</nombre>
<apellido>Barbero</apellido>
<direccion>c/Ulises, 36</direccion>
</ficha>
```

La forma de incluir la DTD directamente como en este ejemplo, se obtiene al añadir a la declaración <!DOCTYPE, la propia DTD entre los símbolos '['y ']'. Todo lo que hay entre ellos será considerado parte de la DTD.

2.2.2. Definiciones

En cuanto a la definición de los elementos, es bastante intuitiva: después de la cláusula <!ELEMENT se incluye el nombre del elemento (el que luego se indicará en la etiqueta), y después diferentes cosas en función del elemento:

- Si el elemento es no vacío, se indica entre paréntesis el contenido que puede tener el elemento; la lista de elementos hijos o que descienden de él si los tiene, separados por comas; o el tipo de contenido, normalmente #PCDATA, que indica datos de tipo texto, que son los más habituales.
- Si es un elemento vacío, se indica con la palabra EMPTY.

A la hora de indicar los elementos descendientes (los que están entre paréntesis) se ve que van seguidos de unos caracteres especiales: '+', '*', '?'y '|'[1]. Sirven para indicar el tipo de uso que se permite hacer de esos elementos dentro del documento:

- + : uso obligatorio y múltiple; permite uno o más elementos de ese tipo dentro del elemento padre, pero como mínimo uno.
- * : opcional y múltiple; puede no haber ninguna ocurrencia, una o varias.
- ? : opcional pero singular; puede no haber ninguno o sólo uno.
- | : equivale a un OR, es decir, da la opción de usar uno y sólo un elemento de entre los que forman la expresión.

De este modo, si por ejemplo se encuentra en una DTD la siguiente declaración:

```
<!ELEMENT ficha (nombre+, apellido+, direccion*, foto?, telefono*|fax*)>
```

Se sabe que del elemento ficha puede contener los siguientes elementos: un nombre y un apellido como mínimo, pero puede tener más de uno de cada uno de ellos; opcionalmente puede incluirse una o varias direcciones, pero no es obligatorio; opcionalmente también se puede incluir una única foto; y por último, pueden incluirse, aunque no es obligatorio en ninguno de los dos casos, uno o más teléfonos o uno o más números de fax.

Un documento XML presenta una jerarquía muy determinada definida en la DTD si es un documento válido, pero siempre inherente al documento en cualquier caso (siempre se puede inferir esa estructura a partir del documento sin necesidad de tener una DTD en el que basarse) con lo que se puede representar como un árbol de elementos. Existe un elemento raíz que siempre debe ser único (sea nuestro documento válido o sólo bien formado) y que se llamará con el mismo nombre que en la definición del <!DOCTYPE si está asociado a una DTD o cualquiera que se desee en caso contrario. Y de él descienden las ramas de sus respectivos elementos descendientes o hijos.

La DTD, por ser precisamente la definición de esa jerarquía, describe precisamente la forma de ese árbol. La diferencia está en que la DTD define la forma del árbol de elementos, y un documento XML válido puede basarse en ella para estructurarse, aunque no tienen que tener en él todos los elementos, si la DTD no te obliga a ello. Un documento XML bien formado sólo tendrá que tener una estructura jerarquizada pero sin tener que ajustarse a ninguna DTD concreta.

Para la definición de los atributos, se usa la declaración <!ATTLIST, seguida de:

- El nombre de elemento del que se están declarando los atributos.
- El nombre del atributo.
- Los posibles valores del atributo entre paréntesis y separados por el caracter |, que al igual que para los elementos, significa que el atributo puede tener uno y sólo uno de los valores incluidos entre paréntesis. O bien, si no hay valores definidos, se escribe CDATA para indicar que puede ser cualquier valor. También se puede indicar con la declaración ID que el valor alfanumérico que se le de será único en el documento, y se podrá referenciar ese elemento a través de es e atributo y valor.
- De forma opcional y entrecomillado, un valor por omisión del atributo si no se incluye otro en la declaración.

- Por último, si es obligatorio cada vez que se usa el elemento en cuestión declarar este atributo, es necesario declararlo con la cláusula #REQUIRED; si no lo es, se debe poner #IMPLIED, o bien #FIXED si el valor de dicho atributo se debe mantener fijo a lo largo de todo el documento para todos los elementos del mismo tipo (notar que no es lo mismo esto a lo que significa ID).

2.3. Esquema XML

Un esquema XML es una herramienta eficaz y compleja para crear y validar la estructura de documentos XML compatibles. De forma parecida al modelado de datos de una base de datos relacional, un esquema proporciona una forma de definir la estructura de los documentos XML al especificar los elementos que se pueden utilizar en ellos, así como la estructura y tipos que estos elementos deben tener para ser válidos con respecto al esquema específico.

Un esquema es un documento XML, que suele tener la extensión .xsd, en el que se describe el contenido de los elementos XML mediante código XML válido: los elementos y los atributos se declaran con los elementos *element* y *attribute*, y la estructura se crea con los elementos *simpleType* y *complexType*.

Un esquema es un documento XML en el que se define una clase de documentos XML mediante la especificación de la estructura o el modelo de los documentos XML para un esquema determinado. En un esquema se identifican las restricciones en el contenido de los documentos XML y se describe el vocabulario (reglas o gramática) que deben seguir los documentos XML compatibles para ser considerados válidos con respecto al esquema en particular. La validación de un documento XML es el proceso que asegura que el documento se ajusta a la gramática especificada en el esquema.

Los esquemas proporcionan las siguientes mejoras sobre las DTD (Document Type Definitions, definiciones de tipo de documento):

- Al usar un esquema se dispone de tipos de datos adicionales.
- Con un esquema se pueden crear tipos de datos personalizados.
- Un esquema utiliza la sintaxis XML.
- Un esquema admite conceptos orientados a objetos, como polimorfismo y herencia.

2.3.1. Definiciones

Las unidades de creación básicas de los esquemas XML son los elementos y los atributos. Los tipos de datos definen el contenido válido de los elementos

y atributos. Cuando se crean esquemas XML, se definen los elementos y los atributos individuales y se les asignan tipos válidos. Los elementos describen datos, mientras que los atributos son como las propiedades de un elemento, ya que proporcionan información adicional sobre el elemento de la misma forma que las propiedades describen las características de objetos y clases.

Elementos

Un elemento describe los datos que contiene. Los elementos también pueden contener otros elementos y atributos. Cuando una definición de elemento contiene elementos o atributos adicionales, se trata de un tipo complejo.

La definición básica de un elemento consta de un nombre y un tipo de datos. En el siguiente ejemplo se muestra cómo definir un elemento denominado *cantidad*, con un tipo de valor entero.

```
<xs:element name="cantidad" type="xs:integer" />
```

El elemento clasifica los datos, en este caso como una cantidad. El tipo define el contenido válido que contiene el elemento, en este caso un entero.

En el ejemplo siguiente se muestra una instancia válida de datos XML que son compatibles con el elemento *cantidad* definido en el ejemplo anterior:

```
<cantidad>63</ cantidad>
```

En el ejemplo siguiente se muestra una instancia no válida del elemento *cantidad*:

```
<cantidad>sesenta y tres</ cantidad>
```

Atributos

Un atributo es una definición de tipo simple con nombre que no puede contener otros elementos. Los atributos también pueden asignarse a un valor predeterminado opcional y deben aparecer en la parte inferior de las definiciones de tipo complejo.

El siguiente código muestra cómo declarar un atributo denominado *DescuentoPedido* que está definido con el tipo simple *number*. El uso de un atributo aquí tiene sentido ya que los atributos son opcionales. Si no se proporciona *DescuentoPedido*, los datos XML serán válidos.

```
<xs:element name="InfoPedido">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="NombreCliente" type="xs:string" />
         <xs:element name="NumeroPedido" type="xs:positiveInteger" />
```

```
            <xs:element name="TotalPedidol" type="xs:number" />
        </xs:sequence>
        <xs:attribute name"DescuentoPedido" type="xs:number" />
    </xs:complexType>
</xs:element>
```

Nombre de elementos y atributos

A continuación, se muestran algunos detalles importantes relacionados con elementos y atributos:

- XML distingue entre mayúsculas y minúsculas. Un elemento denominado *codigoestado* es distinto de un elemento denominado *codigoEstado*.
- Los valores de atributos deben estar siempre entre comillas.
- Los nombres de elementos no pueden empezar por un número o un signo de subrayado ni por las letras "XML".
- Los nombres de elementos no pueden contener espacios.

Cuando utilizar elemento o atributo

Los elementos se utilizan para encapsular porciones de datos, y los atributos generalmente se utilizan para proporcionar información adicional sobre un elemento, en vez de encapsular los propios datos sin formato. En realidad, el uso de un elemento o un atributo depende de las necesidades de la aplicación.

Se utilizan los atributos si la información requiere datos de tipo simple y:

- La información requiere un valor predeterminado o fijo.
- La información requiere datos que son metadatos de un elemento existente.
- Si el tamaño del archivo XML es un problema, los atributos tienden a ocupar menos bytes que los elementos.

En la lista siguiente se describen las diferencias principales entre elementos y atributos desde la perspectiva del esquema:

- Un esquema puede definir si el orden de los elementos es significativo, sin embargo los atributos pueden producirse en cualquier orden.
- Los elementos pueden anidarse con la etiqueta <*choice*>, lo que significa que sólo puede aparecer uno y sólo uno de los elementos enumerados.
- Los elementos pueden producirse más de una vez, mientras que los atributos no.

2.3.2. Tipos

Los tipos en esquemas XML definen el tipo de datos válido que pueden contener los elementos o atributos y pueden ser simples o complejos[7]. Además, los tipos pueden ser con nombre o sin nombre.

Tipos simples

Hay dos categorías principales de tipos simples:

- *Tipos integrados*, que están definidos por la especificación de esquema XML del consorcio de World Wide Web, por ejemplo, *string*, *boolean* y *float*. Los tipos integrados incluyen tanto tipos primitivos como derivados. Los tipos de datos primitivos no provienen de otros tipos de datos. Por ejemplo, *float* es un concepto matemático que no proviene de otros tipos de datos. Los tipos de datos derivados se definen en función de tipos de datos existentes. Por ejemplo, un entero es un caso especial derivado del tipo de datos decimal.

- Los *tipos simples definidos por el usuario* provienen de los tipos integrados de W3C mediante la aplicación de valores definidos por el usuario en elementos denominados aspectos.

Los aspectos restringen los valores aceptables de los tipos simples. Mediante la aplicación de aspectos se pueden crear tipos simples definidos por el usuario.

En el ejemplo siguiente se aplica el aspecto *maxInclusive* a un tipo simple denominado *Limite* para restringir los valores aceptables del tipo *positiveInteger* a cantidades de 1 a 100:

```
<xs:simpleType name="Limite">
   <xs:restriction base="xs:positiveInteger">
      <xs:maxInclusive value="100" />
   </xs:restriction>
</xs:simpleType>
```

Tipos complejos

Los tipos complejos son definiciones de elementos que pueden incluir otros elementos, atributos y grupos. Una diferencia importante entre los tipos simples y complejos es que los complejos pueden incluir elementos y atributos declarados como tipos simples o complejos, mientras que los tipos simples sólo pueden incluir aspectos.

Un escenario habitual donde se utilizaría un tipo complejo es parte de un esquema que valida un pedido donde se requiere una dirección de envío y una dirección de facturación. Podría definir un tipo complejo que representa una

dirección y declarar los elementos *enviarA* y *facturarA* de ese tipo complejo.

En el ejemplo siguiente se muestra cómo crear un tipo complejo denominado Direccion:

```
<xs:complexType name="Direccion">
   <xs:sequence>
      <xs:element name="nombre" type="xs:string" />
      <xs:element name="calle" type="xs:string" />
      <xs:element name="ciudad" type="xs:string" />
      <xs:element name="estado" type="xs:string" />
      <xs:element name="CP" type="xs:string" />
   </xs:sequence>
</xs:complexType>
```

Un elemento declarado con el tipo Direccion podría ser como éste:

```
<xs:element name="enviarA" type="Direccion" />
```

En el ejemplo siguiente se muestra cómo declarar varios elementos utilizando el tipo complejo definido.

```
<xs:element name="InfoCliente">
  <xs:complexType>
     <xs:sequence>
        <xs:element name="Nombre" type="xs:string" />
        <xs:element name="enviarA" type="Direccion" />
        <xs:element name="facturarA" type="Direccion" />
     </xs:sequence>
  </xs:complexType>
</xs:element>
```

2.4. El analizador y la aplicación

La arquitectura de los programas con XML, se divide en dos partes[20]:

- El analizador trata con el documento XML.
- La aplicación consume el contenido del documento por medio del analizador.

El analizador y la aplicación deben compartir un modelo común para los datos XML. El analizador lee el documento XML y la aplicación manipula el contenido del documento como si estuviera trabajando directamente con el documento XML.

Existen dos formas básicas de crear una interfaz entre el analizador y la aplicación:

- Por medio de interfaces basadas en objetos.
- Por medio de interfaces basadas en eventos.

En la práctica, los dos métodos son más complementarios que competitivos.

2.4.1. Interfaz basada en objetos

Al utilizar una interfaz basada en objetos, el analizador crea explícitamente un árbol de objetos que contiene todos los elementos del documento XML. Esta es probablemente la interfaz más natural para la aplicación, ya que recibe un árbol en memoria que concuerda exactamente con el documento en disco.

Obviamente es más conveniente para la aplicación trabajar con el árbol en memoria, aunque sea sólo porque no tiene que preocuparse por la sintaxis de XML. Además, si se utiliza un analizador de validación, el árbol puede haber sido validado con la DTD.

DOM es una interfaz basada en objetos: se comunica con la aplicación mediante la creación explicita de un árbol de objetos en memoria. El árbol en memoria es un mapa exacto del árbol de elementos existentes en el documento XML.

Las interfaces basadas en objetos tales como DOM son simples ya que ofrecen una vista que se asemeja mucho al documento XML. También son ideales para aquellas aplicaciones cuyo objetivo es manipular XML tan sólo para manejar documentos XML.

No obstante, para aplicaciones que no están tan centradas en XML, una interfaz basada en objetos es menos atractiva. Ciertamente esas aplicaciones tienen su propia estructura de datos y sus propios objetos los cuales no están basados en XML. Para esas aplicaciones es mejor no crear el árbol DOM, sino cargar directamente el documento en su estructura de datos.

Por otro lado, la aplicación tiene que mantener dos copias de los datos en memoria (una en el árbol DOM y otra en la estructura de la aplicación), lo cual es ineficiente. Tal vez esto no represente un problema para las aplicaciones de escritorio, pero puede hacer que un servidor se haga muy lento. En estos casos, una interfaz basada en eventos es más adecuada. La principal diferencia entre una interfaz basada en objetos y una basada en eventos es que ésta última no crea explícitamente el árbol; lo crea en forma implícita.

2.4.2. Interfaz basada en eventos

El segundo método para crear una interfaz entre el analizador y la aplicación se basa en el uso de eventos. Una interfaz basada en eventos es natural para

el analizador pero es más compleja para la aplicación. No obstante, con algo de práctica, las interfaces basadas en eventos resultan ser muy poderosas. Por esta razón, cada vez más programadores están optando por el uso de interfaces basadas en eventos.

Con una interfaz basada en eventos el analizador no crea explícitamente un árbol de objetos, sino que lee el archivo y genera eventos a medida que encuentra elementos, atributos o texto en el archivo. Existen eventos para el inicio de elementos, el fin de elementos, atributos, contenido de texto, entidades, etcétera.

A primera instancia esta solución es menos natural para la aplicación, ya que no recibe un árbol explícito que concuerda con el documento, sino que tiene que escuchar a los eventos y determinar que árbol se esta describiendo.

En la práctica, ambas formas de interfaces son útiles pero sirven para distintos objetivos. Las interfaces basadas en objetos son ideales para aplicaciones que manipulan documentos XML tales como exploradores, editores, procesadores de XSL, etcétera.

Las interfaces basadas en eventos están dirigidas a las aplicaciones que mantienen su propia estructura de datos en un formato que no es XML. Por ejemplo, las interfaces basadas en eventos se adaptan muy bien a las aplicaciones que importan documentos XML en bases de datos. El formato de la aplicación es el esquema de la base de datos, no el esquema XML. Estas aplicaciones tienen su propia estructura de datos y realizan asignaciones de una estructura XML a su estructura interna.

Una interfaz basada en eventos también es más eficiente ya que no crea explícitamente el árbol de XML en memoria. Son necesarios menos objetos, además de que se utiliza menos memoria.

Un analizador basado en eventos genera eventos a medida que lee un documento XML. Los eventos del analizador son similares a los de la interfaz de usuario tales como ONCLICK (en un explorador) o los eventos AWT (en Java).

Los eventos indican a la aplicación que algo ocurrió, por si ésta desea reaccionar a ese evento. Las aplicaciones registran *controladores de eventos*, que son funciones que procesan los eventos.

En un analizador XML los eventos no están relacionados con las acciones de los usuarios, sino con los elementos del documento XML que se está leyendo. Existen eventos para:

- Etiquetas de apertura de elementos.
- Etiquetas de cierre de elementos.

- El contenido de los elementos.
- Entidades.
- Errores de análisis.

El analizador XML lee este documento y lo interpreta. Siempre que reconozca algo en el documento, generará un evento. El analizador lee la declaración de XML y a continuación genera un evento correspondiente a dicha declaración.

Los eventos en conjunto describen a la aplicación el árbol del documento. Un evento de etiqueta de apertura significa "bajar un nivel en el árbol", mientras que un elemento de etiqueta de cierre significa "subir un nivel en el árbol".

Una interfaz basada en eventos es la interfaz más natural para un analizador porque simplemente tiene que reportar lo que ve. Hay que resaltar que el analizador reporta suficiente información como para crear el árbol de los documentos XML pero, a diferencia de un analizador basado en objetos, no crea el árbol en forma explícita.

2.4.3. La necesidad de tener estándares

Idealmente la interfaz entre el analizador y la aplicación debe ser estándar. Una interfaz estándar permite escribir software utilizando un analizador e implementar el software con otro analizador.

Hay una similitud con las bases de datos. Las bases de datos relacionales utilizan SQL como su interfaz estándar. Puesto que todas comparten la misma interfaz, los desarrolladores pueden escribir software sobre una base de datos y mas adelante cambiar a otra base de datos (por razones económicas, disponibilidad, etcétera) sin tener que cambiar la aplicación.

Esa es la teoría, pero en la práctica, las pequeñas diferencias, extensiones de distribuidores y otras cuestiones indican que cambiar de un distribuidor a otro requiere de más trabajo que sólo volver a compilar la aplicación. Pues incluso, aun siguiendo los mismos estándares los distribuidores tienden a introducir distintos errores.

Por una parte es aun más sencillo adaptar una aplicación de una versión del estándar modificada por un distribuidor, a otra versión modificada por otro distribuidor del mismo estándar, que portar la aplicación entre distribuidores que utilicen interfaces completamente distintas. Además, los estándares facilitan el aprendizaje de nuevas herramientas. Es más fácil aprender una nueva interfaz cuando el 90 por ciento de la misma es similar a la interfaz de otro producto.

Los dos métodos distintos para las interfaces entre la aplicación y el analizador se traducen en dos estándares diferentes:

1. El estándar para las interfaces basadas en objetos es DOM (Modelo de Objetos de Documento) publicado por el W3C (www.w3.org/TR/REC-DOM-Level-1).

2. El estándar para las interfaces basadas en eventos es SAX, API desarrollada en conjunto por los miembros de la lista de correo XML-DEV y editada por David Megginson (www.megginson.com/SAX).

Los dos estándares realmente no se oponen entre si, ya que sirven para distintas necesidades (ver Figura 2.1). Muchos analizadores ofrecen soporte para ambas interfaces.

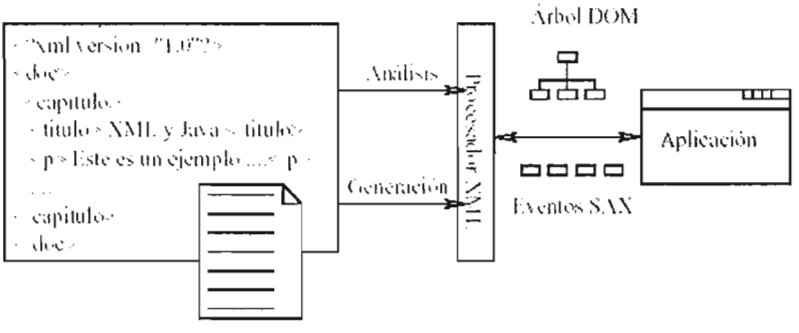


Figura 2.1: Analizadores

2.5. DOM
Modelo de Objetos de Documento

Originalmente, el W3C desarrolló a DOM para los navegadores. DOM surgió de un intento por unificar los modelos de objetos de Netscape Navigator 3 y de Internet Explorer 3. La recomendación DOM soporta tanto los documentos XML como los de HTML.

El Modelo de Objetos de Documento o DOM, es la interfaz que permite acceder y manipular, mediante la programación, el contenido de un documento o de una página web. Proporciona una representación estructurada, orientada a objetos, de los elementos individuales y el contenido de una página con métodos para recuperar y fijar las propiedades de dichos objetos. Además, proporciona métodos para agregar y eliminar dichos objetos permitiendo crear contenido dinámico[21].

El DOM proporciona también una interfaz para trabajar con eventos, permitiendo capturar y responder a las acciones del usuario o del navegador.

Con el fin de proporcionar cierta compatibilidad hacia atrás, se definieron diferentes niveles del estándar de DOM. Se encontrarán referencias al DOM Nivel 0 (o "DOM0") que corresponde al modelo usado por los primeros navegadores principalmente Internet Explorer y Netscape anteriores a la versión 4 de cada uno. A continuación está el DOM1 que fue adoptado en 1998 y que cubre algunas de las características introducidas en la versión 4 de los navegadores[20].

La mayoría de los navegadores actuales (versión 5 y superior) soportan el estándar DOM2, o al menos una buena parte de éste. Además, continúan soportando algunas características de los anteriores niveles del DOM, o sus propias extensiones propietarias, por lo que son compatibles con páginas web antiguas.

La posición de DOM como recomendación oficial por parte del W3C significa que la mayoría de los analizadores ofrecen soporte para el mismo. DOM también se implementa en navegadores, lo que significa que puede escribir aplicaciones DOM con un navegador y JavaScript.

DOM define clases de objetos para representar cada elemento en un documento XML. Existen objetos para elementos, atributos, entidades, texto, etcétera.

2.5.1. El Árbol del Documento

Cuando se analiza un documento XML con DOM se crea una representación jerárquica de su contenido. Esto desemboca en una organización parecida a un árbol de nodos, cada uno representando un elemento, un atributo, contenido o algún otro objeto.

A continuación se muestra un ejemplo de la representación de un documento XML con DOM:

```
<?xml version="1.0">
<mensajes>
  <mensaje>
    <para>ti@tu.casa.es</para>
    <de>mi@mi.casa.es</de>
    <asunto>XML </asunto>
    <urgente/>
    <texto>
      <!--comentario inutil-->X
      XML es un lenguaje de marcado...
      &amp;
    </texto>
```

```
  </mensaje>
</mensajes>
```

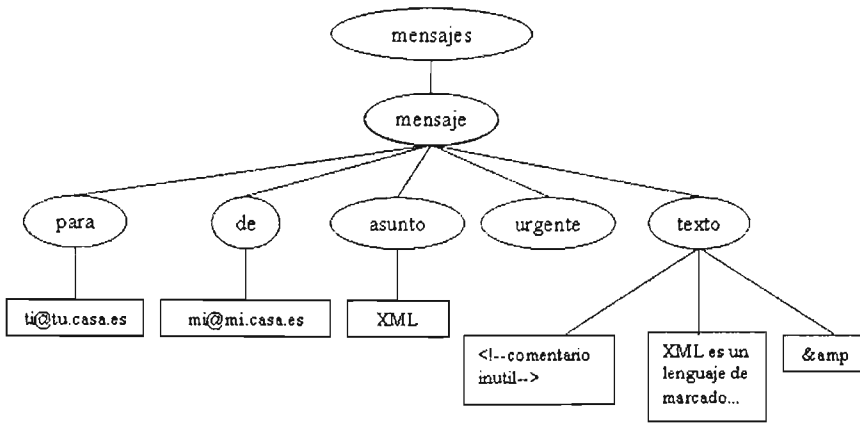


Figura 2.2: Árbol DOM

El objeto Node

Cada uno de los tipos diferentes de objetos tendrá sus propios y únicos métodos y propiedades. Pero cada tipo implementará la interfaz *Node*. Esta interfaz es un conjunto común de métodos y propiedades relacionadas con la estructura de árbol del documento.

Node define varias propiedades que sirven para recorrer el árbo[20]:

- *nodeType* es código que representa el tipo del objeto (ver Tabla 2.1).
- *parentNode* es el padre (si es que hay uno) del objeto Node actual.
- *childNode* es la lista de hijos para el objeto Node actual.
- *firstChild* es el primer hijo del objeto Node.
- *lastChild* es el último hijo del objeto Node.
- *previousSibiling* es el nodo que sigue inmediatamente del actual(hermano izquierdo).
- *nextSibiling* es el nodo que precede inmediatamente del actual(hermano derecho).
- *attributes* es la lista de atributos, en caso de que el objeto Node actual tenga alguno.

Tipo	Código
Element	1
Attribute	2
Text	3
CDATA section	4
Entity reference	5
Entity	6
Processing instruction	7
Comment	8
Document	9
Document type	10
Document fragment	11
Notation	12

Cuadro 2.1: Código nodeType de DOM

Además, *Node* define dos propiedades para manipular al objeto actual:

- *nodeName* es el nombre del nodo (para un elemento es el nombre de la etiqueta).
- *nodeValue* es el valor del nodo (para un nodo de texto viene siendo el texto en si).

La interfaz *Node* también proporciona métodos para añadir, actualizar y eliminar nodos dinámicamente, tales como:

- *insertBefore()*
- *replaceChild()*
- *appendChild()*
- *cloneNode()*

El objeto NodeList

El objeto *NodeList*, es un objeto DOM que contiene una lista de objetos Node. Dicho objeto tiene dos propiedades[20]:

- *length*, el número de nodos de la lista.
- *item(i)*, un método para tener acceso al nodo i de la lista.

El objeto Document

El elemento superior en un árbol DOM es *Document*. Implementa también la interfaz *Node*. Va a tener nodos hijos pero no tendrá nodo padre ni nodos de su mismo nivel, dado que él es el nodo inicial. Además, agrega nuevas propiedades:

- *documentElement* es el elemento superior en el documento.
- *doctype* es el tipo del documento. DOM nivel 1 no especifica completamente el tipo del documento. Esto se hará en DOM nivel 2.

Esta interfaz proporciona métodos para acceder y crear otros nodos en el árbol del documento. Algunos métodos son:

- *getElementById()*
- *getElementsByTagName()*
- *createElement()*
- *createAttribute()*
- *createTextNode()*

Notar que, de modo distinto a los otros nodos, sólo hay un objeto *Document*. Todos los métodos anteriores, excepto *getElementsByTagName()*, pueden utilizarse con el objeto *Document* usando la sintaxis *document.methodName()*.

Para devolver un árbol, el analizador devuelve un objeto *Document*. Desde el objeto *Document* es posible tener acceso al árbol completo.

2.5.2. Recorrido del árbol del Documento

Como se ha mencionado, el árbol del documento refleja la estructura del documento. Cada etiqueta o par de etiquetas está representada por un nodo elemento con otros nodos representando atributos o datos de caracter (texto).

Para extraer información o manipular de cualquier otra forma el documento, la aplicación recorre el árbol.

2.5.3. Tipos de Nodos

Como se ha mencionado, hay varios tipos de nodos definidos en el modelo de objetos de documento, pero los que se utilizan con más frecuencia son *element*, *text* y *attribute*.

Los nodos *Element*, corresponden a las etiquetas individuales o pares de éstas. Éstas pueden tener nodos hijos que pueden ser otros elementos o nodos

de texto.

Los nodos *Text* representan contenido, o datos de caracter. Van a tener un nodo padre y, posiblemente, nodos del mismo nivel, pero no pueden tener nodos hijos.

Los nodos *Attribute* son un caso especial. No están considerados una parte del árbol del documento, no tienen un nodo padre(*parent*), ni nodos hijos(*children*) o nodos del mismo nivel (*siblings*). En lugar de ello, son utilizados para posibilitar el acceso a los atributos de un nodo elemento.

El objeto Element

Element es el descendiente de *Node* que se utiliza específicamente para representar elementos de XML. Además de las propiedades heredadas de *Node*, *Element* define la propiedad tagName con base en su nombre de etiqueta.

Element también define métodos específicos[20]:

- *getElementsByTagName()* devuelve un objeto NodeList que es la lista de todos los descendientes del elemento con un nombre de etiqueta dado.
- *normalize()* reorganiza los nodos de texto que están por debajo del elemento de manera que se encuentren separados solo por etiquetas.

Nótese que los valores de los atributos son siempre cadenas de texto.

El objeto Text

Algo importante a resaltar aquí es que los nodos de texto no tienen un atributo ID como pueden tenerlos los nodos elementos, por lo tanto, no se puede acceder a ellos directamente usando métodos como *document.getElementById()* o *document.getElementsByTagName()*.

Es importante recordar que los nodos de texto contienen justamente eso, texto.

Atributos

Hay varios modos de hacer referencia a los atributos de un elemento. La razón de esto es que el estándar del DOM2 fue diseñado para muchos tipos de documentos estructurados (documentos XML), por lo que define un tipo formal de nodo para los atributos.

Pero para todos los documentos también proporciona algunos métodos más convenientes para acceder, agregar y modificar atributos.

No obstante, es más fácil acceder a los atributos de un elemento directamente usando los métodos del elemento *getAttribute()* y *setAttribute()*.

El objeto NameNodeMap

Los objetos *Element* tienen una propiedad llamada *attributes*, este es un objeto de tipo *NamedNodeMap*.

Un objeto *NamedNodeMap* es una lista de nodos que tienen nombre. Soporta las mismas propiedades y métodos de *NodeList* (*length* e *item(i)*), pero también tiene métodos especiales para acceder por nombre a los nodos[20]:

- *getNamedItem()* devuelve el nodo con el nombre dado.
- *setNamedItem()* asigna un valor al nodo con el nombre dado.
- *removeNamedItem()* elimina el nodo con el nombre dado.

El objeto Attr

Los objetos *Attr* representan los atributos. *Attr* es descendiente de *Node*. Además de las propiedades que hereda de *Node*, *Attr* define las siguientes:

- *Name* es el nombre del atributo.
- *Value* es el valor del atributo.
- *Specified* es true si el atributo recibió un valor en el documento, es false si el atributo ha tomado un valor predeterminado de la DTD.

2.5.4. DOM y Java

DOM no está limitado a los exploradores. Tampoco está limitado a JavaScript. DOM es una interfaz multiplataforma puede usarse en varios lenguajes.

DOM e IDL

Existen versiones de DOM para JavaScript, Java y C++. De hecho, existen versiones de DOM para la mayoría de los lenguajes, ya que el W3C adoptó un truco inteligente; especificó a DOM utilizando el IDL de OMG[20].

IDL es un lenguaje de especificación para interfaces de objetos. Se utiliza para describir no lo que hace un objeto, sino qué métodos y propiedades tiene. IDL (Lenguaje de Definición de Interfaz) fue publicado por el OMG (Grupo de Administración de Objetos).

Lo bueno acerca de IDL es que ha sido asignada a muchos lenguajes de programación orientados a objetos. Existen asignaciones de IDL para Java,

C++, Smalltalk, Ada e incluso para Cobol. Al escribir la recomendación DOM en IDL el W3C se beneficio de este soporte multilenguaje. En esencia, DOM está disponible en todos estos lenguajes.

Java, al igual que Java Script, es un lenguaje privilegiado para el desarrollo con XML. De hecho, la mayoría de las herramientas XML están escritas en Java y/o tienen una versión en Java. Actualmente existen más analizadores Java que analizadores escritos en cualquier otro lenguaje. La mayoría de estos analizadores ofrecen soporte para la interfaz DOM[22].

El analizador DOM de Java

Hay analizadores de Java disponibles en Microsoft (www.microsoft.com), Oracle (www.oracle.com) en Sun (java.sun.com). El analizador Sun se conoce como ProjectX .

La recomendación DOM empieza con el objeto *Document*. No existe un estándar en relación con la manera de llamar al analizador. La llamada al analizador DOM desde Java se ejemplifica en la Figura 2.3 donde:

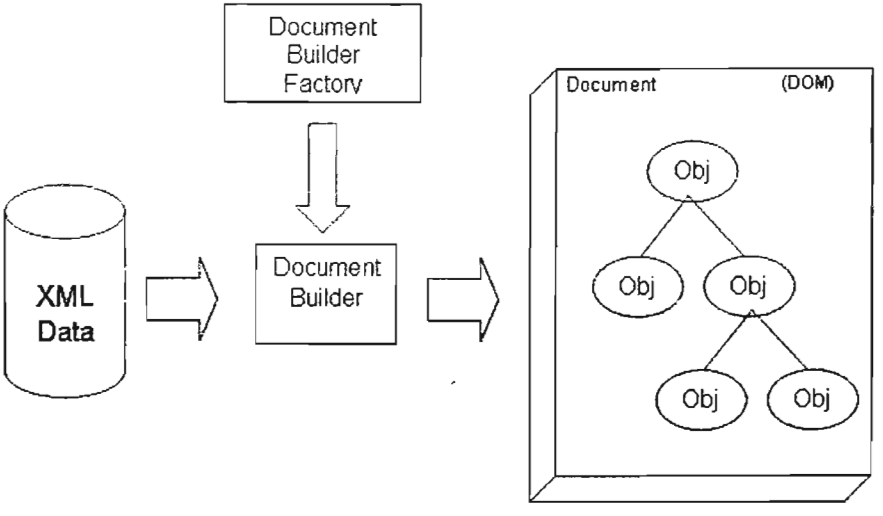


Figura 2.3: DOM desde Java

Document es el árbol que representa el contenido del documento y se obtiene con un analizador. Se encuentra en la interfaz org.w3c.dom.Document.
DocumentBuilderFactory crea un ejemplar de analizador (DocumentBuilder).Se encuentra en la interfaz javax.xml.parsers.DocumentBuilderFactory.

DocumentBuilder analiza el documento XML y genera el Document.Se encuentra en la interfaz javax.xml.parsers.DocumentBuilder.

A continuación se muestra un ejemplo de cómo llamar al analizador DOM desde Java :

```
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import org.w3c.dom.Document;

public class DOM1 {
static Document document;

public static void main(String argv[]) {
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
try {
DocumentBuilder builder = factory.newDocumentBuilder();
document = builder.parse("Equipos.xml");
} catch ...
```

2.6. SAX
API Simple para XML

Los miembros de la lista de correo XML-DEV han desarrollado una API estándar para los analizadores basados en eventos conocida como SAX, que viene siendo la abreviación de *API Simple para XML.*

SAX es editada por David Megginson y se publica en whttp://www.saxproject.org. A diferencia de DOM, SAX no está patrocinada por una organización oficial de estandarización, pero es ampliamente utilizada y se considera como un estándar[3].

Esta API consta de una serie de clases, con sus correspondientes métodos, que permite trabajar con un documento XML desde un programa escrito en Java, pudiendo acceder a los datos, comprobar si está bien formado, si es válido, etcétera.

El acceso que se proporciona en SAX es en gran parte jerárquico y a la vez secuencial. Se alcanzan las hojas del primer elemento, después se va hacia atrás y hacia arriba en el árbol, de nuevo se vuelve a bajar hasta la hoja del segundo elemento y así sucesivamente. En ningún punto hay una relación clara que diga en qué nivel de jerarquía se encuentra.

SAX proporciona un marco basado en eventos para analizar datos XML, es decir, realizar el proceso de leer detenidamente el documento, desglosando los

datos en partes utilizables; en cada paso, SAX define los eventos que pueden ocurrir[11].

SAX define una interfaz *ContentHandler* que define métodos para el manejo del contenido de un documento XML tales como *startDocument()* y *endElement()*. La implantación de esta interfaz permite un control completo sobre las partes del proceso de análisis de XML. Hay una interfaz muy parecida para el manejo de errores y construcciones léxicas. Se define un conjunto de errores y avisos, permitiendo el manejo de varias situaciones que puedan tener lugar durante el análisis XML, como un documento inválido o un documento con mal formato. Se puede añadir más funcionalidad para hacer a medida el proceso del análisis, obteniendo como resultado unas tareas muy específicas de la aplicación disponibles para su definición, todas ellas con una interfaz estándar para documentos XML.

La principal característica de SAX es que el documento se lee secuencialmente de principio a fin, sin cargar todo el documento en memoria, lo cual tiene consecuencias positivas y negativas:

- La ventaja es la eficiencia en cuanto al tiempo y la memoria empleados en el análisis.
- La desventaja es que con SAX no se dispone de la estructura en árbol del documento, luego no se puede modificar ni crear documentos XML, ni recorrerlos jerárquicamente, solamente analizarlos secuencialmente.

SAX era originalmente el primer API extensamente adoptado para XML en Java, y es un estándar. La versión actual es SAX 2.0.1, y hay versiones para varios ambientes de lenguajes de programación además de Java[3].

2.6.1. Lectores de SAX

Lo primero que se tiene que hacer es conseguir una instancia de una clase que se ajuste a la interfaz *XMLReader* de SAX. Esta interfaz define el comportamiento del análisis y permite establecer unas características y propiedades[11].

SAX proporciona una interfaz que deberían implantar todos los analizadores XML que sean compatibles con él. Esto permite conocer exactamente qué métodos están disponibles para las llamadas de retorno y para usarlos en la aplicación.

Todo analizador XML deberá tener una clase que implante la interfaz *XMLReader* y ésta es la clase que se necesita instanciar para poder analizar XML:

```
XMLReader parser = New SAXParser ( );
//...
paser.parse(uri);
```

2.6.2. Analizar un documento

Una vez que el analizador está cargado y listo para usarse, se pueden dar instrucciones para analizar el documento. El método *parse()* de *XMLReader* lo manejará convenientemente. Este método puede aceptar una fuente de entrada (*InputSource*) o una cadena de caracteres URI. Este URI podría ser una dirección accesible por la red. Se puede utilizar la ruta completa al documento XML. Si se eligiera usar un URL[2] , para acceder al documento XML a través de la red, la aplicación tendría que resolver el URL antes de pasárselo al analizador (generalmente esto sólo requiere alguna forma de conectividad de red).

Se necesita añadir al programa el método *parse()*, así como dos manejadores de excepciones. Debido a que el documento se tiene que cargar local o remotamente puede aparecer la excepción *java.io.IOException* y debe capturarse. Además se puede lanzar la excepción *SAXException* si ocurren problemas durante el análisis del documento. De esta forma se pueden añadir instrucciones *import*.

2.6.3. Utilizar una fuente de entrada

En vez de utilizar un URI completo, también se puede invocar el método *parse()* con *InputSource* como argumento. Esta clase se usa como una clase auxiliar o envolvente más que para cualquier otra cosa. Un *InputSource* encapsula la información de un objeto simple. En situaciones en las que puede que un identificador de sistema, un identificador público o un stream estén ligados a un URI, puede venir bien encapsular todo mediante un *InputSource*. La clase tiene métodos para acceder (*accessor*) y transformar (*mutator*) su identificador de sistema y su identificador público, una codificación de carácter, un stream de bytes (*java.io.InputStream*) y un stream de caracteres (*java.io.Reader*). Si se pasa como argumento al método *parse()*, SAX también garantiza que el analizador nunca modificará el *InputSource*. Esto asegura que la entrada original al analizador se mantiene inalterable después de que se use por el analizador o por una aplicación compatible con XML[11].

2.6.4. Manejadores

Para permitir que una aplicación realice algo útil con los datos XML según se van analizando, se deben declarar *manejadores* con el analizador SAX. Un manejador no es más que un conjunto de llamadas de retorno que SAX define para agregar al código de la aplicación en eventos importantes, durante el

[2]URL: Uniform Resource Locator. "Localizador de Recursos Uniforme". Es el modo estándar de proporcionar la dirección de cualquier recurso en Internet, que es parte de la WWW. Las URLs, pueden ser absolutas o relativas. Una URL absoluta consiste en un prefijo que denota un método ("http"para puntos de Internet, "gopher"para gophers, "ftp"para trasferencia de ficheros, etcétera.), seguido por dos puntos y dos barras (://), una dirección , que consiste en un nombre de dominio seguido por una barra, un nombre de vía, y un ancla opcional precedido por un símbolo * que apunta a un lugar dentro de una página Web.

análisis de un documento. Estos eventos se producirán cuando el documento se analice, no después de que el análisis haya terminado. Esta es una de las razones por las que SAX es una interfaz tan poderosa: permite manejar un documento secuencialmente sin tener que cargar primero el documento entero en memoria.

SAX 2.0 define cuatro interfaces de manejadores principales:

1. ContentHandler.
2. ErrorHandler.
3. DTDHandler.
4. EntityResolver.

ContentHandler permite manejar eventos estándar relacionados con los datos en un documento XML, *ErrorHandler* recibe notificaciones del analizador cuando se encuentran errores en los datos XML. *EntityResolver* funciona como los otros manejadores y está construido específicamente para resolver entidades externas que se especifiquen en un documento XML. Cada una de estas interfaces puede implantarse por clases de aplicaciones que realicen acciones específicas en el proceso de análisis.

Estas clases de implantación se pueden declarar en el analizador con los métodos:

- setContentHandler()
- setErrorHandler()
- setDTDHandler()
- setEntityResolver()

Después el analizador invoca los métodos de llamada de retorno en los manejadores apropiados durante el análisis.

2.6.5. ContentHandler

Como su nombre lo indica el ContentHandler es el encargado de manejar los eventos relacionados con el contenido de un documento XML, tales como los elementos, atributos, CDATA, etcétera.

El localizador de documento

El primer método que se necesita definir es el que pone un *Locator* para cada evento SAX.

Cuando ocurre un evento de una llamada de retorno, a menudo una clase que implante un manejador necesita acceder al lugar donde esté el analizador

de SAX en un documento XML. Esto se puede usar para ayudar a la aplicación a tomar decisiones sobre el evento y su posición en el documento XML.

La clase *Locator* tiene varios métodos útiles tales como *getLineNumber()* y *getColumnNumber()* que devuelven la posición actual dentro de un documento XML cuando se invocó. Debido a que esta posición es válida solamente para el ciclo de vida actual del análisis, el *Locator* se debería utilizar únicamente en el ámbito de la implantación de *ContentHandler*. Como es posible que se quiera usar esto más tarde, se salva la instancia *Locator* dada como una variable miembro y se muestra un mensaje diciendo que ocurrió la llamada de retorno. Esto ayudará a perfilar el estado y la ocurrencia de los eventos de SAX.

Se pueden añadir detalles a este método si se necesita seguir la información sobre el origen de los eventos. Si se quiere mostrar información sobre dónde están ocurriendo los eventos en el documento, como el número de línea en el que aparece un elemento, se debería asignar este Locator a una variable miembro para un uso posterior en la clase:

```
public void setDocumentLocator(Locator locutor){
    System.out.println(" * setDocumentLocator( ) called");
    //Se guarda por si se desea utilizar más tarde.
    this.locator = locator;
}
```

El comienzo y el final de un documento

En cualquier proceso de ciclo de vida, siempre debe haber un comienzo y un final que ocurren una sola vez. El primero antes que el resto de eventos y el último después de todos ellos.

Este hecho obvio es crítico para las aplicaciones, ya que les permite saber exactamente dónde empieza y dónde acaba el análisis. SAX proporciona métodos de llamadas de retorno para cada uno de estos eventos, *startDocument()* y *endDocument()*.

El primer método, *startDocument()* se llama antes que cualquiera del resto de llamadas de retorno, incluyendo los métodos de llamadas de retorno que se encuentren dentro de otros manejadores SAX, como por ejemplo *DTDHandler*. En otras palabras, *startDocument()* no es solamente el primer método que se llama dentro de *ContentHandler*, sino en todo el proceso de análisis completo, además del método *setDocumentLocator()*. Esto asegura al análisis un comienzo finito y permite que la aplicación realice cualquier tarea que necesite antes de que comience el análisis.

El segundo método, *endDocument()* siempre es el último método que se llama teniendo en cuenta a todos los manejadores. Aquí se incluyen las situaciones en

las que tienen lugar errores que provocan que el análisis se pare. Habrá dos tipos de errores: recuperables y no recuperables. Si ocurre un error irrecuperable, se invoca el método de llamada de retorno de *ErrorHandler* y, posteriormente una llamada final a *endDocument()* completará el intento de análisis. Ambos métodos de llamada de retorno pueden lanzar excepciones *SAXExceptions*. Éstas son el único tipo de excepciones que los eventos SAX lanzan y proporcionan otra interfaz estándar para el comportamiento de análisis. Sin embargo, estas excepciones a menudo envuelven otras excepciones que indican que están ocurriendo problemas. Por ejemplo, si un documento XML se está analizando a través de la red vía un URL, y la conexión se vuelve de repente no válida, ocurrirá una excepción *IOException*, sin embargo, una aplicación que utilice las clases SAX no debería capturar esta excepción, porque no debería saber dónde se encuentra el recurso XML. En vez de esto, la aplicación puede capturar la excepción *SAXException*. Dentro del analizador SAX, se captura la excepción original y se vuelve a lanzar con una excepción *SAXException*, incluyendo a la excepción original dentro de la nueva. De esta forma se logra que las aplicaciones tenga una excepción estándar que capturar, a la vez que se permiten detalles específicos del error ocurrido dentro del proceso de análisis envolviéndolos y haciéndolos disponibles al programa llamante mediante esta excepción estándar. La clase *SAXException* proporciona un método, *getException()*, que devuelve la excepción (*Exception*) original.

```
public void startDocument ( ) throws SAXException {
    System.out.println("Comienza parseo...");
}
public void endDocument ( ) throws SAXException {
    System.out.println("...Termina parseo.");
}
```

Instrucciones de proceso

No se consideran elementos XML y se manejan de forma diferente pasándolas a la aplicación que llama. Debido a estas características especiales, SAX define una llamada de retorno específica para manejar instrucciones de proceso. Este método recibe el destino de la instrucción de proceso y cualquier dato enviado a la instrucción de proceso:

```
public void processingInstruction(String target, String data)
                                    throws SAXException {
    System.out.println("proccesingInstruction: Etiqueta:" + target +
                       " y datos:" + data);
}
```

En una aplicación real que utilice datos XML, se pordría recibir instrucciones y fijar los valores de las variables o ejecutar métodos para realizar el proceso específico de la aplicación. Este método, como las otras llamadas de retorno de

SAX, lanza una excepción *SAXException* cuando tiene lugar un error.

En cuanto a lo que se refiere a las declaraciones XML, estas instrucciones de proceso especiales dan la versión, la información opcional sobre la codificación del documento e información sobre si el documento es autónomo:

```
<?xml versión="1.0" encoding="UTF-8" standalone="yes"?>
```

Ésta es una instrucción específica para el analizador XML que le permite informar, al inicio del análisis sobre un error, como por ejemplo una versión que no se soporta.

Como esta instrucción no está destinada únicamente a ser utilizada por el analizador, no inicia una llamada de proceso a *processingInstruction()*. Hay que asegurarse de no construir código para aplicaciones que espere esta instrucción o información de versión, porque la aplicación nunca recibirá una llamada de retorno de esta instrucción de proceso. De hecho, sólo debería ser el analizador el que tuviera interés en la codificación y versión de un documento XML, ya que estos elementos se utilizan en el análisis. Una vez que tenga acceso a los datos mediante la API de Java, estos detalles serán irrelevantes.

Espacio de nombres

Los espacios de nombres XML tienen gran importancia y un gran impacto en el análisis y manejo de XML. Junto con los esquemas XML, los espacios de nombres XML son de los conceptos más significativos añadidos a XML desde la recomendación original XML 1.0. Con el SAX 2.0 se introdujo el soporte para espacios de nombres al nivel de elemento. De esta forma, se permite hacer una distinción entre el espacio de nombres de un elemento, definido por un prefijo de elemento y el URI del espacio de nombre asociado y el *nombre local* de un elemento. En este caso se utiliza nombre local para referir al nombre de un elemento sin prefijo. Por ejemplo, el nombre local de JavaXML:Book es sencillamente Book. El prefijo de espacio de nombres es JavaXML.

Hay dos llamadas de retorno SAX específicas que tratan con espacios de nombres. Estas llamadas de retorno se invocan cuando el analizador llega al inicio y al final de un *mapeo de prefijo*. Un mapeo de prefijo es simplemente un elemento que utiliza el atributo *xmlns* para declarar un espacio de nombres. A menudo éste es el elemento raíz (que puede tener múltiples mapeos), pero puede ser cualquier elemento dentro de un documento XML el que declare un espacio de nombres explícito. Por ejemplo:

```
<root>
    <element1>
        <myNamespace:element2 xmlns:myNamespace=http://myURL.com>
        <myNamespace:element3>Aqui van datos</myNamespace:element3>
        </myNamespace:element2>
```

```
    </element1>
</root>
```

En este caso, se declara un espacio de nombres explícito tras varios anidamientos de elemento en el documento.

La llamada de retorno *startPrefixMapping()* recibe el prefijo del espacio de nombres, así como el URI asociado con ese prefijo. El mapeo se considera cerrado o finalizado cuando el elemento que declaró el mapeo se cierra. La única direrencia a esta llamada de retorno es que no se comporta completamente de la misma manera secuencial en la que SAX está estructurado; la llamada de retorno de mapeo de prefijo tiene lugar directamente antes de la llamada de retorno para el elemento que declara el espacio de nombres:

```
public void startPrefixMapping(String prefix, String uri){
    System.out.println("Mapping starts for prefix " + prefix +
                       " mapped to URI " + uri);
}
```

Si el mapeo se declara como un atributo del elemento raíz, se debería esperar a ser invocada esta llamada de retorno antes que la llamada de retorno del primer elemento, pero después de la llamada de retorno a *startDocument()* así como a cualquier instrucción de proceso que se tiene en la parte superior del documento. Las llamadas de retorno de espacio de nombres que se invocan para indicar el final del mapeo deben aparecer directamente después de la etiqueta de cierre del elemento que declara el mapeo:

```
public void endPrefixMapping(String prefix){
System.out.println ("endPrefixMapping " + prefix);
}
```

Elementos

Más de la mitad de las llamadas de retorno de SAX no tienen nada que ver con los datos XML, como elementos y atributos. Esto es porque el propósito del proceso de análisis XML va más alla de sólo proporcionar datos XML a su aplicación; debé proporcionar a la aplicación las acciones que debe tomar con las instrucciones de proceso XML, también debe indicar cuándo empieza y termina el análisis e incluso avisar cuando hay un espacio en blanco que debe ser ignorado.

Todavía hay algunas llamadas de retorno de SAX que tienen como propósito dar acceso a los datos XML en los documentos. Los tres eventos primarios que interesan a las Interfaces SAX para obtener los objetos de datos, son el principio y el fin de los elementos y la llamada de retorno *characters()*. Estos dicen cuándo se analiza un elemento, los datos de ese elemento y cuándo se alcanza

la etiqueta de cierre para ese elemento.

El primero de ellos, *startElement()* da a una aplicación la información sobre un elemento XML y los atributos que pueda tener. Los parámetros para esta llamada de retorno son el nombre del elemento y una instancia *Attributes*. Esta clase de ayuda tiene referencias a todos los atributos de un elemento. Permite una iteración sencilla por los atributos del elemento de una forma similar a un vector. Además de poder hacer referencia a un atributo por su índice (se usa cuando se itera entre todos los atributos), es posible referenciar un atributo por su nombre. Por supuesto, hay que tener cuidado cuando aparezca la palabra *"nombre"* refiriéndose a un elemento o atributo XML, ya que puede significar varias cosas. En este caso, se puede usar el nombre completo de un atributo, con un prefijo del espacio de nombre, si es que lo hay, denominado nombre sin procesar, o la combinación de su nombre local y el espacio de nombres URI si se usa un espacio de nombres. Hay también métodos de ayuda como *getURI (int index)* y *getLocalName (int index)* que ayudan a dar información adicional de los espacios de nombres sobre un atributo. Usado en su totalidad, la interfaz *Attributes* puede ser un conjunto extenso de información de los atributos de un elemento.

Además de los atributos de elemento, hay distintas formas de nombres de elemento. Esto es debido a los espacios de nombres XML. Primero se proporciona el espacio de nombre URI del elemento. Esto sitúa al elemento en su contexto correcto dentro del conjunto de espacios de nombres de todo el documento. Entonces se proporciona el nombre local del elemento, que es el nombre del elemento sin prefijo. Además se proporciona el nombre sin procesar del elemento.

Éste es el nombre del elemento sin modificar y sin cambiar que incluye un prefijo del espacio de nombres, si existe. En otras palabras, esto es exactamente lo que estaba en el documento XML, y de esta forma, sería JavaXML:Book para nuestro elemento Book. Con estos tres tipos de nombre se puede escribir un elemento con o sin relación a su espacio de nombres.

Como ejemplo una implantación de la llamada retorno SAX que muestra como hacer disponibles un elemento y sus atributos. Esta llamada imprime esta información en pantalla cuando se le invoca. En este ejemplo, se ve si el nombre de elemento tiene un espacio de nombre URI asociado; si es así, se imprime el espacio de nombre; si no, se imprime un mensaje afirmando que el elemento no tiene ningún espacio de nombre asociado. SAX hace este proceso muy simple y sencillo, se analiza la llamada de retorno *startElement()*:

```
public void startElement(String namespaceURI, String localName,
            String rawName, Attributes atts) throws SAXExcepcion {
    System.out.print("startElement: " + localName);
    if(!namespaceURI.equals("")){
        System.out.println("Espacio de nombre " + namespaceURI + "
```

```
                        ("+rawName + ")");
    }
    else{
        System.out.println("no tiene espacio de nombre asociado");
    }
    for(int i=0; i<atts.getLength( ); i++)
        System.out.println("Atributo : " + atts.getLocalName(i) +
                           "=" +  atts.getValue(i));
}
```

Una cosa que se debe de tener en cuenta es que los atributos de la llamada *startElement()* no permanecen ordenados. Cuando se repite una implantación de *Atributes*, los atributos no estarán disponibles en el orden en que se analizaron, que es el orden en el que se escribieron. Esto significa que no es buena idea depender del orden de los atributos, ya que XML no requiere que los analizadores XML mantengan este orden.

La encargada del cierre de una llamada de retorno de elemento es el método *endElement()*. Sólo se le pasa el nombre del elemento, permitiendo que este nombre se empareje con el nombre de elemento apropiado que se pasó con anterioridad a la llamada de retorno *startElement()*. El propósito principal de esta llamada es indicar el cierre de un elemento y hacer saber a la aplicación que el resto de caracteres no son parte del elemento en curso que se está cerrando. En el siguiente ejemplo esto se puede ver debido a que se imprime el nombre del elemento cuando se cierra:

```
public void endElement(String namespaceURI, String localName,
                       String rawName) throws SAXException {
    System.out.println ("endElement: " + localName + "\n");
}
```

Datos de elemento

Una vez que se identifiquen el principio y el final del bloque de un elemento y se enumeren los atributos de los elementos para la aplicación, la próxima parte de información importante son los datos contenidos en los propios elementos. Cada uno de estos consiste, generalmente, en elementos adicionales, datos de tipo texto o combinaciones de ambos. Cuando aparecen otros elementos, se inician las llamadas de retorno de estos elementos y tiene lugar a una especie de recursión: los elementos anidados en elementos causan llamadas de retorno *anidadas* en llamadas de retorno. En algún momento se encontrarán datos de tipo texto. Normalmente ésta es la información más importante para un cliente XML, ya que éstos son los datos que se muestran en el cliente o que se procesan para generar la respuesta al cliente.

En XML los datos de tipo texto en los elementos se mandan a una aplicación llamada *characters()*. Este método proporciona un vector de caracteres, así como el índice de comienzo y de final, con los que se leerán los datos de tipo texto:

```
public void characters(char[] ch, int start, int end)
                        throws SAXException {
    String s = new String(ch, start, end);
    System.out.printfln("characters: " + s);
}
```

Aunque sea una llamada aparentemente sencilla, a veces crea confusión porque la interfaz SAX y los estándares no definen cómo se deben usar estas llamadas de retorno con grandes porciones de datos de tipo carácter. Un analizador tiene que elegir si devuelve todos los datos en una sola invocación o dividir los datos en varias invocaciones. Para un elemento dado no se llamará este método, si no hay datos de tipo carácter, o se llamará varias veces.

Finalmente, el método *characters()* informa de los espacios en blanco. Esto introduce más confusión ya que otra llamada de retorno SAX, *ignorableWhitespace*, también informa de los espacios en blanco.

Así, la forma en que se informa de los espacios en blanco depende de si el analizador puede validar o no.

Los analizadores que validan informarán de los espacios en blanco a través del método *ignorableWhitespace()*. Los analizadores que no validan, informan con el método *ignorableWhitespace()* o con el método *characters()*. Para determinar la diferencia, se deberá consultar la documentación del analizador que se esté utilizando, y por último, se debería evitar la utilización de espacios en blanco para representar datos y no se deberían hacer suposiciones sobre que llamadas de retorno del documento informarán de los espacios en blanco.

Espacios en blanco

Para esto se necesita añadir la última llamada de retorno SAX a la clase *MyContentHandler*. El método *ignorableWhitespace()* toma los parámetros exactamente en el mismo formato que el método *characters()* y deberá usar los índices de inicio y final proporcionados para leer el vector de caracteres:

```
public void ignorableWhitespace(char[] ch, int start, int end)
                                throws SAXException{
    String s = new String(ch, start, end);
    System.out.println("ignorableWhitespace : [" + s + "]");
}
```

Se informará de los espacios en blanco de la misma manera que los datos de tipo carácter; se pueden manejar en una llamada de retorno, o un analizador

SAX puede separar los espacios en blanco e informar de ellos en varias invocaciones de método.

Entidades omitidas

Los analizadores que están más establecidos no omiten entidades, esto existe para que los analizadores lo usen, pero es muy difícil encontrar un caso donde se vea.

SAX 2.0 tiene una llamada de retorno que ocurre cuando un analizador que no valida se salta una entidad. La llamada de retorno da el nombre de la entidad que será incluida en la salida:

```
public void skippedEntity(String name) throws SAXException {
    System.out.println("skippedEntity " + name);
}
```

2.6.6. Error Handler

Además de proporcionar la interfaz *ContentHandler* para manejar los eventos del análisis, SAX proporciona una interfaz *ErrorHandler* que se puede implantar para tratar varias condiciones de error que pueden surgir durante el análisis. Esta clase funciona de la misma manera que el manejador de documentos, pero sólo define tres métodos de llamadas de retorno. Mediante esos tres métodos los analizadores SAX son capaces de manejar e informar de todas las posibles condiciones de error[11].

Cada método recibe información sobre el error o advertencia que ha tenido lugar mediante una excepción *SAXParseException*. Este objeto contiene el número de línea en el que se encontró el problema, el URI del documento que se está tratando, que puede ser el documento analizado o una referencia externa dentro de ese documento, y detalles normales de la excepción como un mensaje y una traza imprimible de la pila (stack). Además, cada método puede lanzar una excepción *SAXException*. Esta excepción podría ser una advertencia que no debería provocar que el proceso de análisis se pare o un error que necesita solucionarse para que el análisis continúe; sin embargo, la llamada de retorno puede necesitar realizar una operación de entrada/salida del sistema u otra operación que puede lanzar una excepción, y necesita tener la capacidad de propagar esta excepción en la cadena de la aplicación. Esto se puede lograr mediante la excepción *SAXException*, el método que tiene permitido lanzar. Ejemplo: Manejador de errores que puede lanzar una excepción *SAXException*:

```
public void warning(SAXParseException exception)
throws SAXException{
    try {
        FileWriter fw = new FileWriter("error.log");
        BufferedWriter bw = new BufferedWriter(fw);
```

```
        bw.write ("Warning: " + exception.getMessage()+"\n);
        bw.flush( );
        bw.close( );
        fw.close( );
    }
    catch (Exception e){
        throw new SAXException("No se puede escribir en
                                el archivo", e);
    }
}
```

Se debe definir el esquema del manejador de errores y declararlo en el analizador de la misma forma que se declaró el manejador de documento. Lo primero que se debe hacer es añadir la clase *SAXParseException* y la interfaz *ErrorHandler* a las instrucciones *import*.

Después se debe crear una clase dentro del mismo archivo Java para hacer efectiva la interfaz *ErrorHandler* definidas por SAX.

```
class MyErrorHandler implements ErrorHandler {
    public void warning (SAXParseException exception)
     throws SAXException {
    }
    public void error(SAXParseException exception)
     throws SAXException{
    }
    public void fatalError(SAXParseException exception)
      throws SAXException{
    }
}
```

Finalmente se define el manejador de errores en el analizador SAX. Esto se hace con el método *setErrorHandler()* de la interfaz *XMLReader*. Este método toma como parámetro la interfaz *ErrorHandler* o una implantación de esta interfaz:

```
ContentHandler contentHandler = new MyContentHandler( );
ErrorHandler errorHandler = new MyErrorHandler( );
try{
    XMLReader parser = new SAXParser ( );
    parser.setContentHandler(contentHandler);
    parser.setErrorHandler(errorHandler);
    parser.parse(uri);
}
catch (IOException e) {
    System.out.println("Error readin URI : " +
    e.getMessage());
```

```
}
catch (SAXException e) {
    System.out.println("Error in parsing: " +
    e.getMessage());
}
```

Advertencias

Cada vez que tiene lugar una advertencia se invoca el método *warning()* en el manejador de errores declarado. Hay varias condiciones que pueden generar una advertencia, esto indicará que aunque no se hayan violado ninguna de las reglas XML, hay algo mal o falta algo.

Todas las advertencias están relacionadas con la DTD y la validez de un documento. Se necesita definir un método simple que imprima el número de línea, el URI y un mensaje que avise cuando ocurra una advertencia. Como se quiere que las advertencias provoquen que se pare el análisis, se lanza una excepción *SAXException* y se deja que la aplicación envolvente salga liberando los recursos utilizados:

```
public void warning (SAXParseException exception)
                    throws SAXException {
    System.out.println("Warning \n" + "Line: " +
                       exception.getLineNumer() + "URI: " +
                       exception.getSystemId( ) + "Message: " +
                       exception.getMessage( ));
    throw new SAXException ("Warning encountered");
}
```

Errores no fatales

Se consideran errores no fatales aquellos que ocurren dentro del análisis y de los que éste se puede recuperar pero que constituyen una violación de alguna parte de la especificación XML. Un manejador de errores debería, por lo menos, registrar estos errores ya que son lo suficientemente serios como para que se informe al usuario de la aplicación, aunque no sean tan críticos como para provocar que el análisis se pare.

Como en las advertencias, el manejador de errores mostrará la información enviada al método de llamada de retorno y saldrá del proceso de análisis:

```
public void error(SAXParseException exception)
                  throws SAXException{
    System.out.println("Error\n" + "Line: " +
                       exception.getLineNumer() + "URI: "  +
                       exception.getSystemId( ) + "Message: " +
                       exception.getMessage( ));
```

```
    throw new SAXException ("Error encountered");
}
```

Errores fatales

Los errores fatales son aquellos que necesitan que pare el analizador. Están relacionados con que un documento no esté bien formado y hacen que cualquier análisis posterior sea técnicamente imposible. Un manejador de errores debería notificar al usuario que ha ocurrido un error fatal; estos errores pueden provocar que una aplicación pare repentinamente. Ejemplo:

```
public void fatalError(SAXParseException exception)
                       throws SAXException{
    System.out.println("Fatal Error\n" + "Line: " +
                       exception.getLineNumer() + "URI: " +
                       exception.getSystemId( ) + "Message: " +
                       exception.getMessage( ));
    throw new SAXException ("Fatal Error encountered");
}
```

2.6.7. DTDHandler

DTDHandler declara dos eventos relacionadas con el análisis de la DTD:

- *notationDecl()* notifica a la aplicación que se ha declarado una notación.
- *unparsedEntityDecl()* notifica a la aplicación que se ha encontrado una declaración de entidad no declarada[11].

2.6.8. EntityResolver

La interfaz *EntityResolver* define sólo un evento, *resolveEntity()*. El método devuelve un objeto *InputSource*.

Pocas aplicaciones necesitan implementar *EntityResolver* ya que el analizador de SAX puede resolver de antemano nombres de documentos y la mayoría de los URLs.

Capítulo 3

XML y Bases de Datos

Internet está provocando que las organizaciones acaben migrando a la Web. Por otro lado, la información es poder y los datos acaban convirtiéndose en información. Cuando un dato se convierte en información puede servir para la toma de decisiones del negocio.

Los nuevos entornos de trabajo han hecho aparecer nuevos tipos y formatos de datos como HTML o XML utilizados para la publicación de información en la Web. Sin embargo, en paralelo se tiene que seguir trabajando con los datos almacenados en las tradicionales bases de datos relacionales. En conclusión, en la organizaciones se trabaja al mismo tiempo con datos procedentes de fuentes heterogéneas más o menos estructuradas por lo que es necesario algún método intermedio que permita integrar datos con diferentes formatos para trabajar de forma conjunta con ellos.

En cuanto a los nuevos tipos de datos creados como respuesta a las necesidades de la Web, destaca XML como estándar para la representación e intercambio de información en este entorno. Una de las consecuencias de trabajar en este medio suele ser la necesidad de intercambiar información con otros sistemas y para ello XML es el lenguaje más usado. Incluso están apareciendo sistemas de almacenamiento específico para este tipo de datos, que van más allá de los documentos XML tratados de forma independiente, estas son las bases de datos nativas XML.

El mundo de las bases de datos también se ha visto afectado por los cambios, por un lado las bases de datos relacionales siguen almacenando la mayor parte de información de la organización, pero las últimas versiones de los sistemas de gestión de base de datos, se han visto obligadas a adaptarse a las nuevas necesidades, por ejemplo permitiendo al tratar y almacenar información en formato XML.

La importancia de XML en el medio Web por un lado, y la gran cantidad de información que continua siendo almacenada en las bases de datos relacionales, lleva a estudiar cómo tratar de forma conjunta datos procedentes de ambos en-

tornos. Si los datos procedentes de ambos entornos pudieran estar en el mismo formato común, XML al ser éste el lenguaje de la Web, sería más fácil integrarlos, para suuso en el intercambio de información a través de la red, como para su posterior almacenamiento y tratamiento.

Para mantener datos en una base de datos, éstos deben ser obtenidos y almacenados de manera consistente, confiable y eficiente. El uso de la infraestructura existente de bases de datos para la administración de demandas de datos XML parece ser lógico, sin embargo, se han creado nuevos productos nativos XML que manejan las demandas de datos XML en forma nativa sin la necesidad de conversiones a otras estructuras de bases de datos como la relacional(ver figura 3.1). Por otro lado, existe una gran diversidad de estrategias de almacenamiento XML, procesos de conversión, y niveles de soporte para XML con los productos líderes de bases de datos[24].

3.1. Bases de Datos XML Nativas

XML está provocando la aparición de nuevas tecnologías, nuevas clases de bases de datos están emergiendo como bases de datos XML nativas. Estas bases de datos ganan el término nativas porque almacenan los datos estructurados como XML sin la necesidad de traducir los datos a una estructura relacional o de objeto.

Las bases de datos XML nativas están diseñadas para trabajar con XQL (eXtensible Query Language), cuyo propósito es similar a SQL en una base de datos relacional. XQL está diseñado para trabajar con documentos XML jerárquicamente estructurados y puede proveer características de consulta como filtros y joins[9]. Los esquemas XML son implementados en bases de datos XML nativas para registrar reglas de almacenamiento e indexación de datos y para proveer y obtener información de almacenamiento a los mecanismos de bases de datos XML nativas. Adicionalmente, todos los objetos en una base de datos XML nativa son accesibles directamente mediante un URL. El trabajo con bases de datos XML nativas involucra dos pasos básicos:

1. Describir los datos mediante Definiciones de Tipos de Datos (Document Type Definitions, DTD) o esquemas XML.

2. Definir un nuevo esquema de base de datos XML nativa o Mapa de Datos a usar para el almacenamiento y obtención de datos.

El producto XML nativo visto como líder en el mercado es Tamino de Software AG. Al revisar las capacidades de Tamino, los beneficios de trabajar nativamente con XML se aclaran. Tamino provee tanto almacenamiento XML nativo como mecanismo de almacenamiento relacional SQL dentro del mismo producto. Esta característica permite a los usuarios consultar datos heterogéneos mediante

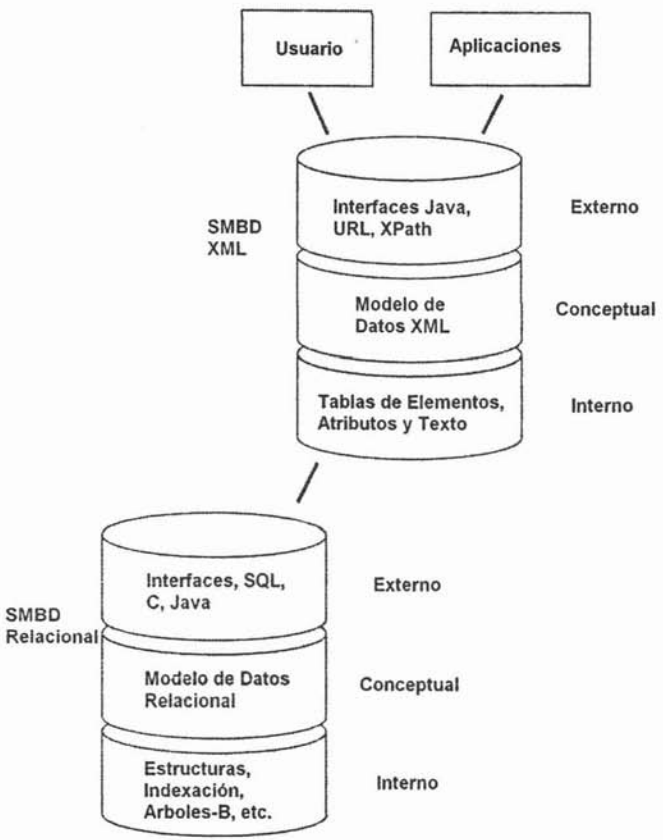


Figura 3.1: Modelos SMBD Relacional y SMBD XML

XQL y recibir conjuntos de resultados en formato XML[9].

Al almacenar y obtener documentos XML usando el mecanismo XML, no es necesaria una traducción a formato relacional. Tamino puede almacenar casi cualquier tipo de documento incluyendo información formateada XML, páginas HTML, cartas, planillas de cálculo, audio, video, imágenes y datos de bases de datos SQL o de objetos.

Existen otras bases de datos nativas XML comerciales como dbXML (http://www.dbxml eXcelon (http://www.stylusstudio.com), y x-Hive/DB (http://www.x-hive.com/). Cabe resaltar que para poder utilizar Tamino y cualquiera de las bases de datos nativas antes mencionadas se requiere de una licencia.

La opción de cambiar a una nueva base de datos XML nativa es una decisión que la mayoría de las organizaciones no hará rápidamente a menos que sea una decisión específica de una aplicación. Se ha invertido demasiado en entrenamiento, procedimientos y aplicaciones existentes para que las organizaciones abandonen su arquitectura de bases de datos actual. Además tomará algún tiempo para los vendedores de nuevas bases de datos XML nativas implementar completamente las capacidades transaccionales, la seguridad, el respaldo, la recuperación y las herramientas administrativas que proporcionan las bases de datos relacionales existentes.

3.2. Estrategias de Almacenamiento

Los documentos y los requerimientos de almacenamiento de datos XML pueden ser pensados a menudo en dos categorías generales:

- Enfocados en los datos.
- Enfocados en los documentos.

Los enfocados en datos suelen incluir por ejemplo documentos de facturación u órdenes de compra, mientras que los enfocados en documentos son menos estructurados y son apropiados para contenidos de periódicos, artículos y publicidad. Si el documento XML tiene una estructura bien definida y contiene datos actualizables usados en maneras diversas, el documento es típicamente enfocado en los datos. Los enfocados en el documento tienden a ser más impredecibles en tamaño y contenido que los enfocados en los datos los cuales son altamente estructurados, con tipos de datos de tamaño limitado y reglas menos flexibles para campos opcionales y contenido.

Los sistemas de almacenamiento XML deben acomodarse eficientemente con ambos tipos de requerimientos de datos, dado que XML está siendo ampliamente usado en sistemas que administran ambos tipos de datos. La mayoría de los productos se enfocan en servir uno de estos formatos de datos mejor que el otro. Las bases de datos relacionales tradicionales son mejores al tratar con requerimientos enfocados en los datos, mientras que los sistemas de administración de contenido y de documentos son mejores para almacenar datos enfocados en el documento.

Los documentos enfocados en los datos se pueden originar en bases de datos relacionales o como un documento XML que puede haber sido transmitido como parte de un sistema B2B[1]. Los sistemas de bases de datos deben ser capaces de exponer los datos relacionales como XML y almacenarlo recibido como datos relacionales para transferir, obtener y almacenar transparentemente los datos

[1]B2B: Business (B) to(2) Business (B), es decir, de negocio a negocio. Define el tipo de comercialización on-line que se establece de empresa a empresa.

requeridos por la aplicación.

La diferencia entre ambos tipos de documentos no siempre es fácil de identificar. Por ejemplo, una orden de compra puede contener datos tales como comentarios o notas. Mientras que, los datos enfocados en los documentos, tales como artículos de revistas, pueden tener datos estructurados como el nombre del autor y la fecha. Los requerimientos que manejan los datos se inclinan con mayor peso en una dirección o la otra y tomar el tiempo en entender si los datos son más enfocados en el documento o en los datos será una ventaja cuando se elija la estrategia de almacenamiento[9].

Con el objeto de mover datos entre un documento XML y una base de datos o viceversa, los datos en la base de datos deben ser mapeados a la estructura del documento XML. Lo más simple, es mapear el documento XML completo a una columna única en la base de datos. Estrategias más complejas incluyen el mapeo de cada elemento a una columna correspondiente en la base de datos o el mapeo de la estructura del documento XML a la base de datos.

Las estrategias existentes pueden ser descompuestas en tres métodos básicos:

1. **Almacenar el documento completo en forma de texto, como un gran objeto binario (BLOB[2]) en una base de datos relacional.** Ésta es una buena estrategia si el contenido del documento XML es estático y sólo será modificado cuando el documento completo sea reemplazado. Esta estrategia es común para datos enfocados en el documento conforme estos datos son obtenidos y almacenados como una unidad. Los tres vendedores líderes de bases de datos relacionales (Microsoft SQL Server, Oracle Oracle8i, IBM DB2) soportan este método. Almacenar el documento completo en formato de texto es fácil de implementar dado que no se necesita mapeo o traducción, pero puede limitar también la búsqueda, indexamiento y granularidad de la obtención de documentos XML. Por otro lado, el documento XML se mantiene como antes de ser almacenado, lo que minimiza el trabajo de rearmarlo luego de su obtención.

2. **Almacenar una forma modificada del documento completo en el sistema de archivos.** Este método es popular cuando el número de documentos es pequeño y los documentos XML rara vez son actualizados y transferidos entre sistemas de archivos. Nuevamente, los vendedores líderes de bases de datos soportan este método, pero tiene limitaciones obvias que incluyen escalabilidad, flexibilidad en almacenamiento y recuperación, y seguridad dado que los datos descansan fuera de la base de datos. Almacenar una forma modificada del documento completo en el sistema de archivos es bastante limitado dado que el sistema de archivos no hace una base de datos muy buena. Este método funciona para un número

[2]BLOB: Binary Large Object. Un archivo grande, tipicamente una imagen o sonido, que debe ser manejado de manera especial debido a su tamaño.

menor de documentos XML y típicamente puede ser incluido solamente en el diseño dado que el documento XML se está moviendo por el sistema de archivos. Eventualmente, los contenidos de un documento XML podrían terminar en una base de datos.

3. **Mapear la estructura de los datos en el documento en la base de datos.** Por ejemplo, mapear un documento XML que contiene una orden de ventas con tablas como Ordenes, Partes, y Clientes u objetos como Orden, Parte, y Cliente. Si la estructura del documento XML no es compatible con la estructura de la base de datos, el documento debe ser transformado para ajustarlo a la estructura de la base de datos antes de almacenarlo. Mapear la estructura de los datos en el documento a la base de datos es una opción muy popular y es el foco de muchas características habilitadoras de XML en los productos principales de bases de datos y es el tema de muchos artículos y libros en los últimos años.

3.3. Importación de documentos XML a bases de datos relacionales

Actualmente, la mayoría de las compañías a través de todas las industrias utilizan las bases de datos relacionales o SMBDR (Sistemas Manejadores de Bases de Datos Relacionales) para almacenar y para manejar su información crítica del negocio. De hecho, los tres vendedores más importantes de las bases de datos relacionales son las tres compañías más importantes del software del mundo: Oracle, IBM (DB2), y Microsoft (servidor del SQL). En paralelo, XML se ha convertido en el estándar para el intercambio de datos y de manejo de contenido. Por consiguiente, muchos millares de base de datos y los desarrolladores de aplicaciones ahora están haciendo frente al desafío del desarrollo de convertir datos XML a un formato relacional.

3.3.1. Fundamentos de RDB/XML

1. ¿Por qué son las bases de datos relacionales y XML complementarios en vez de competitivos?

   XML ha tenido un impacto tan grande en la industria de la tecnología que muchos piensan que las bases de datos XML substituirán eventualmente un RDBMS. Ahora que los profesionales han comenzado a poner soluciones viables de XML en ejecución y el primer entusiasmo y sensación generada por esta nueva tecnología ha pasado, XML y RDBMS se pueden considerar como tecnologías complementarias. De hecho, el valor traído por el uso inteligente de estas tecnologías combinadas es significativo porque sus fuerzas individuales residen en áreas muy diversas.

Las bases de datos relacionales son famosas por su capacidad de búsqueda usando SQL e índices. Almacenan datos eficientemente y sin redundancia porque cada unidad de información se guarda en un solo lugar (normalización). Su confiabilidad y escalabilidad es sin igual y pueden ser utilizadas por una gran número de usuarios concurrentemente. Además, tienen fuertes características de administración y seguridad.

Sin embargo, a pesar de iniciativas recientes de los vendedores nativos de la bases de datos XML y de la evolución de algunos nuevos estándares que emergen como XQuery, XML está lejos de la competencia como formato de bases de datos con una tecnología madura de 30 años como los RDBMS. La abstracción de instrucciones legíbles por máquinas al texto legíble por humanos no viene sin un precio, y XML sigue siendo ineficaz como mecanismo del almacenamiento y de acceso de datos. Las fuerzas de XML mienten en diversas áreas: Es texto humanamente legíble, independiente de alguna plataforma, y un estándar abierto. De hecho, se ha convertido en el lenguaje de todos los sistemas. Su naturaleza de descripción segura permite que se representen datos estructurados sin ninguna información adicional. Su estructura es inherente al documento XML, haciendo su transmisión, validación, y presentación más intuitiva que cualquier otro estándar de los datos. Puesto que la información del mundo para el futuro próximo continuará siendo salvada en almacenes relacionales de datos debido a la escalabilidad, confiabilidad, y razones del funcionamiento, ahora es importante ser capaz de mapear XML a bases de datos relacionales.

2. ¿Por qué importar documentos XML a bases de datos relacionales?

Hay un número de razones para mudar XML a un almacén relacional de datos:

a) Intercambio de datos

Los negocios a través de industrias están adoptando XML para compartir datos entre los usuarios (A2A) y las organizaciones (B2B). XML es una manera natural y persistente para que los sistemas heterogéneos de bases de datos compartan datos a través de redes. Además, está siendo adoptado rápidamente por las compañías del software como el medio estándar de enviar y recibir datos. De hecho, la mensajería XML está comenzando progresivamente a sustituir el formato propietario de EDI[3] para las transacciones de negocio a negocio. El número de las industrias que han definido o están en el proceso de definir estándares de la mensajería de XML es ya impre-

[3]EDI: Electronic Data Interchange, Intercambio electrónico de datos, un estándar para intercambio de datos, popular entre los mainframe y las grandes aplicaciones.

sionante.

La naturaleza de descripción segura de XML hace fácil intercambiar datos transaccionales entre los socios de negocio y los sistemas incompatibles. Por lo tanto, simplifica en gran parte comunicaciones de B2B y se ve como reemplazo rentable y estándar abierto para EDI.

b) Depósito de datos XML

El poder de las consultas SQL de los RDBMS puede influenciar al analizar documentos XML grandes. Si el tamaño de los archivos XML excede cierto límite o son demasiado numerosos, es ventajoso convertir la información a un formato donde puede ser recuperada con eficacia. En un sentido, los competidores más grandes de las bases de datos XML son los de las bases de datos relacionales pues RDBMS como Oracle o SQL Server están comenzando a interconectar directamente con datos XML. Desafortunadamente las interfaces de RDBMS-XML son diferentes entre provedores y por lo tanto no son portables. Además, la diferencia natural entre estas dos marcas de formatos de almacenamiento de datos actualmente es muy difícil de mapear uno al otro, y el desarrollo y el arreglo para requisitos particulares es necesario para importar documentos XML a las bases de datos relacionales.

Esto es, sin embargo, un proceso crítico, pues los negocios eligen mover datos XML a las bases de datos relacionales para consultar la cantidad grande de datos XML y para centralizar la información usada por muchas aplicaciones.

c) Abrir datos relacionales con Servicios Web

Los servicios de Web ahora experimentan el mismo interés inicial en la industria de la computación como lo hizo Java hace varios años y XML hace algunos años. Es apoyada unánimemente por todos los gigantes del software como Microsoft (Net), IBM, SUN, Oracle, HP, y otros. Los servicios de Web son accesados usando el protocolo simple de acceso a objeto basado en XML (SOAP[4]). Los datos se transforman en XML. Entonces, usando servicios Web, los datos XML pueden ser importados transparentemente a la base de datos.

3. Relacional contra la representación jerárquica de los datos

[4]SOAP: (Simple Object Access Protocol) es un protocolo estándar creado por el W3C que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML. SOAP es uno de los protocolos utilizados en los servicios Web.

Cuando se mira el mapeo de XML a una base de datos relacional, se están considerando dos estructuras de datos muy diversas.

a) Estructura de la base de datos relacional

Una base de datos relacional se describe lo mejor posible como esquema de la base de datos que sea definido por las diversas entidades (tablas) creadas para resolver los requisitos del negocio. Es representada por un sistema de tablas planas ligadas uno a uno, uno a mucho, o relaciones múltiples. Cada tabla tiene una colección fija de columnas (también llamadas los campos) que corresponden a las cualidades del modelo de los datos. Un número indefinido de filas (o de registros) ocurre dentro de cada tabla. Una base de datos relacional apoya muchos tipos de datos. Cada tabla es caracterizada por una llave primaria única, a la que otras tablas pueden referirse cuando usan una llave foránea que lleve a cabo el mismo valor. Esas llaves foráneas sirven, alternadamente, como llaves primarias para las mismas tablas. Diseñar las relaciones de las tablas relacionales para hacer que la base de datos conteste a todas las consultas necesarias del negocio y normalice todos los datos puede llegar a ser elaborado porque cada grupo de datos que no sea llave necesita estar presente en un sólo lugar (normalización).

b) Estructura de datos XML

En contraste, los datos XML son representados lo mejor posible por una estructura arborescente. Cada nodo del árbol es expresado por un elemento XML, que puede llevar a cabo unas o más cualidades tan bien como otros elementos. El grado de complejidad de este árbol es representado por una DTD (Document type definition) o un esquema de XML. La fuerza de XML puede también ser su debilidad: Puede ser un formato versátil con una estructura floja que pueda representar cualquier elemento de datos casi dondequiera. Por el contrario, puede también aparecer como estructura jerárquica rígida en otros usos. Todo depende de la DTD o del esquema XML que lo caracteriza. Sin embargo, XML es menos natural en la representación de datos relacionales porque es difícil describir el sistema de las relaciones que pueden existir entre las tablas relacionales en un documento XML. Los mismos apremios de RDBMS que lo hacen eficiente no se expresan fácilmente en XML. Por lo tanto, mapear un documento XML a una base de datos relacional es complejo en la mayoría de los casos. También requiere una cierta maestría en diseño de los datos de RDBMS y de XML y un método claro para mapear cada elemento al esquema de XML al esquema de la base de datos.

3.4. Algoritmos de Conversión entre XML y Bases de Datos Relacionales

Existen algunos algoritmos de conversión entre un esquema XML y un esquema relacional y viceversa, que varian dependiendo de la finalidad con la que fueron creados. Existen los basados en la semántica del documento, los que tienen grandes fundamentos matemáticos basados en la teoría de gramática de árbol regular, los que fueron creados únicamente para documentos que contengan una DTD y otros cuya única restricción es que cuenten con un Esquema XML.

3.4.1. Métodos de Conversión entre un esquema XML y un esquema relacional basados en la semántica

A continuación se presentan tres métodos de conversión basados en la semántica:

1. **CPI** (*Constraints preserving Inlining*): identifica varias restricciones semánticas en el esquema XML original y las preserva en el esquema relacional final[15].

2. **NeT** (*Nesting-based Translation*): deriva una estructura jerarquizada de un esquema relacional plano aplicando repetidamente el operador de jerarquía de modo que el resultado sea un esquema XML jerárquico. La idea principal es encontrar el modelo de contenido de elementos del esquema XML más intuitivamente que utilice los operadores de expresiones regulares proporcionados por la especificación de XML esquema (por ejemplo, * o +) [16].

3. **CoT** (*Constraints-based Transalation*): Aunque NeT deduce las características ocultas de datos jerarquizados, es sólo aplicable a una sola tabla a la vez. Por lo tanto, no es capaz de capturar el cuadro total del esquema relacional donde se interconectan las tablas. Para remediar este problema CoT considera dependencias de inclusión durante la traslación y combina las tablas interconectadas en una estructura coherente y jerárquica en el esquema XML final [17].

3.4.2. Algoritmo CPI

Transformar un modelo jerárquico XML a un modelo relacional plano no es una tarea trivial debido a varias dificultades inherentes tales como mapear una relación de 1-1, la existencia de un conjunto de valores, una recursión complicada, y/o problemas de fragmentación. La mayoría de los algoritmos de conversión XML-Relacional se han centrado hasta ahora principalmente en aplicar una conversión estructural, no haciendo caso en gran parte de la semántica que existe en el esquema original XML[15].

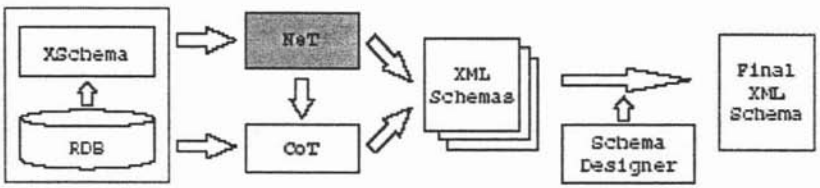


Figura 3.2: NeT y CoT: Inferir Esquemas XML del Relacional

Restricciones Semánticas en DTDs

Restricciones de Cardinalidad: En una declaración DTD, hay solamente 4 posibles relaciones de cardinalidad entre un elemento y sus sub-elementos según lo ilustrado a continuación:

```
<!ELEMENT article (titulo, autor+, ref*, precio?)>
```

1. (0,1): Un elemento puede tener cero o un sub-elemento (por ejemplo, sub-element precio).

2. (1,1): Un elemento debe tener únicamente sólo un sub-elemento (por ejemplo, sub-element titulo).

3. (0,N): Un elemento puede tener cero o más de un sub-elemento (por ejemplo, sub-element ref)

4. (1,N): Un elemento puede tener uno o más sub-elementos (por ejemplo, sub-element autor).

De estas relaciones de cardinalidad, se pueden deducir tres restricciones importantes. La primera es que un sub-elemento puede ser o no nulo. Se utiliza la notación 'X $\rightarrow \emptyset$'para denotar que un elemento X no puede ser nulo. Esta restricción es fáci de cumplir por la cláusula NULL o NOT NULL en SQL. La segunda es que un sub-elemento ocurre únicamente sólo una vez y es una clase de dependencia de generación-igualdad. La tercera, es que dado un elemento, un sub-elemento debe o no ocurrir. Ésta es una clase de dependencia de generación-tupla.

Dependencias de Inclusión (INDs): Una *dependencia de inclusión* asegura que los valores en las columnas de un fragmento deben también aparecer como valores en las columnas de otros fragmentos y es una generalización de la noción de *integridad de referencial.*

Una forma trivial de INDs encontrada en una DTD es que dado un elemento X y su sub-elemento Y. Y debe estar incluido en X ($Y \subseteq X$). La mejor forma

de representar IND's es meidante una "llave foránea"si el atributo que es referenciado es una llave primaria. Si no, se necesita usar CHECK ASSERTION o TRIGGERS de SQL.

Dependencias de Generación-Igualdad (EGDs): La restricción *Singleton* restringe un elemento a tener a lo mas un sub-elemento. Cuando un tipo de elemento X satisface la restricción *singleton* hacia su sub-elemento tipo Y, si un elemento x instanciado por el tipo X tiene dos sub-elementos *y1* y *y2* que instanciados por el tipo Y, entonces *y1* y *y2* deben ser el mismo. Esta propiedad es conocida como Dependencia de Generación-Igualdad (EGDs) y es denotada por $X \rightarrow Y$ en teoría de base de datos. Este tipo de EGDs se puede representar con una construcción UNIQUE de SQL. En general, EGDs ocurren en el caso de mapear las restricciones de cardinalidad (0,1) y (1,1).

Dependencia de Generación-Tupla (TGDs): TGDs en un modelo relacional requiere que algunas tuplas de una cierta forma estén presentes en la tabla y usa el símbolo '$\rightarrow$'. Dos formas usuales de TGDs de DTD son las restricciones de hijo y padre:

1. Restricciones Hijo: Padre $\rightarrow$ Hijo establece que cada elemento de tipo *Padre* debe tener al menos un elemento de tipo *Hijo*. Este es el caso del mapeo en las restricciones de cardinalidad (1,1) y (1,N).

2. Restricción Padre: Hijo $\rightarrow$ Padre establece que cada elemento de tipo *Hijo* debe tener un elemento de tipo *Padre*. De acuerdo a la especificación XML, los documentos XML pueden comenzar de algún nivel de elementos sin necesidad de especificar su elemento padre cuando un elemento raíz es especificado por <!DOCTYPE root>.

Relación	Simbolo	No NULL	EGDs	TGDs
(0,1)	?	no	si	no
(1,1)		si	si	si
(0,N)	*	no	no	no
(1,N)	+	si	no	si

Cuadro 3.1: Relaciones de Cardinalidad y sus restricciones semánticas correspondientes

Descubrir y preservar las restricciones de Semántica de DTDs

El algoritmo CPI identifica varias de las restricciones de semántica descritas anteriormente. Utiliza un algoritmo hibrido como la base del algoritmo. CPI primero construye un *grafo de la DTD* que representa la estructura de una DTD dada. Un grafo de la DTD puede ser construido cuando se analiza la DTD

dada. Sus nodos son elementos, atributos u operadores en la DTD. Cada elemento aparece exactamente una sola vez en el grafo, mientras los atributos y operadores aparecen tantas veces como aparecen en la DTD. Entonces CPI anota varias relaciones de cardinalidad (ver Tabla 3.1) entre nodos a cada borde del grafo de la DTD. Notar que los tipos de relaciones de cardinalidad en el grafo no sólo consideran relaciones elementos y sub-elemento sino también relaciones elementos y atributos.

Una vez que se construye el grafo anotado de la DTD, el CPI sigue el método básico de la navegación proporcionado por el *algoritmo híbrido*; identifica los *nodos superiores*[15] que son los nodos:

1. no accesibles de cualquier nodo o
2. el nodo hijo directo con operador (* o +) o
3. nodo recursivo con grado > 1 o
4. un nodo entre dos nodos mutuamente recursivos con grado $= 1$.

Entonces, saliendo de cada nodo superior T, se recorren todos los elementos y atributos en los nodos hojas accesibles desde T a menos que sean otros nodos superiores. Así, hasta que cada relación de cardinalidad anotada se puede convertir correctamente a sus contrapartes en sintaxis del SQL según se describió anteriormente.

Ejemplo

Dada la siguiente DTD:

```
<!ELEMENT conf (titulo,fecha,editor?,articulo*)>
<!ATTLIST conf id ID #REQUIRED>
<!ELEMENT titulo (#PCDATA)>
<!ELEMENT fecha EMPTY>
<!ATTLIST fecha año CDATA #REQUIRED
mes CDATA #REQUIRED
dia CDATA #IMPLIED>
<!ELEMENT editor (person*)>
<!ATTLIST editor eids IDREFS #IMPLIED>
<!ELEMENT articulo (titulo,contacto?,autor,cita?)>
<!ATTLIST articulo id ID #REQUIRED>
<!ELEMENT contacto EMPTY>
<!ATTLIST contacto aid IDREF #REQUIRED>
<!ELEMENT autor (persona+)>
<!ATTLIST autor id ID #REQUIRED>
<!ELEMENT persona (nombre,(email|telefono)?)>
<!ATTLIST persona id ID #REQUIRED>
```

```
<!ELEMENT nombre EMPTY>
<!ATTLIST nombre primer_nombre CDATA #IMPLIED
apellido CDATA #REQUIRED>
<!ELEMENT email (#PCDATA)>
<!ELEMENT telefono (#PCDATA)>
<!ELEMENT cita (articulo*)>
<!ATTLIST cita id ID #REQUIRED
formato (ACM|IEEE) #IMPLIED>
```

Se generá el grafo de la DTD que se muestra en la Figura 3.3

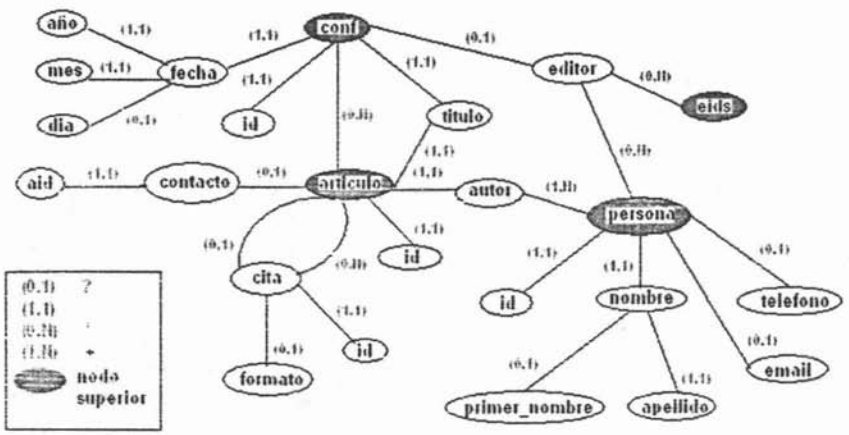


Figura 3.3: Grafo de DTD

Y se obtiene el siguiente esquema relacional:

```
CREATE TABLE articulo (
  id NUMBER NOT NULL,
  titulo VARCHAR(50) NOT NULL,
  contacto VARCHAR(20),
  cita_id VARCHAR(20),
  cita_formato VARCHAR(50) CHECK (VALUE IN ("ACM", "IEEE")),
  root_elm VARCHAR(20) NOT NULL,
  parent_elm VARCHAR(20),
  fk_cita VARCHAR(20) CHECK (fk_cita IN (SELECT cita_id FROM articulo)),
  fk_conf VARCHAR(20),
  PRIMARY KEY (id),
  UNIQUE (cita_id),
  FOREIGN KEY (fk_conf) REFERENCES conf(id),
  FOREIGN KEY (contacto_aid) REFERENCES persona(id)
```

);

Como se puede observar este algoritmo sólo funciona si se tiene la DTD del documento XML que se quiere almacenar, además de que para cada DTD crea una base de datos específica.

3.4.3. Conversión de XML a Relacional utilizando la Teoría de Gramática Árbol Regular

Esta conversión está basada en la Teoría de Gramática Árbol Regular [23], que proporciona un marco de trabajo útil y formal para comprender varios aspectos de los lenguajes de esquemas XML. Se introducen dos representaciones de forma normal para RTG's: denominadas NF1 y NF2. [18]

NF1, se usa como primer paso en el proceso de validación por razones de eficiencia y para verificar si un esquema determinado satisface las restricciones estructurales impuestas por el lenguaje de esquema.

NF2, forma la base para la conversión de un conjunto de definiciones de tipos en un lenguaje de esquema L1 que soporta unión de tipos, a un lenguaje de esquema L2, que no soporta unión de tipos. También se usa como el primer paso en el algoritmo de conversión de Esquema XML a Esquema Relacional.

El enfoque que se proporciona es diferente de los existentes ya que se consideran las restricciones semánticas de XML y como consecuencias estas se capturan en el Modelo Relacional que resulta. La conversión está basada en fundamentos matemáticos fuertes y claros proporcionados por las RTG's.

A grandes rasgos el algoritmo propuesto para realizar la conversión es:

- Simplificación del esquema, donde se obtiene un esquema XML más simple, que no tiene las restricciones que no pueden capturarse en el Modelo Relacional.
- Recorrrido de elementos y atributos, donde se usan heurísticas simples para determinar cuales son los atributos de una relación.
- Mapeo de tipos colección, los tipos colección pueden representarse usando * o +; se consideran como relaciones de 1: muchos y se mapean como tales.
- Captura del orden especifico en el modelo XML.
- Aplicación de restricciones de claves y dependencias.

Este algoritmo se probó y el resultado obtenido fueron buenos esquemas relacionales. Después se tomaron los datos y se convirtieron a XML usando el algoritmo CoT y se probó que cuando se convierten los datos XML resultantes

de CoT a relaciones se obtendrá el esquema Relacional original con el que se inicia[18].

Para poder utilizar este algoritmo es necesario contar con el Esquema XML[19] del documento XML que se desee almacenar y está basado en la Teoria de Gramática de Árbol Regular con fundamentos matemáticos formales.

3.4.4. Algoritmo de Conversión de XML con una DTD a BDR

Regla 1: Crear una llave primaria. Crear siempre una tabla en la base de datos relacional con el nombre del elemento:

- Agregar una columna para que contenga un entero autoincrementable.
- El nombre de la columna será el nombre del elemento junto con la palabra *'Key'*.
- Fijar esta columna para que sea la llave primaria en la tabla creada.

Regla 2: Creación de la tabla básica. Para cada elemento estructural encontrado en el DTD:

- Crear una tabla en la base de datos relacional.
- Si el elemento estructural tiene exactamente un elemento permitido por el padre (o es el elemento raíz de la DTD), agregar una columna en la tabla. Esta columna será una llave foránea que referencia a un elemento padre.
- Hacer la llave foránea requerida.

Regla 3: Manipulación de elementos padres múltiples. Si un elemento particular puede tener más que un elemento padre, y el elemento puede ocurrir en el padre cero o una vez:

- Agregar una llave foránea a la tabla que representa el elemento padre que apunte al registro correspondiente a la tabla hijo, haciéndolo opcional o requerido.
- Si el elemento puede ocurrir cero, una o más veces, agregar una tabla intermedia a la base de datos que exprese la relación entre el elemento padre y este elemento.

Regla 4: Representar elementos de texto. Si un elemento es de texto puede aparecer en un elemento padre particular una vez:

- Agregar una columna a la tabla padre para vaciar el contenido del elemento.
- Asegurar que el tamaño de la columna creada es lo suficientemente grande para contener el elemento.

- Si el elemento es opcional, hacer que la columna acepte nulos.

Regla 5: Representando elementos de texto múltiple. Si un elemento es de texto, y puede aparecer en un elemento padre más de una vez:

- Crear una tabla para que contenga el valor del texto del elemento y una llave foránea que haga referencia a su elemento padre.
- Y si elemento puede aparecer en más de un elemento padre, crear tablas para expresar la relación entre cada padre y este elemento.

Regla 6: Manipulación de elementos Empty. Para cada elemento vacío encontrado en la DTD:

- Crear una tabla en la base de datos.
- Si el elemento estructural tiene exactamente un padre permitido, agregar una columna a la tabla, esta columna será una llave foránea que referencia al elemento padre.
- Hacer la llave foránea requerida.

Regla 7: Representación de elementos con contenido mixto. Si un elemento tiene un modelo de contenido mixto:

- Crear una tabla llamada *TableLookup* (si todavía no existe) y agregar filas para cada tabla en la base de datos. También se agregar un fila cero que apunte a una tabla llamada *TextContent*.
- Crear una tabla con una llave, una cadena que represente el nombre para el elemento texto y un valor de texto.
- Después crear dos tablas, una para el elemento y otra para una liga a varios contenidos de este elemento, nombrar al elemento seguido por el subelemento respectivamente.
- En la tabla del subelemento, agregar una llave foránea que apunte al elemento principal de la tabla un *TableLookupKey* que apunte a los elementos de la tabla por el contenido del subelemento, una *TableKey* que apunte a una fila especifica en esa tabla y un contador(secuencia) que indique que subelementos o posición de elementos de texto presenta este elemento.

Regla 8: Manipulación elementos de contenido ANY.

- Crear una tabla llamada *TableLookup* (si no existe) y agregar filas para cada tabla en la base de datos.
- Agregar una fila cero que apunte a una tabla llamada *TextContent*.
- Crear una tabla con una llave, una cadena que represente el nombre para el elemento texto y un valor de texto.

- Crear dos tablas, una para el elemento y una para la liga a varios contenidos de ese elemento, nombre estos después del elemento seguido del subelemento, respectivamente.
- En la tabla del subelemento, agregar una llave foránea que haga referencia al elemento principal de la tabla, una *TableLookupKey* que apunte al elemento tabla por el contenido del subelemento, una *TableKey* que apunte a una fila específica en esa tabla, y un contador que indique que subelemento o posición del elemento de texto presenta este elemento.

Regla 9: Atributos CDATA. Para cada atributo con tipo CDATA:

- Agregar una columna a la tabla que corresponde al elemento que lleva ese atributo y dara la tabla el nombre del elemento.
- Asignar a las columnas una variable de longitud cadena, y asignarles tamaño considerable de tal manera que los valores de los atributos no la excedan.

Regla 10: Atributos requeridos/ opcionales/ fijos.

- Si un atributo es especificado como #REQUIRED, entonces será requerido en la base de datos.
- Si un atributo es especificado como #IMPLIED, entonces permitirá nulos en alguna columna que sea creada como un resultado.
- Si un atributo es especificado como #FIXED, entonces debe ser almacenado como se necesite en la base de datos, por ejemplo, como una constante en un cálculo, debe ser tratado como un atributo requerido.

Regla 11: Valores de atributos enumerados. Para atributos con valores enumerados:

- Crear un entero de dos bytes cuyo contenido será el valor enumerado traducido a entero.
- Crear una tabla lookup con el mismo nombre como el atributo y la palabra lookup añadida.
- Insertar una fila en esta tabla correspondiente a cada posible valor para el atributo enumerado.
- Al insertar las filas en la tabla en la cual se encuentran los atributos, traduzca el valor de los atributos al valor entero correspondiente.

Regla 12: Manipulación de atributos ID. Si un atributo con tipo ID tiene significado fuera del contexto del documento XML, almacénelo en la base de datos.

- Si representa el valor de una llave primaria podemos usarlo para insertar o actualizar registros en la base de datos como sea necesario.

- En cualquier otro caso, colgarlo tal que podamos ligar a cualquier IDREF o IDREFS que lo apunte en otra parte del documento.

Regla 13: Manipulación de atributos IDREF.

- Si un atributo IDREF es presentado por un elemento y se sabe que apunta siempre a un tipo específico de elemento, agregar una llave foránea al elemento que referencia la llave primaria del elemento al cual apunta el atributo.

- Si el atributo IDREF puede apuntar a más de un tipo de elemento, agregar una TableLookupKey que indique a que tabla corresponde la llave.

Regla 14: Manipulando atributos IDREFS.

- Si un atributo IDREFS es presentado por un elemento, agregar una tabla join (con el nombre de ambos elementos que contienen el atributo y el elemento inicia apuntando a la concatenación) que contiene una llave foránea que referencia ambos elementos contenidos en el atributo y el elemento inicia apuntándolo.

- Si el atributo IDREFS puede apuntar a elementos de diferente tipo, borrar la llave foránea que referencia al elemento apuntado y agregar una *TablaLookupKey* que indique el tipo de elementos apuntados.

- Agregar una llave foránea a la relación entre esta tabla y una *TableLookup* que contiene nombres de todas la tablas de la base de datos SQL.

Regla 15: Atributos NMTOKEN.

- Para cada atributo con el tipo NMTOKEN, crear una columna en la tabla correspondiente a ese elemento para obtener su valor.

Regla 16: Atributos NMTOKENS.

- Para cada atributo con el tipo NMTOKENS, crear una tabla con una llave primaria autoincrementable, una llave foránea apuntando las filas en la tabla que corresponden al elemento en el cual el atributo es encontrado y una cadena que contiene el valor de cada token encontrado en el atributo.

- Agregar una fila a esta tabla por cada token encontrado en el atributo por el elemento.

Regla 17: Atributos Entidad (Entity) y entidades (Entities).

- Los atributos declarados con el tipo ENTITY o ENTITIES serán tratados como si hubiesen sido declarados con los tipos NMTOKEN o NMTOKENS, respectivamente. (véase reglas 15 y 16).

Regla 18: Chequeo por Colisión de nombres.

- Después de aplicar todo el procedimiento de reglas, checar los resultados de los procesos por colisión de nombres. Si existe una colisión de nombres, cambiar los nombres de las columnas o tablas como sea necesario para resolverla.

Este algoritmo sólo se puede utilizar si el documento XML contiene una DTD y para cada DTD crea una base de datos específica. Este resultado no es el óptimo si lo que se desea es almacenar muchos documentos XML con diferentes DTD's.

3.5. Bases de Datos Comerciales y XML

Muchos de los mayores vendedores de bases de datos están incorporando soporte XML en sus productos o proveen herramientas para el uso de XML en sus bases de datos. Algunos de ellos como Oracle8i de Oracle, SQL Server 2000 de Microsoft, y DB2 de IBM tienen mucho en común en términos de soporte XML. Todos ellos permiten que un documento XML pueda ser almacenado en una única columna en la base de datos con indexamiento y búsqueda limitada del documento[9].

Ellos permiten además el particionamiento de un documento XML en columnas y tablas en la base de datos. Las herramientas y técnicas ofrecidas para lograr la obtención y almacenamiento de datos XML es variable, así como la facilidad con la que los desarrolladores pueden traducir datos XML a una estructura relacional.

3.5.1. Oracle8i

Oracle8i puede almacenar y obtener documentos XML completos como columnas, puede acceder XML almacenado en archivos externos o en la Web y puede mapear elementos de un documento XML a tablas y columnas en la base de datos.

Oracle ofrece también una utilidad SQL XML para Java. Esta herramienta es un conjunto de clases Java que permite la inserción de datos XML en tablas o vistas de objetos. Las clases Java pueden generar también documentos XML de resultados de consultas SQL. Un Servlet Java XSQL está también disponible que permite que la ejecución de consultas SQL retornen los resultados como XML y luego transformen el XML a HTML usando hojas de estilo. Este conjunto de herramientas puede ser utilizado efectivamente de una manera relativamente rápida, cuando son usadas por desarrolladores experimentados con habilidades Java y Oracle intermedias.

Oracle ofrece una rica implementación de soporte XML que provee acceso mediante programación a datos estructurados como XML. Se proveen carac-

terísticas para estructuras centradas en el documento y estructuras centradas en los datos.

3.5.2. DB2 de IBM

El producto de base de datos DB2 de IBM ofrece soporte XML mediante su producto DB2 XML Extender. XML Extender provee nuevas funciones que permiten el almacenamiento y manipulación de documentos XML. Documentos XML enteros pueden ser almacenados en bases de datos DB2 como datos de caracter, como una única columna o como archivos externos, y aún pueden ser administrados por DB2. Se proveen funciones para obtener el documento XML completo o elementos individuales o atributos del documento XML. El producto XML Extender sirve además como repositorio para la administración de DTD. De manera similar a los otros productos de bases de datos líderes, DB2 almacena un documento XML como una única columna o mapea el documento XML a múltiples tablas y columnas.

Una característica única de DB2 es la habilidad de administrar e indexar documentos XML localizados disparmente en el sistema de archivos, una única columna, o distribuidos a través de múltiples tablas y columnas. El producto XML Extender está diseñado para proveer capacidades de búsqueda rápida con páginas (archivos) XML. Esto es útil para aplicaciones que comparten archivos XML o aplicaciones que buscan contra archivos XML como motores de búsqueda. El indexamiento es realizado usando Definición de Acceso a Datos (Data Access Definition, DAD) para definir los elementos y atributos XML que deben ser indexados. Funciones definidas por el usuario (User-defined functions, UDFs) pueden ser usadas para insertar, seleccionar o actualizar un documento completo o elementos y atributos dentro de un documento. DAD es usado también para definir el mapeo de DTD a las tablas y columnas relacionales.

Los procedimientos almacenados son implementados como parte de XML Extender que permiten la obtención o generación de documentos XML sobre la base de consultas SQL. Se mapea un DTD contra las tablas relacionales usando DAD para proveer la estructura del documento generado. Son también soportadas consultas dinámicas, que construyen el documento XML sobre la base de nombres de tablas y nombres de columnas en la consulta en vez del mapeo definido en el DAD.

3.5.3. SQL Server 2000 de Microsoft

SQL Server 2000 ha introducido muchas nuevas características que se enfocan en soporte XML. SQL Server provee soporte para Esquemas de Datos XML, la habilidad de ejecutar consultas XPath, y la habilidad de obtener y escribir datos XML. Los conjuntos de resultados de sentencias SELECT pueden ser retornados como documentos XML mediante el uso de una nueva palabra clave FOR XML. Son soportados múltiples modos o formatos de obtención de datos

que varían la interpretación de la sentencia SELECT y la estructura resultante del documento XML con niveles crecientes de control sobre el documento XML resultante. La jerarquía de árbol XML puede ser determinada automáticamente por el órden de las columnas en la sentencia SELECT y sus nombres de tablas respectios. Alternativamente, el desarrollador puede controlar el contenido y estructura detallada del documento XML directamente en la consulta SELECT. Las columnas pueden ser mapeadas como elementos o atributos o un esquema puede ser especificado para proveer el mapeo.
SQL Server ha agregado también nuevas características para soportar la escritura de documentos XML desde el sistema de archivos a la base de datos usando una nueva función OpenXML. Para usar la función OpenXML, una representación interna del documento XML debe ser creada en memoria usando un nuevo procedimiento almacenado del sistema. La función OpenXML referencia el documento XML en memoria, especifica una consulta XPath para filtrar o localizar los datos XML dentro del documento, y especifica si las columnas relacionales en la base de datos serán mapeadas a elementos de documento XML o atributos. Un esquema puede ser también especificado para determinar el mapeo entre el documento XML y las tablas y columnas de la base de datos. Esto permite básicamente al desarrollador cargar un documento XML en memoria y luego usar SQL y XPath para retornar datos de consulta desde el documento XML en memoria.

Los tres vendedores líderes ofrecen capacidades similares para el manejo de XML. Oracle ofrece una rica interfaz de programación para datos XML usando las clases, servlets y utilidades Java. SQL Server de Microsoft ofrece la mayor flexibilidad para la obtención de estructuración de datos como XML a través de los varios modos descritos arriba. Los tres soportan las estrategias de almacenamiento más comunes aunque las capacidades objeto-relacionales de Oracle pueden proveer mayor flexibilidad en esta área y prevenir parte de la traducción XML adicional que puede ser necesaria en los otros dos productos discutidos. El producto DB2 de IBM parece estar un paso atrás de la competencia con su soporte XML; aunque es posible soportar las tres estrategias comunes de almacenamiento, este soporte de la herramienta para las estrategias no parece ser tan rico[9].

3.6. Conclusión

Las bases de datos relacionales, son por mucho, el paradigma más utilizado para almacenar datos, encontrado a través de todos los dominios de aplicaciones en la industria y la academia. La mayoría de las organizaciones con una infraestructura sustancial habrán implementado con éxito una cierta clase de tecnología de bases de datos relacional, y muchas de estas tendrán equipos de programadores, administradores de la base de datos, y otro personal técnico para diseño, despliegue, soporte, mantenimiento y desarrollo de su base de datos relacional. Incluso mucho personal no técnico involucrado en la disposi-

ción o uso de los sistemas de bases de datos tendrá una apreciación razonable de las capacidades y de la estructura aproximada de la plataforma de la base de datos relacional. El modelo de datos relacional es, en breve, una de las historias prominentes más exitosas de la ciencia de la computación, y una de las ganancias de ingresos más grandes en la industria.[8]

En una aplicación típica, un servidor de la base de datos actúa como depósito de la información para un interfaz cliente/servidor, que corre sobre la máquina de escritorio del usuario. En esta arquitectura cliente/servidor, el cliente envía peticiones de crear, recuperar, modificar, o eliminar los datos almacenados en el servidor. Aplicaciones más sofisticadas pueden implicar aplicaciones de múltiples clientes que interactúan con información particionada en servidores de datos heterogéneos. Además, en una arquitectura multitarea, las capas de aplicación adicionales pueden estar situadas entre la interfaz de usuario y la base(s) de datos.

El uso creciente de Internet como plataforma para aplicaciones ha incrementado la popularidad de los Sistemas Manejadores de Base de Datos Relacional (SMBDR) como almacenamiento de información, con muchos vendedores incorporando soporte Web dentro del conjunto de características de sus gamas de productos.

Los vendedores de SMBDR no han sido lentos en incorporar funcionalidad relacionada con XML en sus productos, y la mayoría ahora ofrece algún grado de soporte de XML. Hay (por ejemplo) extensiones de SQL en muchos productos que permiten que los datos sean extraídos de tablas relacionales y se proyecten en formato XML. Los que permiten el almacenamiento de datos XML pueden ofrecer unas o más opciones en cuanto al nivel de granularidad del almacenamiento.

A pesar de estas innovaciones, hay poco consenso de la industria en cuanto a la manera más apropiada de apoyar XML dentro del modelo de datos relacional. Aplicaciones que atan demasiado cerca las extensiones de la base de datos del producto específico corren el riesgo de encadenarse con un único vendedor y la incompatibilidad con otros productos, y los sistemas que implican más de una plataforma de RDBMS pueden requerir interfaces substancialmente diversas a cada uno. Algunas organizaciones están evaluando las tecnologías nativas de la base de datos de XML que están emergiendo, pero muchas son renuentes en tomar una decisión hasta que estos productos hayan madurado y hayan alcanzado los niveles de escalabilidad, funcionamiento, y rentabilidad de SMBDR.

Capítulo 4

Análisis y Diseño

Debido a las razones dadas en capítulos anteriores para incorporar XML dentro del modelo de datos relacional se propone una arquitectura genérica para el almacenamiento y la recuperación de documentos XML bien formados, y en su caso válidos, dentro de una base de datos relacional. Para ello es necesario tener conocimiento de XML, de Java, de JDBC[1] , de los conceptos de bases de datos relacionales, de SQL y de las extensiones comunes de la base de datos tales como procedimientos almacenados.

Se desarrollará un sistema que almacene dentro de una base relacional documentos XML. Para poder llevar a cabo esta función el sistema deberá verificar que el documento esté bien formado y en caso de contener una DTD o un esquema XML deberá verificar que sea válido. Además el sistema deberá también extraer los documentos XML de la base de datos.

4.1. Administración Del Proyecto

Dentro de la administración del Proyecto se lleva acabo la planeación, el monitoreo y el control del desarrollo del software; para lo cual se genera el Plan de Desarrollo de Software y el Plan de Mediciones.

4.1.1. Plan de Desarrollo de Software

El sistema se desarrollará en base al Proceso Unificado.

El Proceso Unificado plantea 4 fases:

1. Fase de Inicio.
2. Fase de Elaboración.
3. Fase de Construcción.

[1] JDBC: Java Database Connectivity. Es una API de Java que permite que programas en Java ejecuten comandos SQL, permitiendo esto que los programas puedan usar bases de datos en ese esquema

4. Fase de Transición.

Estas fases, tienen los siguientes flujos de trabajo:

- Administración del proyecto.
- Administración de la configuración.
- Definición de requerimientos.
- Análisis.
- Diseño.

4.1.2. Plan de Mediciones

Estas cuatro fases cuentan con los siguientes indicadores de su seguimiento:

1. Fase de Inicio.
   Indicador: Identificar los objetivos del producto a desarrollar.
2. Fase de Elaboración.
   Indicador: Definición de la arquitectura del producto
3. Fase de Construcción.
   Indicador: Producto terminado.
4. Fase de Transición.
   Indicador: Revisión y liberación del producto.

4.2. Administración de la Configuración

Dentro de la administración de la configuración se identifican los productos que se generarán dentro del proceso de desarrollo del software así como también se definen las líneas base.

4.2.1. Elementos de la configuración

Todos los productos que se generarán en el proyecto serán:

- Administración del proyecto.
  - Plan de desarrollo
  - Plan de mediciones
- Definición de requerimientos.
  - Definición del problema
  - Diagrama general de casos de uso

- Prototipo
- Detalle de casos de uso
- Requerimientos no funcionales
- Glosario de términos
- Plan de prueba del sistema

- Administración de configuración
  - Elementos de la configuración a resguardar
  - Creación del depósito del proyecto
- Análisis
  - Diagramas de clases
  - Modelo de la arquitectura del análisis
- Diseño
  - Diseño de la base de datos
  - Diagramas de secuencia
  - Diagrama de estados

4.2.2. Esquema de nombrado

El mecanismo de nombrado, será el siguiente:

Nombre completo junto con la versión, ejemplo:
/plan de desarrollo VER 1.0

4.2.3. Definición de líneas base

- Administración del proyecto
- Definición de requerimientos
- Administración de la configuración
- Análisis
- Diseño

4.2.4. Creación del depósito del proyecto

El depósito electrónico, estará disponible en una sola computadora y su localización será:

`F:\Karen\UNAM\Tesis\Sistema`

El depósito en papel estará disponible en la carpeta de desarrollo.

4.3. Definición de Requerimientos

Para definir los requerimientos del sistema se realizarán los siguientes actividades:

1. Definición del Problema
2. Diagrama general de Casos de Uso
3. Prototipo
4. Detalle de casos de uso
5. Requerimientos no funcionales
6. Glosario de términos
7. Plan de prueba del sistema

4.3.1. Definición del problema

El sistema que se desarrollará, permitirá al usuario almacenar y manipular documentos XML dentro de una base de datos relacional. Antes de almacenar el documento dentro de la base de datos el sistema verifica que esté bien formado y en el caso de tener una DTD o un Esquema XML verifica también que sea válido. Si el documento está bien formado, y en su caso es válido, el sistema almacenará el documento XML dentro de la base de datos. El sistema también extraerá los documentos XML que se encuentran almacenados dentro de la base de datos.

La aplicación debe ser usada fácilmente y debe realizar principalmente dos funciones:

- Almacenar el documento XML dentro de la base de datos.
- Extraer el documento original XML de la base de datos.

Otra característica de la aplicación es que pueda trabajar también con DTDs y Esquemas XML. Como un Esquema XML es un documento XML trabajar con él no involucra mayor esfuerzo, pero en el caso de una DTD es necesario convertirla a un Esquema XML para que poder así trabajar con ella y almacenarla en la base de datos.

Este sistema se implementará con software de distribución gratuita, utilizando como base de datos PostgreSQL y el lenguaje Java para la programación e interfaz.

4.3.2. Lista de características deseadas

El sistema deberá:

- Ser de uso fácil e intuitivo para el usuario.
- Tener la información necesaria.
- Permitir almacenar cualquier documento XML que esté bien formado y en su caso sea válido.
- Permitir al usuario ver la lista de los documentos XML que se encuentran almacenados dentro de la base de datos.
- Extraer los documentos XML que se encuentran almacenados dentro de la base de datos y mostrarlos de la misma forma en que fueron almacenados.

4.3.3. Diagrama General de Casos de Uso

A continuación se muestra el Diagrama General de Casos de Uso donde se encuentran los principales casos de uso del sistema. El diagrama tiene un único actor que es el usuario del sistema, el cual puede ingresar y salir de éste, así como también cargar y extraer un documento XML de la base de datos.

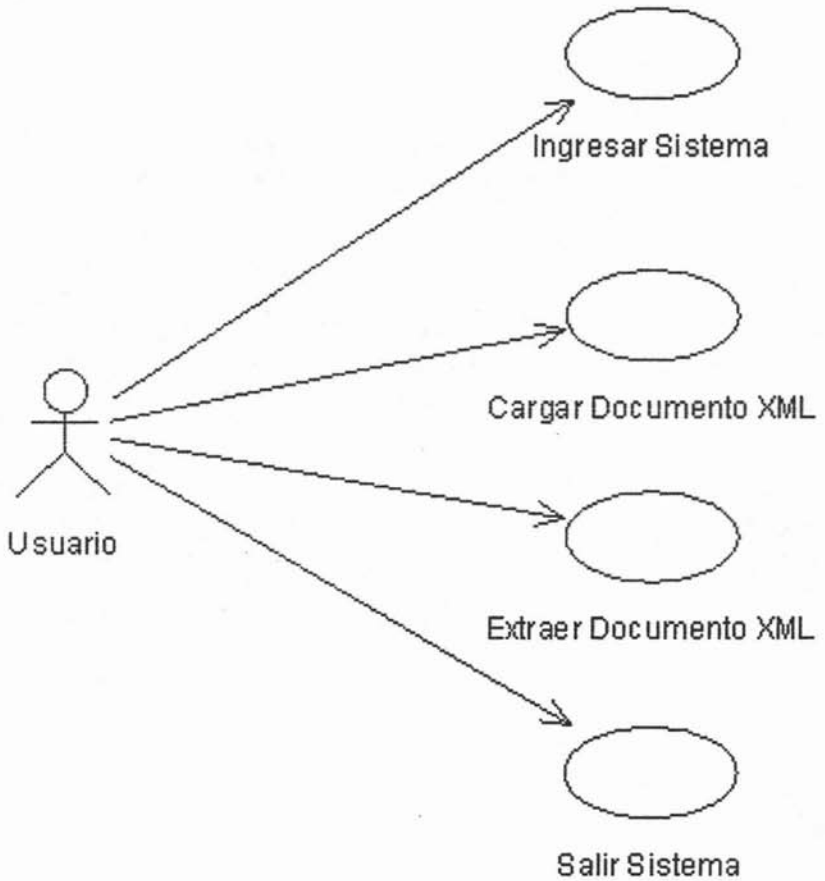


Figura 4.1: Diagrama General de Casos de Uso

4.3.4. Prototipo

A continuación se muestran los prototipos de las pantallas del sistema para dar una idea de como será la interfaz final de acuerdo a los casos de uso anteriormente planteados.

Para poder ingresar al sistema es necesario registrar el nombre y la contraseña, esto es como seguridad para que sólo los usuarios con permisos puedan utilizarlo.

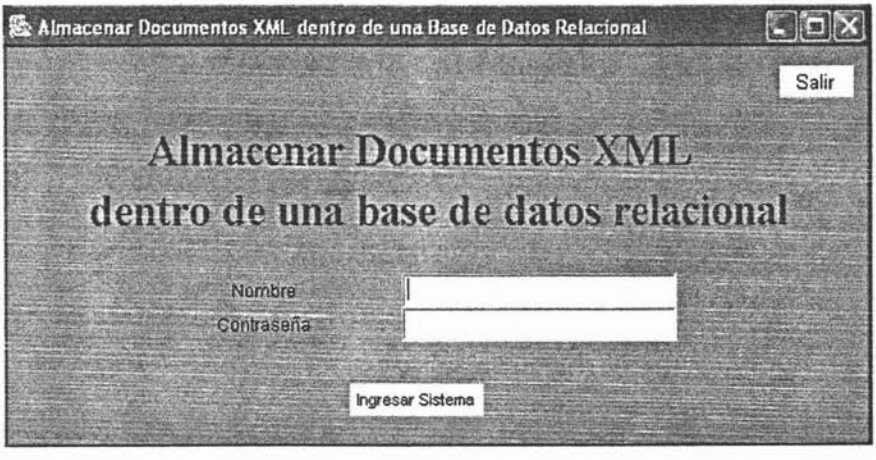


Figura 4.2: Prototipo Ingresar al Sistema

Para poder cargar un documento XML dentro de la base de datos el usuario deberá ingresar la ruta o la dirección donde se encuentra el documento que desea almacenar.

Para poder extraer un documento XML de la base de datos el usuario deberá seleccionar de la lista el documento que desea extraer.

El usuario deberá seleccionar la opción Salir, para finalizar la sesión.

Figura 4.3: Prototipo Cargar Documento XML

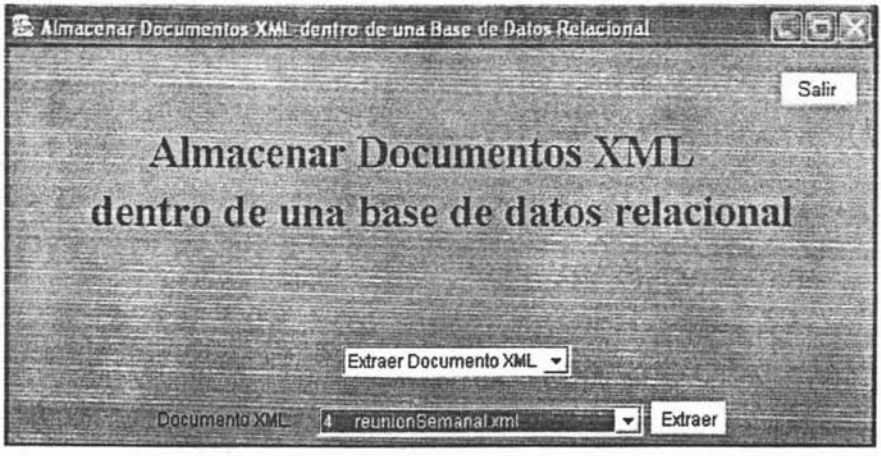


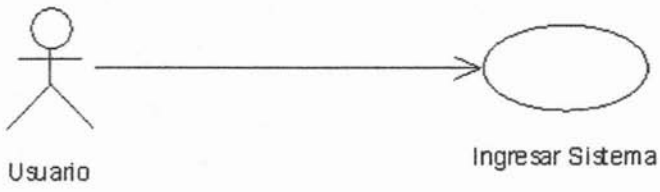
Figura 4.4: Prototipo Extraer Documento XML

Figura 4.5: Prototipo Salir del Sistema

4.3.5. Detalle de Casos de Uso

Caso de uso: Ingresar Sistema

Actor: Usuario



Descripción: El usuario ingresa al sistema una vez que se validaron sus datos de permiso.
Flujo:

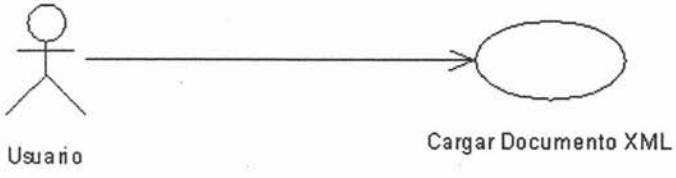
ACTOR		SISTEMA		
Paso	Acción	Paso	Acción	Excepción
1	El usuario ejecuta el programa de java.	2	El programa muestra la ventana de interfaz al usuario.	
3	El usuario introduce sus datos de validación y hace la petición de ingreso al sistema.	4	El sistema valida los datos del usuario.	E_1
		5	El sistema permite al usuario entrar al sistema.	

Excepciones:

EXCEPCION	NOMBRE	ACCIÓN
E_1	Usuario sin permiso de entrada al sistema.	Indicar al usuario que sus datos de autenticación son incorrectos, y que deberá de proporcionar los datos válidos para que pueda entrar al sistema.

Caso de uso: Cargar Documento XML

Actor: Usuario



Descripción: El usuario almacena un documento XML dentro de la Base de Datos.
Flujo:

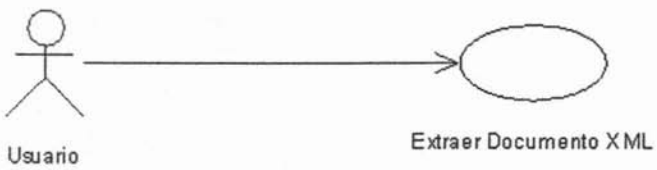
ACTOR		SISTEMA		
Paso	Acción	Paso	Acción	Excepción
1	El usuario selecciona la opción dentro del combo de "Cargar Documento XML".	2	El sistema muestra un campo de texto para colocar el nombre del documento XML que se desea almacenar dentro de la base de datos.	
3	El usuario coloca dentro del campo de texto el nombre del documento XML que desea almacenar dentro de la base de datos.	4	El sistema verifica que el documento exista.	E_1
		5	Si el documento es una DTD el sistema lo transforma a Esquema XML	
		6	El sistema verifica que el documento este bien formado y en su caso válido.	E_2
		7	El sistema almacena el documento XML dentro de la base de datos.	

Excepciones:

EXCEPCION	NOMBRE	ACCIÓN
E_1	Almacenar un documento XML que no existe.	El sistema indica al usuario que el documento no existe.
E_2	Almacenar un documento que no está bien formado o sea válido.	El sistema indica al usuario que el documento XML no está bien formado o no es válido.

Caso de uso: Extraer Documento XML

Actor: Usuario

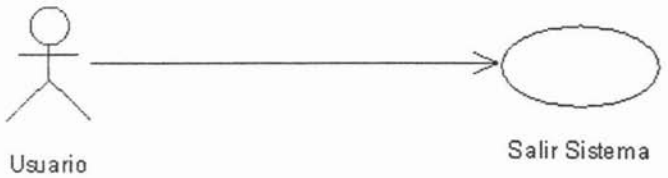


Descripción: El usuario selecciona el documento que desea extraer.
Flujo:

ACTOR		SISTEMA		
Paso	Acción	Paso	Acción	Excepción
1	El usuario selecciona la opción dentro del combo de "Extraer Documento XML".	2	El sistema muestra un combo con la lista de documentos XML que se encuentran almacenados dentro de la base de datos.	
3	El usuario selecciona de la lista el documento XML que desea extraer de la base de datos.	4	El sistema extrae el documento XML de la base de datos y lo muestra al usuario.	

Caso de uso: Salir Sistema

Actor: Usuario



Descripción: El usuario sale del sistema.
Flujo:

ACTOR		SISTEMA		
Paso	Acción	Paso	Acción	Excepción
1	El usuario sale del sistema.	2	El sistema pregunta al usuario si de verdad quiere salir del sistema.	
3	El usuario confirma la salida del sistema.	3	El sistema se cierra.	

4.3.6. Requerimientos no funcionales

- Look and feel
  - Funcionará de manera correcta en monitores con resolución mínima de 640x480 pixeles.
  - Deberá ser aplicable pára configuraciones de monitor de 16 colores como mínimo.
  - El entorno visual será simple.
- Usabilidad
  - El manejo de las ventanas deberá de ser lo más simple y útil.
  - Podrá ser utilizado de manera fácil por cualquier tipo de persona, con facultades mínimas para usar una computadora.
  - Deberá de reducir las posibilidades de equivocación
  - Motivará a que su uso ofrece grandes ventajas
- Desempeño

- Todos los resultados y manejos del sistema deberán ser exactos.

- Operacionales
- Hardware
  - Cualquier computadora que soporte el Software que se necesita.
  - Software
  - Lenguaje de programación Java
  - Manejador de base de datos PostgreSQL.
  - Parser Xerces.
- Herramientas
  - JCreator LE
  - PgAdmin III
- Mantenibilidad y portabilidad
  - El sistema podrá ser ejecutado desde cualquier computadora.
  - El sistema podrá montarse en cualquier tipo de plataforma.
- Seguridad
  - El sistema sólo podrá ser accesado por un usuario con permisos.
- Peticiones culturales y políticas
  - La interfaz de la aplicación será en idioma español.
  - Por convención el nombre de las tablas y los procedimientos almacenados de la base de datos serán en el idioma inglés, los mismo aplicará para el nombre de clases y variables dentro del código java.
- Legales
  - No deberá de incluir información ofensiva y que viole derechos de autor.

4.3.7. Glosario de términos

- **Base de Datos**: conjunto de datos almacenados y organizados bajo estructuras de control bien definidas.
- **Sistema**: conjunto de elementos que interrelacionan entre sí para lograr un objetivo en común.
- **Usuario**: nombre que recibe las personas que pueden utilizar el sistema.
- **Documento XML**:Es un documento que sigue las reglas de XML.

4.3.8. Plan de prueba del sistema

Objetivo

Asegurar que el sistema opere de manera eficaz y confiable, es decir, que cumpla con todos los requerimientos funcionales y no funcionales.

Estrategias de prueba

Se probarán todos los casos de uso reflejados en la interfaz de usuario (prototipo) y el orden será el siguiente: ingresar al sistema, cargar documento XML, extraer documento XML y salir del sistema.

Para el mecanismo de pruebas participará un agente externo y un evaluador.

Casos de prueba

Prueba 1: Ingresar al Sistema

Fundamento: Es necesario restringir la entrada al sistema por cuestiones de seguridad, sólo deberá dejar acceder a usuarios con permiso.

Precondición: El usuario deberá de estar registrado

Entradas	Resultados esperados	Caso de uso
Ejecutar el programa.	Se carga la ventana de interfaz para el usuario.	Ingresar sistema
Introducir el nombre y la contraseña y presionar el botón de "Ingresar al Sistema".	El sistema muestra las opciones que puede realizar el usuario.	

Prueba 2: Cargar Documento XML bien formado y válido

Fundamento: Para poder manipular y tener acceso a información de documentos XML es necesario almacenar el documento dentro de la base de datos y para ello el documento debe estar bien formado y en su caso ser válido.

Precondición: Haber ingresado al sistema

Prueba 3: Cargar Documento XML no bien formado

Fundamento: El sistema debe validar que el documento XML que se desea almacenar dentro de la base de datos este bien formado.

Entradas	Resultados esperados	Caso de uso
Seleccionar la opción "Cargar Documento XML".	Aparece un campo de texto para colocar el nombre del documento XML que se desea almacenar dentro de la base de datos.	Cargar Documento XML
Colocar el nombre del documento XML dentro del campo de texto y oprimir el botón "Cargar".	El sistema indica al usuario que el documento a sido almacenado exitosamente y le muestra el ID que identifica al documento dentro de la base de datos.	

Precondición: Haber ingresado al sistema y que el documento XML no esté bien formado

Entradas	Resultados esperados	Caso de uso
Seleccionar la opción "Cargar Documento XML".	Aparece un campo de texto para colocar el nombre del documento XML que se desea almacenar dentro de la base de datos.	Cargar Documento XML no válido
Colocar el nombre del documento XML dentro del campo de texto y oprimir el botón "Cargar".	El sistema indica al usuario que el documento no se puede almacenar dentro de la base de datos porque de acuerdo a la DTD o al esquema XML no es un documento válido.	

Prueba 4: Cargar Documento XML no válido

Fundamento: En caso de que el documento XML que se quiere almacenar dentro de la base de datos contenga una DTD o un esquema XML, el sistema debe validar que sea válido.

Precondición: Haber ingresado al sistema y que el documento XML no sea válido

Prueba 5: Extraer Documento XML

Fundamento: Para verificar que el documento XML haya sido almacenado satisfactoriamente dentro de la base de datos.
Para observar la información del documento XML.

Precondición: Haber ingresado al sistema
El documento XML que se desea extraer haya sido almacenado dentro de la base de datos anteriormente.

Entradas	Resultados esperados	Caso de uso
Seleccionar la opción "Cargar Documento XML".	Aparece un campo de texto para colocar el nombre del documento XML que se desea almacenar dentro de la base de datos.	Cargar Documento XML no bien formado
Colocar el nombre del documento XML dentro del campo de texto y oprimir el botón "Cargar".	El sistema indica al usuario que el documento no se puede almacenar dentro de la base de datos porque no esta bien formado.	

Entradas	Resultados esperados	Caso de uso
Seleccionar la opción de "Extraer Documento XML".	Aparece un combo con la lista de documentos XML almacenados dentro de la base de datos.	Extraer Documento XML
Seleccionar el documento XML que se desea extraer y oprimir el botón "Extraer".	El sistema muestra al usuario el documento XML seleccionado.	

Prueba 6: Salir Sistema

Fundamento: El usuario tendrá que abandonar en algún momento el sistema, y el sistema por su parte requerirá cerrarse.

Precondición: Haber ingresado al sistema

Entradas	Resultados esperados	Caso de uso
Seleccionar la opción "Salir"	El sistema pregunta si en verdad se desea salir del sistema.	Salir del sistema
Seleccionar la opción "Aceptar"	El sistema se cerrara.	

4.4. Análisis

En esta etapa del desarrollo del software se construye el modelo del análisis que sirve de base para estructurar todo el sistema.

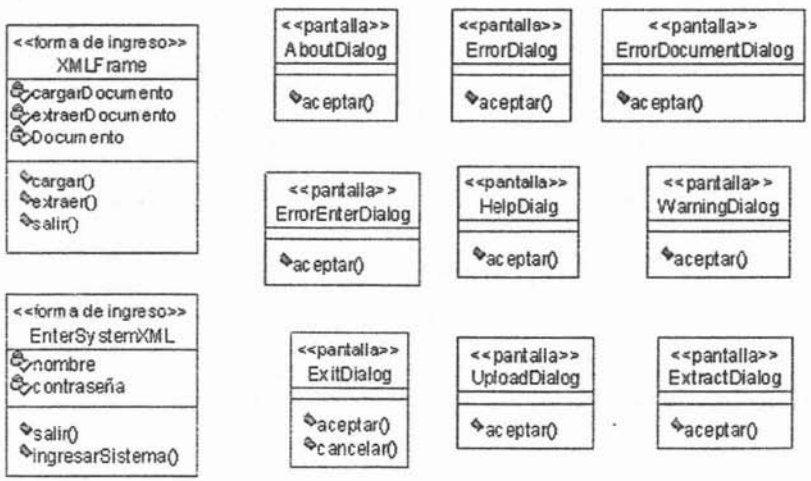
Para el análisis se realizan las siguientes actividades:

- Definir la arquitectura del análisis.

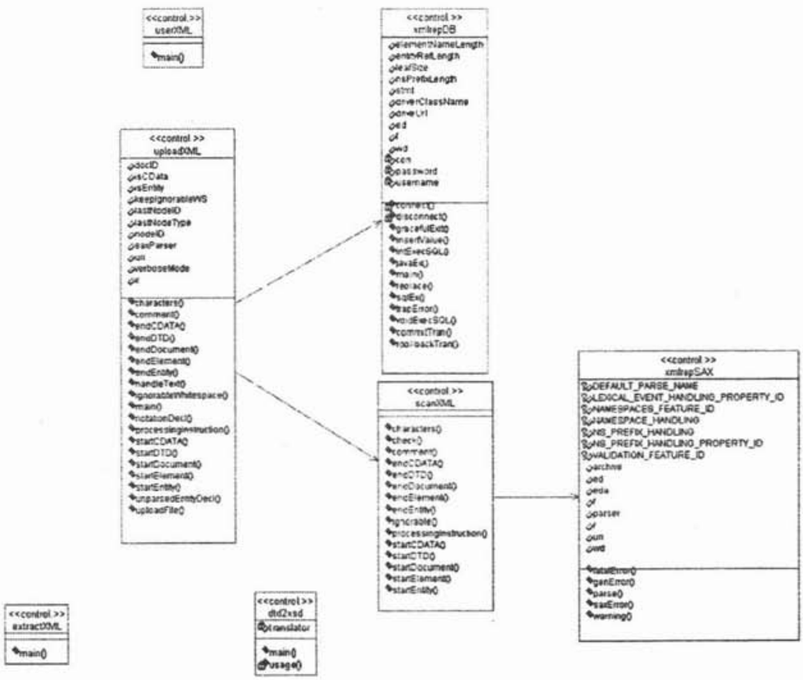
- Analizar los casos de uso para encontrar las clases necesarias para cada uno.
- Definir las clases y sus responsabilidades.

4.4.1. Diagramas de clases

Clases de Interfaz

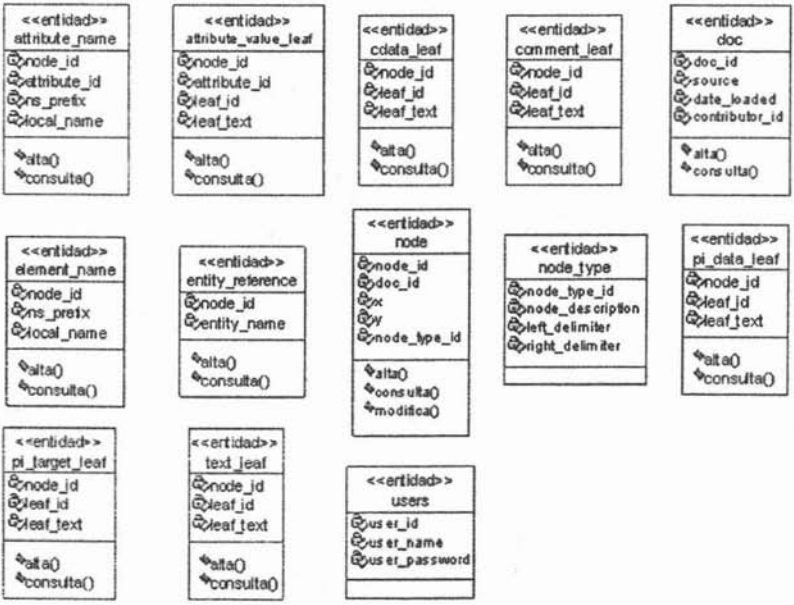


Clases de Control



<<control>>
userXML
main()
<<control>>
xmlrepDB
elementNameLength
nsPrefixLength
driverClassName
con
password
username
connect()
disconnect()
gracefulExit()
insertValue()
main()
replace()
sqlEx()
voidExecSQL()
commitTran()
rollbackTran()
<<control>>
uploadXML
docID
isCData
isEntity
keepIgnorableWS
lastNodeID
lastNodeType
nodeID
saxParser
verboseMode
characters()
comment()
endCDATA()
endDTD()
endDocument()
endElement()
endEntity()
handleText()
ignorableWhitespace()
main()
notationDecl()
processingInstruction()
startCDATA()
startDTD()
startDocument()
startElement()
startEntity()
unparsedEntityDecl()
uploadFile()
<<control>>
scanXML
characters()
comment()
endCDATA()
endDTD()
endDocument()
endElement()
endEntity()
processingInstruction()
startCDATA()
startDTD()
startDocument()
startElement()
startEntity()
<<control>>
xmlrepSAX
DEFAULT_PARSE_NAME
LEXICAL_EVENT_HANDLING_PROPERTY_ID
NAMESPACES_FEATURE_ID
NAMESPACE_HANDLING
NS_PREFIX_HANDLING
NS_PREFIX_HANDLING_PROPERTY_ID
VALIDATION_FEATURE_ID
archive
parser
fatalError()
genError()
parse()
saxError()
warning()
<<control>>
extractXML
main()
<<control>>
dtd2xsd
translator
main()
usage()

Clases de Entidad



4.5. Diseño

El objetivo del Diseño de un software es crear un punto de partida para las actividades de implementación, definir la descomposición del sistema en piezas manejables o subsistemas y definir con mayor detalle los modelos del análisis incluyendo conceptos del ambiente de implementación.

Dentro de la etapa del Diseño se realizan las siguientes actividades:

- Definir la arquitectura del sistema.
- Construir el diagrama de estados.
- Construir los diagramas de secuencias

4.5.1. Arquitectura del Sistema

La arquitectura planteada consiste en una base de datos relacional, que contiene las tablas y los procedimientos almacenados necesarios para tener acceso a ellas; y las clases de Java que proporcionan una capa de interfaz sobre la cual la aplicación será construida.

Los procedimientos almacenados forman una API rudimentaria para las clases de Java; las clases no necesitan saber sobre el diseño de la tabla, sólo requieren conocer cómo son los procedimientos almacenados que insertan, seleccionan, actualizan, y eliminan información de las tablas.

La arquitectura consta de tres niveles:

- objetos de la base de datos (las tablas)
- acceso a los datos (los procedimientos almacenados)
- interfaz (clases Java)

Esta arquitectura proporciona una plataforma básica sobre la cual se pueden construir aplicaciones que mantienen y manipulan documentos XML dentro de una base de datos relacional.

El modelo de datos que se usará para las tablas de la base de datos está basado en el Modelo de Objetos de Documento (DOM) de W3C's.

4.5.2. Almacenamiento de DOM en una Base de Datos Relacional

Los expertos de SQL han abordado el tema de almacenar árboles en bases de datos relacionales y puesto que DOM es un árbol de nodos, se puede utilizar alguna de sus propuestas. Una de las propuestas más común es el *Modelo de*

Conjuntos Anidados.[25]

El Modelo de Conjuntos Anidados, es adecuado para XML. En el Modelo de Conjuntos Anidados, cada nodo se almacena con dos coordenadas de números enteros ((x, y) representando izquierda y derecha respectivamente), que permiten preservar el orden y el sentido de la localización en la jerarquía[12].

El Modelo de Conjuntos Anidados se puede representar por medio de:

- Árboles
- Contenedores anidados

Árboles

Si se trabaja con un árbol se utiliza el algoritmo *Árbol Transversal Preorden Modificado*[25], donde las coordenadas son asignadas a la hora de crear o almacenar los nodos, recorriendo el árbol de nodos de arriba a abajo y de izquierda a derecha, numerando durante el camino como se muestra en la Figura 4.6.

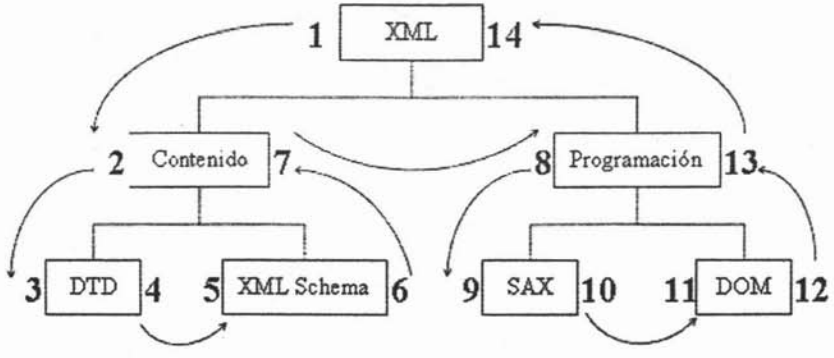


Figura 4.6: Modelo de Conjuntos Anidados con Árboles

Como se puede ver en la Figura 4.6, los números indican la relación entre cada nodo. Por ejemplo, el nodo *'DTD'* que tiene los valores 3,4, es un descendiente del nodo *'XML'* con valores 1,14. De la misma manera, se puede decir que todos los nodos con su valor izquierdo mayor de 2 y suvalor derecho menor de 7, son descendientes del nodo *'Contenido'* con valores 2,7.

Brevemente el algoritmo funciona con los siguientes pasos:

1. Comienza el recorrido y la numeración en el nodo raíz y se coloca un 1 en su coordenada izquierda.

2. Se baja de nivel al primer nodo hijo y se coloca el siguiente número en la coordenada izquierda.

3. Se repite el paso 2 hasta llegar a un nodo hoja(que ya no tiene nodos hijos).

4. Coloca el siguiente número en la coordenada derecha.

   - Si existe un nodo del mismo nivel sigue el paso 5.
   - Si no existe un nodo del mismo nivel sigue el paso 6.
   - Si el nodo es el nodo raíz se ha terminao el recorrido.

5. Se mueve al siguiente nodo del mismo nivel colocando el siguiente número en la coordenada izquierda de este nodo y se vuelve al paso 2.

6. Sube al nodo padre y sigue el paso 4.

Este sistema[8] de dos coordenadas provee métodos para la navegación y creación de la estructura, por ejemplo si se tiene el nodo (x, y):

- El siguiente hermano (y+1, y').
- El primer hijo (x+1, y').
- Los nodos descendentes (x', y') son todos los nodos con valores x'mayor de x y valores y'menor de y.
- Los nodos ancestros (x', y') son todos los nodos con valores x'menor de x y valores y'mayor de y.

Contenedores Anidados

En el Modelo de Conjuntos Anidados, se puede representar la jerarquía de una nueva forma, no como nodos y líneas, sino como contenedores anidados[13] como se muestra en la Figura 4.7.

Para determinar los valores izquierdos (x) y derechos (y) se comienza a enumerar en el extremo izquierdo del contenedor externo y se continua a la derecha como se ve el la Figura 4.8.

De ambas representaciones se puede obtener la Tabla 4.1 usando los valores izquierdo (x) y derecho (y) para representar el anidamiento de los nodos:

Si se aplica el Modelo de Conjuntos Anidados al árbol DOM de la Figura 2.2 las coordenadas obtenidas se muestran en la Figura4.9:

Este esquema de etiquetado claramente facilita la serialización rápida de documentos (o de fragmentos de estos).

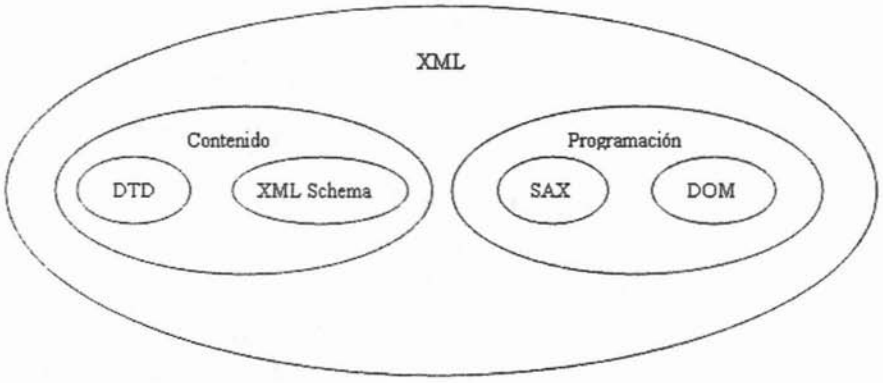


Figura 4.7: Contenedores Anidados

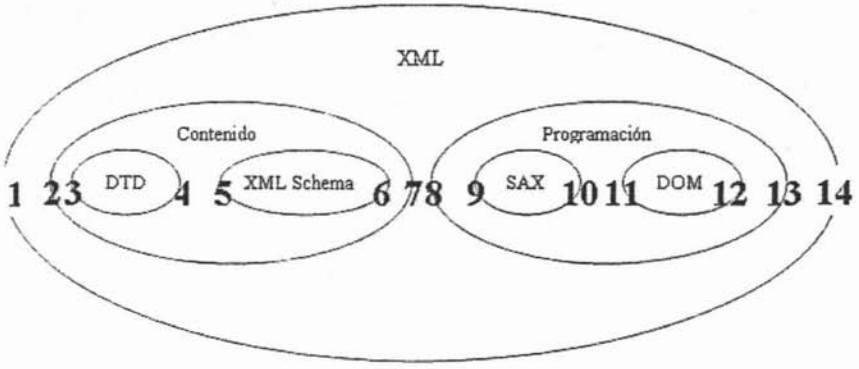


Figura 4.8: Modelo de Conjuntos Anidados con Contenedores Anidados

Padre	Nombre	Izquierdo	Derecho
	XML	1	14
XML	Contenido	2	7
Contenido	DTD	3	4
Contenido	XML Schema	5	6
XML	Programación	8	13
Programación	SAX	9	10
Programación	DOM	11	12

Cuadro 4.1: Valores izquierdo y derecho

4.5.3. Diagrama de estados para la navegación de clases de interfaz

Los diagramas de estados muestran las secuencias de eventos en el sistema; estos se pueden contruir de acuerdo a objetos como: ventanas, transacciones,

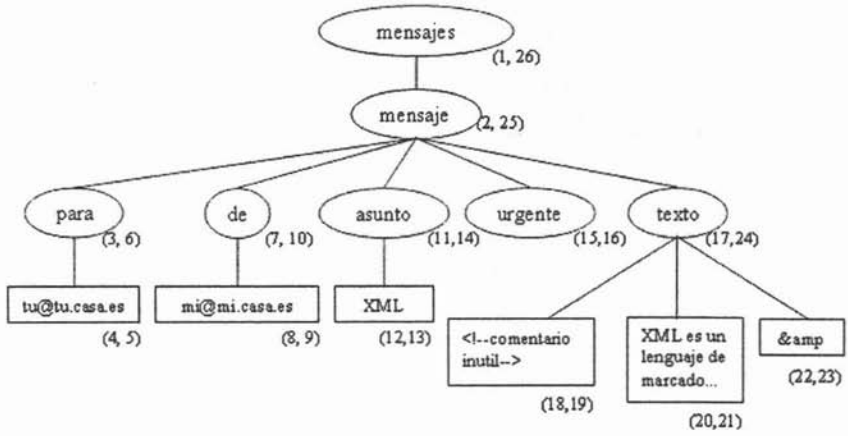
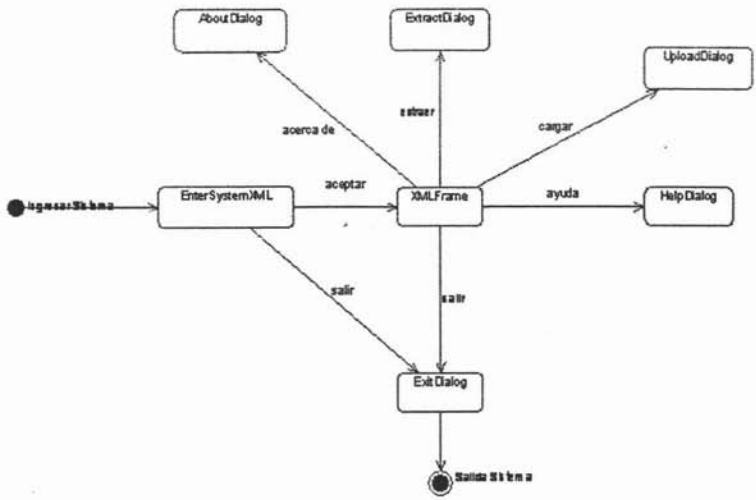


Figura 4.9: Árbol DOM con el Modelo de Conjuntos Anidados

controladores, dispositivos. En este caso se muestra el diagrama de estados para la navegación de clases de la intefaz.

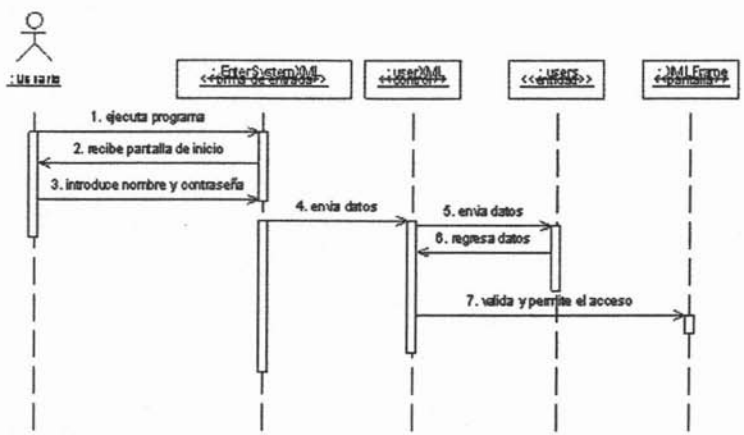


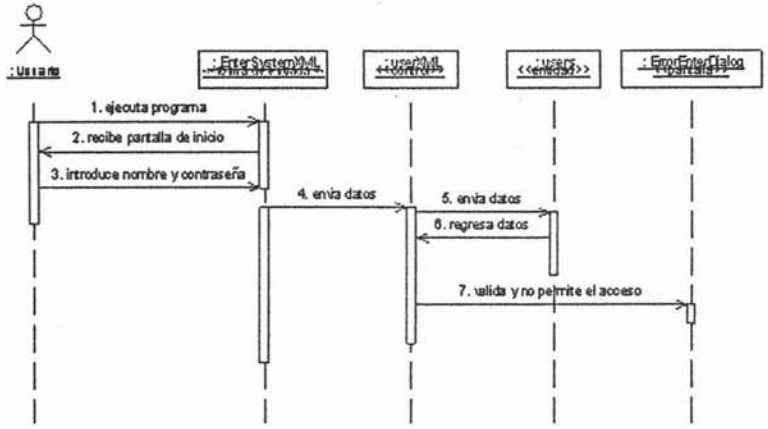
4.5.4. Diagramas de Secuencia

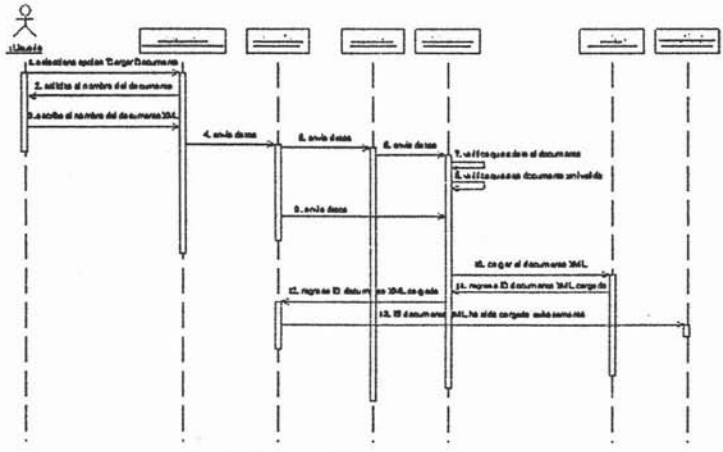
Los diagramas de secuencia se utilizan para modelar los aspectos dinámicos de un sistema. Modelan instancias de clases, interfaces, componentes, con los mensajes enviados entre ellos, en el contexto de un escenario o caso de uso que ilustra un comportamiento.

A continuación se muestran los diagramas de secuencias más significativos del sistema:

Ingresar al sistema válido







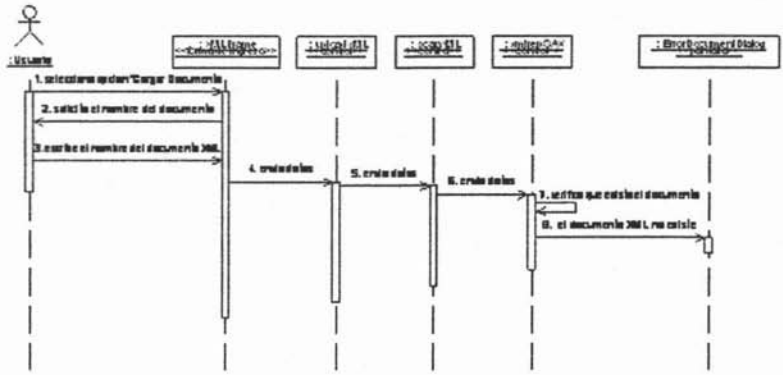
Ingresar al sistema NO válido

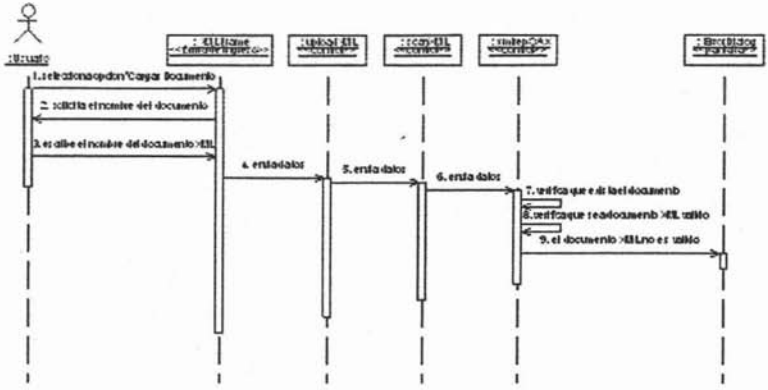
Cargar Documento XML válido

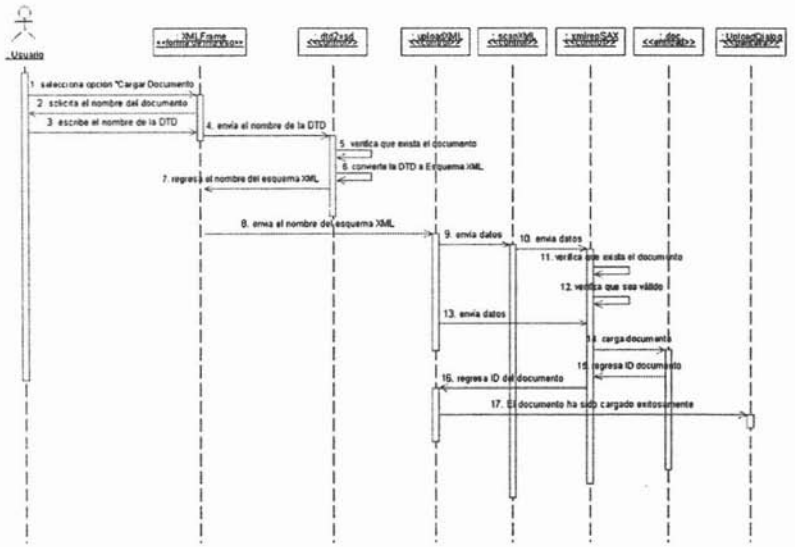
Cargar Documento XML NO válido (no existe documento)

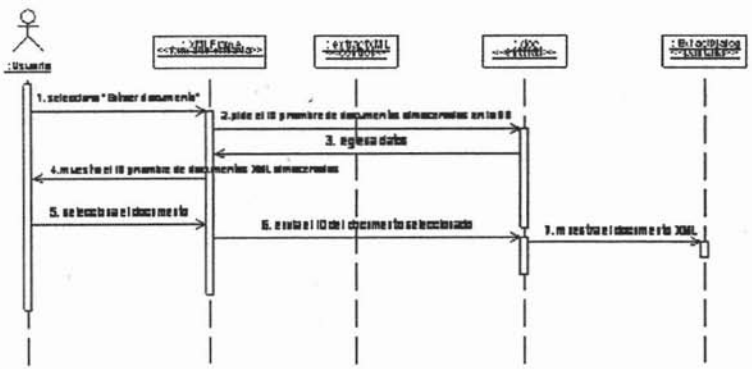
Cargar Documento XML NO válido (documento no válido)

Cargar DTD






:Usuario
:XMLFrame
dtd2xsd
uploadXML
scanXML
xmlreaSAX
doc
UnicaClase
1. selecciona opcion "Cargar Documento
2. solicita el nombre del documento
3. escribe el nombre de la DTD
4. envia el nombre de la DTD
5. verifica que exista el documento
6. convierte la DTD a Esquema XML
7. regresa el nombre del esquema XML
8. envia el nombre del esquema XML
9. envia datos
10. envia datos
11. verifica que exista el documento
12. verifica que sea válido
13. envia datos
14. carga documento
15. regresa ID documento
16. regresa ID del documento
17. El documento ha sido cargado exitosamente


:Usuario
1. selecciona "Obtener documento"
3. regresa datos
5. selecciona el documento
6. envia el ID del documento seleccionado
7. muestra el documento XML

4.5.5. Base de Datos

Para almacenar datos XML de esta manera, se necesitan tablas que reflejen la estructura abstracta de DOM, con la información agregada requerida por el Modelo de Conjuntos Anidados.

Se necesitan tablas en las cuales se pueda almacenar la información sobre el nodo, tales como su tipo, coordenadas (x, y), atributos, y su nombre (que puede incluir un espacio de nombres) o su contenido (en el caso de un nodo texto o de un nodo CDATA). Se necesita estar pendiente del contenido posible de cada tipo de nodo; un nodo elemento tendrá un nombre y puede tener un espacio de nombre y/o una lista de atributos (tratadas como pares de nombre/valor), mientras una hoja de texto sólo tiene contenido.

Puesto que se desea construir un repositorio para más de un documento, se necesita una tabla para almacenar cierta información sobre los documentos (por lo menos, un identificador de documento ID único), y las filas en la tabla nodo necesitarán llevar a cabo una referencia al documento al cual pertenecen (se puede construir un repositorio multidocumentos como si fuera un documento grande de XML, abarcando muchos documentos más pequeños, pero para muchas aplicaciones los gastos indirectos de mantener las coordenadas (x, y) probablemente resultaría demasiado costoso).

El Modelo Físico de Datos

El modelo físico de los datos se muestra en la Figura 4.10. Hay 12 tablas en total; la tabla (*node-type*) es una tabla de catálogo que define los diversos tipos de nodos que se van almacenar. Las 11 tablas restantes almacenarán la información con respecto a los nodos, los atributos, y hojas para cada documento de XML que se almacenará. Los valores de las coordenadas (x, y) (respectivamente, izquierda y derecha de las coordenadas Conjuntos Anidados) para cada nodo serán almacenadas en la tabla node.

La tabla *node_type* contiene los tipos de nodos que puedan aparecer, cada uno de estos tiene un identificador ID (*node_type_id*) y una descripción (*node_description*); las dos columnas restantes (*left_delimiter* y *right_delimiter*) almacenan los caracteres que vienen antes y después (repectivamente) de un tipo de nodo específico; por ejemplo, una sección de CDATA comienza con el *<![CDATA[* y termina con *]]>* (ver Tabla 4.2).

La tabla *doc* almacena el identificador del documento *doc_id*, el nombre del documento fuente (*source*), la fecha en que fue cargado (*date_loaded*), y una referencia al usuario que la cargó (*contributor_id*).

La tabla *node* será el centro de la mayoría de la actividad de la base de datos; cada nodo de cada documento almacenado en el repositorio tendrá que

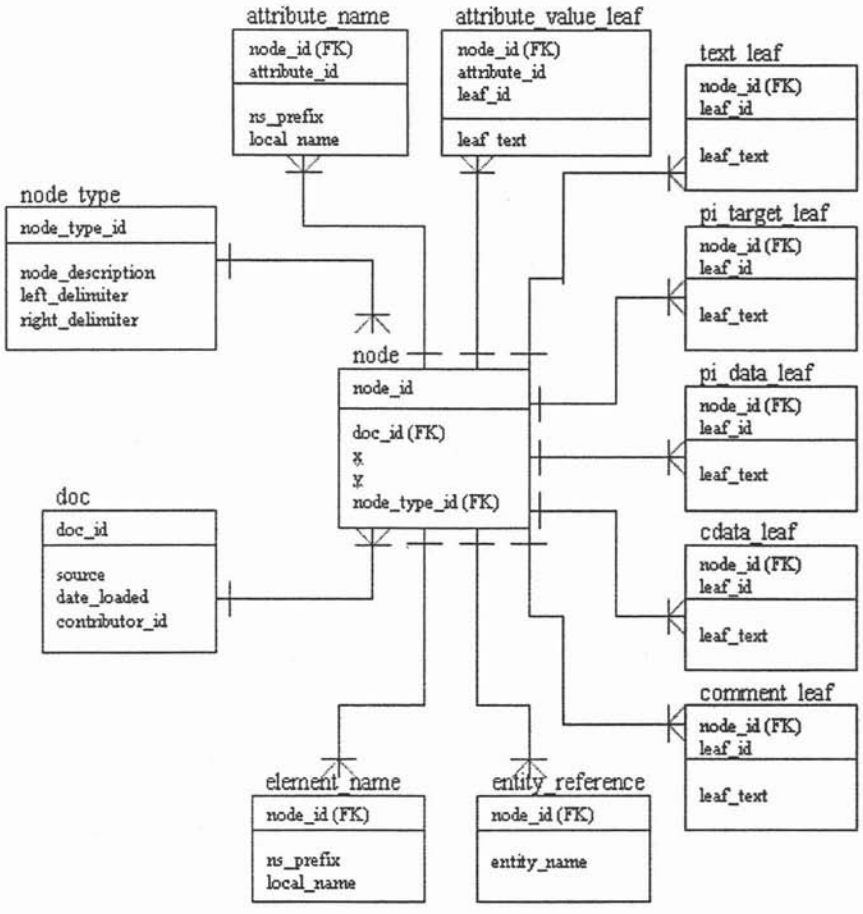


Figura 4.10: Diagrama de la Base de Datos

node_type_id	node_description	left_delimiter	rigth_delimiter
1	ELEMENT	<	>
3	TEXT_NODE		
4	CDATA_SECTION_NODE	<![CDATA[	]]>
5	ENTITY_REFERENCE_NODE	&	;
7	PROCESSING_INSTRUCTION_NODE	<?	?>
8	COMMENT_NODE	<!–	–>
9	DOCUMENT_NODE		

Cuadro 4.2: Tabla node_type

entrar aquí. A cada nodo se le asignará un *node_id arbitrario* (por razones del funcionamiento, esta identidad única será señalada como la llave primaria para

la tabla) y las coordenadas (x, y). Cada nodo tendrá una referencia (*doc_id*) al documento que lo contiene y a un tipo (*node_type_id*).

Si un nodo dado es un elemento, habrá una entrada en la tabla *element_name*, que tiene otra vez *node_id* como su llave primaria. Cada elemento puede tener un espacio de nombre (*ns_prefix*) y tendrá un nombre (*local_name*). En común con muchas de las otras tablas, la columna *node_id* es una llave foránea que hace referencia al *node_id* de la tabla *node*.

La especificación de XML permite nombres y espacios de nombre muy largos de elemento, lo cual crea un problema para el almacenamiento en algunos SMBDR (en algunos sistemas, los campos de carácter se restringen a 255 caracteres). En este caso también se hará una restricción en la longitud de estos campos. Las restricciones quedan de la siguiente forma sin ninguna razón en especial.

Tamaño Hoja = 255
Longitud del nombre del elemento = 50
Longitud del espacio de nombre = 25
Longitud de las referencias de entidad = 50

Un elemento puede tener uno o más atributos. En DOM, un atributo se trata como tipo especial de nodo. Sin embargo, puesto que los atributos se sujetan al elemento que lo contiene y tienen diferentes propiedades a otros nodos, no se almacenará ninguna información de atributos en la tabla *node*. Los nombres de los atributos se almacenan en la tabla *attribute_name*; cada entrada es identificada únicamente por el *node_id* del elemento que lo contiene y un arbitrario *attribute_id*. El espacio de nombre (si existe) se almacena en *ns_prefix*, y el nombre se almacena en *local_name*.

La tabla *attribute_value_leaf* sigue el mismo patrón, puesto que los valores de los atributos son a menudo largos, se necesita tener en cuenta cualquier longitud. Esto se puede hacer en algunos SMBDR con una columna varchar muy larga (o iiimitada), pero como se desea una solución tan portable como sea posible, se puede realiza el mismo efecto creando una lista de los campos varchar de longitud finita.

Si la longitud del valor de los atributos excede la longitud de la columna *leaf_text*, una nueva fila se agrega para el sobrante y se da un secuencial *leaf_id*, y así, hasta que el valor se ha acomodado. Para reconstruir el valor, una consulta tendrá que seleccionar todas las filas de *attribute_value_leaf* para un *node_id* y un *attribute_id* dados, pedirlos por el *leaf_id*, y pasar el resultado fijado a una función que una las hojas traseras.

El contenido de los nodos texto se almacena en *text_leaf*, otra vez con un *leaf_id* para determinar el orden secuencial de los pedazos en los cuales el contenido estuvo partido en caso que fuera demasiado grande para la columna del

leaf_text.

Las instrucciones de procesamiento consisten de una referencia a la aplicación que esta dirigida seguida por los datos, separados por espacio en blanco. Puesto que la aplicación y los datos pueden ser largos, se almacenarán con el mismo mecanismo de particionamiento, en *pi_target_leaf* y *pi_data_leaf*, respectivamente.

Las entidades son generalmente cortas, así que una tabla simple *entity_reference* con sólo una columna *node_id* y una *leaf_text* será suficiente.

Las tablas restantes, *comment_leaf* (para los comentarios de XML) y *cdata_leaf* (para las secciones de CDATA), necesitan almacenar cantidades de información grandes, y por lo tanto se utilizará el mismo mecanismo que particiona.

Capítulo 5

Implementación

5.1. Base de Datos

Existen varias alternativas a la hora de decidir el sistema de bases de datos sobre el que se desea trabajar, se debe saber que cada uno de ellos tiene sus características que lo convierten en el más adecuado según para qué tipo de proyectos o en función de los intereses del programador.

En este sistema como se mencionó anteriormente se trabajará bajo la base de datos PostgreSQL por ser el servidor de bases de datos de código abierto más potente que existe y por contar con características avanzadas como los procedimientos almacenados que serán necesarios para implementar este sistema.

5.1.1. PostgreSQL

En la tabla 5.1se presenta una comparativa, con los criterios principales a tener en cuenta a la hora de decidir qué base de datos utilizar[14].

Criterios	Acces	SQL Server	MySQL	PostgreSQL
Plataforma	Windows	Windows	Windows/Linux	Windows/Linux
Velocidad	Negativa	Positiva	Positiva	Negativa
Volumen Datos	Negativa	Positiva	Positiva	Positiva
Integridad	Negativa	Positiva	Negativa	Positiva
Potencia	Negativa	Positiva	Positiva	Positiva
Coste/MB	Positiva	Negativa	Positiva	Positiva

Cuadro 5.1: Comparación de SMBD

La Base de Datos PostgreSQL está disponible para el sistema operativo Windows o Linux.

PostgreSQL es el servidor de bases de datos de código abierto más potente que existe y por lo tanto es una mejor alternativa que MySQL cuando se necesitan características avanzadas como transacciones, procedimientos almacenados,

triggers, vistas, etc.

PostgreSQL es el servidor de bases de datos más utilizado por los programadores de servlets de Java y, en general, por todos aquellos que realizan aplicaciones cliente servidor complejas o críticas en el mundo Linux/Unix.

Para aplicaciones Windows, PostgreSQL es una alternativa económica a SQL Server, pues su coste por MB es menor y tiene similares prestaciones. Esta diferencia económica es especialmente sustancial si se necesita un Servidor Dedicado de bases de datos.

La mayor limitación de PostgreSQL es su velocidad; pues el sistema de bases de datos más lento que se muestra en la tabla.

PostgreSQL es sin duda el Sistema de Gestión de Bases de Datos de código abierto (gratuito y con código fuente disponible) más avanzado del mundo. Posee las características de los más potentes sistemas comerciales como Oracle o SQL Server.

Entre ellas se pueden destacar:

- Completo soporte para transacciones. Una transacción está formada por un conjunto de acciones de forma que o se ejecutan todas ellas o bien ninguna. Utilizando transacciones se asegura la consistencia de los datos.
- Soporte completo ACID (Atomicity Consistency Isolation Durability): Es posible definir operaciones atómicas, es decir, formadas por comandos que se ejecutan todos o ninguno.
- Consistencia, que garantiza que la base de datos nunca se queda en un estado intermedio de una transacción (con parte de los comandos ejecutados y parte no).
- Aislamiento, que mantiene separadas las transacciones de usuarios distintos hasta que éstas han terminado.
- Durabilidad, garantiza que el servidor de datos guarda las actualizaciones pendientes de forma tal que pueda recuperarse de una terminación brusca tal como desconectar la máquina. Esta característica se implementa mediante el log de transacciones, que almacena todas las transacciones que aún no han sido lanzadas (commit).
- Procedimientos almacenados. Código ejecutable que se almacena compilado en el servidor. Entre otras cosas, permite optimizar las aplicaciones evitando transferencias innecesarias a través de la red (aplicaciones con parte cliente y parte servidor). Los procedimientos almacenados se pueden escribir en su propio lenguaje de programación (PL/pgSQL) o bien en Perl o TCL.

- Soporte para construcciones SQL del tipo subselect.
- Triggers. Procedimientos almacenados que se lanzan automáticamente bajo determinadas circunstancias como actualizaciones de campos. Permite establecer reglas de integridad y consistencia a nivel del servidor de base de datos.
- Vistas. Conjunto de registros resultado de una consulta que se comportan como una tabla física para facilitar su manejo.
- Orientación a objetos con herencia de tablas.

Gestión de bases de datos PostgreSQL

Las bases de datos de PostgreSQL no son archivos que se puedan subir a su sitio web como los de Access, sino que residen en un servidor de datos separado. Por ello se debe utilizar algún programa cliente que le permita conectarse al servidor de datos con el fin de crear las tablas, subir datos, editar registros, etc. Aunque PostgreSQL está en un servidor Linux, se pueden gestionar las bases de datos desde computadoras con cualquier sistema operativo utilizando las aplicaciones adecuadas.

Existen varios métodos para gestionar las bases de datos PostgreSQL:

PhpPgAdmin

PhpPgAdmin es una aplicación realizada en PHP que permite administrar las bases de datos PostgreSQL a través de Internet utilizando páginas web.

PhpPgAdmin tiene entre otras las siguientes funcionalidades:

- Permite crear tablas.
- En cada tabla puede crear y modificar campos, especificando su tipo de datos, valores por defecto, etc.
- Se pueden lanzar sentencias SQL contra la base de datos.
- Se pueden obtener volcados de la base de datos tanto de estructura como de datos.
- Se pueden importar datos al servidor: enviar archivos con los datos de una tabla.

ODBC para PostgreSQL

Si la computadora de trabajo utiliza un sistema operativo Windows, se podrán gestionar las tablas de las bases de datos PostgreSQL utilizando cualquier programa Windows que utilice ODBC, por ejemplo Access.

Para ello se debe empezar por descargar el controlador ODBC para PostgreSQL disponible en el sitio web de los creadores de la base de datos: www.-postgresql.org (proyecto psqlodbc - PostgreSQL ODBC driver). Después de instalarlo en la computadora de trabajo, se podrá crear un DSN que apunte a la base de datos remota en los servidores y utilizar Access para acceder a las tablas.

El menú .^brir base de datos"de Access tiene la posibilidad de abrir una base de datos definida por ODBC. Si se hace eso con el DSN creado con el controlador ODBC de PostgreSQL, desde Access se pueden manejar las tablas de las bases de datos PostgreSQL que están en el servidor. Esto es casi lo mismo que los proyectos de Access (archivos .adp) con SQL Server.

Clientes de terceras partes
Existen programas de terceras partes, algunos gratuitos y otros comerciales, del estilo a la çonsola de administración"de SQL Server, que permiten gestionar por completo el servidor PostgreSQL creando y modificando tablas, exportando o importando datos, creando procedimientos almacenados, triggers, etc. Todo ello mediante interfaces gráficos de usuario intuitivos y fáciles de usar.

Dada la variedad de software a elegir, se destacan los siguientes programas de administración:

- PgAdmin. Gratuito, desarrollado por los creadores de la base de datos. Más información en http://www.pgadmin.org.
- PostgreSQL manager. Muy recomendable. Es un producto comercial aunque con un coste de licencia accesible. Más información en http://ems-hitech.com/pgmanager.

5.1.2. Procedimientos almacenados

Como se menciona anteriormente el repositorio que se construirá consiste en una base de datos relacional, que contiene las tablas y los procedimientos almacenados necesarios para tener acceso a ellas.

Los procedimientos almacenados forman una API rudimentaria para las clases de Java de el sistema; las clases no necesitan saber sobre el diseño de la tabla, sólo necesitan conocer los procedimientos almacenados que insertan, seleccionan, actualizan, y eliminan información para ellas. Dentro de la arquitectura propuesta los procedimientos almacenados son el nivel de acceso a los datos.

Para cargar un documento XML en el repositorio, la aplicación necesita ejecutar muchas instrucciones de insertar y actualizar. Se pueden construir estas instrucciones SQL sobre la marcha y desde el mismo código de las clases, pero por razón de funcionamiento, es mejor crear algunos procedimientos almacenados. Cuando se ejecuta SQL correctamente, el RDBMS analiza el SQL y crea

un 'plan de consulta'; antes de la ejecución; con un procedimiento almacenado, el plan de consulta será considerablemente más rápido si es determinado sobre la primera ejecución.

Los Procedimiento Almacenados son programas autocontrolados escritos en lenguaje del SMBD y son almacenados como parte de la Base de Datos y sus metadatos.

Una vez creado un procedimiento almacenado, se puede invocar directamente desde una aplicación, o sustituir el nombre de una tabla o vista, por el nombre de procedimiento en cláusulas SELECT. Los procedimientos almacenados pueden recibir parámetros de entrada y retornar valores a la aplicación.

Las ventajas de usar los procedimientos almacenados incluyen[4]:

- Diseño modular.
- Aplicaciones que acceden la misma Base de Datos pueden compartir los procedimientos almacenados, eliminando el código doble y reduciendo el tamaño de las aplicaciones.
- El fácil mantenimiento.
- Cuando un procedimiento se actualiza, los cambios se reflejan automáticamente en todas las aplicaciones, sin la necesidad de recompilar. Las aplicaciones son compiladas sólo una vez para cada cliente.
- Los procedimientos almacenados son ejecutados por el servidor, no por el cliente lo que reduce el tráfico en la red y mejora el desempeño, especialmente para el acceso del cliente remoto.
- Están almacenados en los servidores y asegurados por las medidas tomadas en la instalación, lo que impide que los usuarios normales puedan modificarlos e incluso desconocen su existencia. Este es un elemento de gran valor en lo que a seguridad respecta.

Procedimientos Almacenados en PostgreSQL

Es común que los desarrolladores de aplicaciones utilicen las prestaciones de las bases de datos relacionales modernas, en ocasiones simplemente por desconocer las ventajas que le ofrecen o por desconocer su manejo.

Dentro de PostgreSQL, la base de datos de código abierto más poderosa, se pueden desarrollar funciones en varios lenguajes. El lenguaje PL/pgSQL es uno de los más utilizados dentro de PostgreSQL, debido a que guarda cierta similitud con PL/SQL de Oracle y a su facilidad de uso.

SQL es el lenguaje estándar para realizar consultas a un servidor de base de datos. Cada sentencia SQL se ejecuta de manera individual por el servidor, lo cual implica que las aplicaciones cliente deben enviar cada consulta al servidor, esperar a que la procese, recibir los resultados, procesar los datos y después enviar la siguiente sentencia.

Al usar PL/pgSQL es posible realizar cálculos, manejo de cadenas y consultas dentro del servidor de la base de datos, combinando el poder de un lenguaje procedimental y la facilidad de uso de SQL, minimizando el tiempo de conexión entre el cliente y el servidor [10].

5.1.3. Detalles de la Implementación

Para la implementación de este sistema se utilizó la versión 8.0 de PostgreSQL para Windows que se puede obtener de la página http:///www.postgresql.com, donde se puede descargar todo un entorno de PostgreSQL que contiene el JDBC, el ODBC y el software de administración de la base de datos PgAdmin III.

Anteriormente se describieron las 12 tablas que se necesitan para implementar este sistema: node_type, doc, node, element_name, attribute_name, attribute_value_leaf, text_leaf, pi_target_leaf, pi_data_leaf, entity_reference, comment_leaf y cdata_leaf.

Como se mencionó en el capítulo anterior el acceso a los datos es por medio de procedimientos almacenados. En el ápendice se muestran algunos de los procedimientos almacenados más representativos y a continuación se enlistan todos los procedimientos almacenados con una breve explicación:

- xml_i_d: crea una entrada en la tabla doc y regresa el unique doc_id.
- xml_i_n: crea una entrada en la tabla node y regresa el unique node_id. Nota que en esta etapa, el documento ID, el tipo y el índice x son conocidos, pero el índice y no (este no es especificado aquí).
- xml_i_en: inserta una fila en la tabla element_name para un nodo elemento; el node_id hace referencia a la entrada en la tabla node. Los otros parámetros son el prefijo namespace y el nombre local del elemento.
- xml_i_an: inserta una fila la tabla attribute_name. Cada entrada tiene una referencia a la entrada element_name (por node_id), un atributo unique ID (para este nodo), un prefijo namespace, y un nombre local.
- xml_i_avl: inserta una fila en la tabla attribute_value_leaf; cada entrada tiene una referencia a la tabla attribute_name (por node_id y attribute_id). Los valores largos serán partidos a través de hojas, cada una con un unique leaf_id (para el atributo especificado).

- xml_i_cdl: inserta una fila en la tabla cdata_leaf. Otra vez, los valores largos serán partidos a través de hojas.
- xml_i_cl: inserta una fila representando una hoja en la tabla comment_leaf.
- xml_i_er: inserta una fila en la tabla entity_reference.
- xml_i_pidl: inserta una fila en la tabla pi_data_leaf.
- xml_i_pitl: inserta una fila en la tabla pi_target_leaf.
- xml_i_tl: inserta una fila en la tabla text_leaf.
- xml_s_cdlid: regresa el último valor de leaf_id de la tabla cdata_leaf para el nodo especificado.
- xml_s_tlid: regresa el último valor de leaf_id de la tabla text_leaf para el nodo especificado.
- xml_s_n_last: regresa el valor de node_id del ultimo nodo en el documento especificado que todavía no tiene un sistema de coordenada y.
- xml_u_n_y: coloca el valor y para el nodo especificado.
- xml_u_n_y_null: coloca null en el valor de y para el nodo especificado.
- xml_first_el: regresa el primer valor de x de un nodo del documento especificado.
- xml_start_y: regresa el primer valor de y del nodo cuyo valor de x es el primero.
- xml_serialise_nodes: obtener todos los elementos del documento especificado y los regresa en el orden correcto.

5.2. Clases Java

Las clases de Java proporcionan una capa de interfaz y de control. Son la plataforma básica sobre la cual se va a construir la aplicación para almacenar los documentos XML dentro de la base relacional.

En esta aplicación para cargar un documento XML dentro de la base de datos se necesita:

- Conectar a la base de datos.
- Analizar el documento.
- Determinar las coordenadas del Modelo de Conjuntos Anidados de cada nodo.

Para conectar a la base de datos y almacenar el documento XML, en este caso se utiliza el JDBC de PostgreSQL. Esto se logra colocando el archivo jdbc.jar dentro del directorio j2sdk/jre/lib/ext/.

Para cargar el documento XML, se necesita un programa que pueda leer el documento XML, entender el modelo de contenido, distinguir el contenido de la sintaxis, y fijar las coordenadas (x, y) de cada nodo. El API simple para XML (SAX) proporciona los métodos para el análisis secuencial de los documentos XML, así que lo que se necesita hacer es escribir los métodos para manejar los diferentes eventos o acontecimientos que el programa de análisis SAX invocará en su camino a través del documento.

Un analizador SAX lee el modelo de contenido XML y actúa como un procesador de XML. El ContentHandler analizará la referencia de entidad, no hará caso de comentarios, y tratará secciones de CDATA de la misma manera que texto. Puesto que se desea seguir las referencias de entidad y se necesita mantener la diferencia entre CDATA y texto, se necesita implementar el SAX LexicalHandler también; la interfaz de SAX LexicalHandler proporciona el método para manejar comentarios, y los métodos que denotan el principio y el extremo de las secciones de CDATA. Finalmente, si se desea manejar el contenido de una DTD o de un Esquema XML, se necesita implementar la interfaz de DTDHandler que proporciona métodos con respecto a definiciones de entidad y a declaraciones de notación.

En esta aplicación, se utiliza el parser Xerces que se puede descargar de la página http://xml.apache.org/dist/xerxes-j/. Una vez descargado, se debe descomprimir su contenido y agregar los archivos.jar al directorio j2sdk/jre/lib/ext/.

Para determinar las coordenadas del Modelo de Conjuntos Anidados de cada nodo, se comienza con x = 1 y se incrementa x cada vez que se encuentra un nuevo nodo. Cuando se encuentra un nodo hoja, su coordenada y será x+1 (antes de continuar se necesita aumentar x a y + 1, para el nodo siguiente). Las coordenadas y de nodos no-hojas son poco más difíciles. Cuando se crea el nodo del documento o se encuentra nodos elemento, todavía no se sabe cuál será la coordenada y del nodo, así que se tiene que dejar como valor nulo hasta que pueda ser determinado. Afortunadamente, gracias a que los elementos deben estar jerarquizados correctamente, cada vez que se encuentra una etiqueta de cierre (que causa un evento del endElement de SAX), se puede estar seguro que cierra; es decir, sólo se necesita consultar el último nodo con valor nulo de y de la tabla node, se fija a "x + 1", y "x = y+1", y así sucesivamente, hasta que todos los nodos han sido cerrados (el nodo del documento será el último).

Por último, como ya se mencionó anteriormente la aplicación también almacenará documentos de Esquemas XML y DTDs. En el caso de almacenar un Esquema XML no existe mayor complicación ya que es un documento XML pero en el caso de una DTD que tiene una sintaxis diferente es necesario hacer

una conversión para poder almacenarla dentro de la base de datos.

Las principales clases de control son: xmlrepDB.java, xmlrepSAX.java, scanXML.java, uploadXML.java, extractXML.java, dtd2xsd.java y userXML.java. A continuación se descibe brevemente cada una.

5.2.1. Clase xmlrepDB

La clase xmlrepDB maneja la conexión a la base de datos y proporciona métodos para ejecutar los comandos SQL, el manejo de las cadenas largas que se necesitan partir en hojas, y el manejo de las excepciones que pueden presentarse.

Específicamente esta clase debe:

- Iniciar la conexión al repositorio.
- Determinar el tamaño máximo de la hoja para el nombre del elemento, prefijo de espacio de nombre, y referencia de entidad.
- Cerrar las sentencias, cancelar la transacción actual, y cerrar la conexión al repositorio.
- Ejecutar sentencias SQL que no regresan un resultado tal y como una sentencia de insertar, una de actualizar o una de eliminar.
- Ejecutar sentencias SQL que regresan algún resultado.
- Cortar una cadena en hojas, si excede el parámetro de longitud máximo.
- Manejar excepciones.
- Dar de baja o cancelar la transacción actual.

5.2.2. Clase xmlrepSAX

La clase xmlrepSAX inicializa un parser SAX2 y maneja cualquier excepción que pueda presentarse.

Específicamente esta clase debe:

- Instanciar un parser SAX
- Registrar el manejador de contenido SAX.
- Registrar el manejador de errores SAX.
- Colocar el manejo de espacios de nombre.
- Colocar el manejo de prefijos de espacios de nombre.

- Registrar el Manejador de eventos léxicos SAX.
- Registrar el Manejador de eventos DTD SAX.
- Manejar errores y excepciones.
- Abrir y analizar el documento.

5.2.3. Clase scanXML

Un parser SAX reportará un error si el documento XML dado no esta bien formado, así que se puede evitar sobrecargar la base de datos analizando cada documento para que este bien formados antes de cargarlo. La clase scanXML hará esto, instanciando un parser SAX y proporciona métodos de manejo de contenido vacíos.

5.2.4. Clase uploadXML

La clase uploadXML debe:

1. Instanciar scanXML y analizar el documento para asegurar que este bien formado.
2. Comenzar una transacción en la base de datos.
3. Registrarse como el manejador de contenido para el parser SAX .
4. Asegurar que el proceso de carga del documento tenga éxito o falle como una sola unidad.

La clase necesita los mismos métodos de manejo de contenido que proporciona la clase scanXML, pero esta vez harán algo con el contenido; llama a los procedimientos almacenados para insertar el contenido dentro de la base de datos.

Los métodos de ContentHandler (manejo de contenido):

- startDocument
- endDocument
- startElement
- endElement
- processingInstructions
- characters
- "ignorableWhitespace

Los métodos LexicalHandler:

- starDTD
- endDTD
- comment
- startCDATA
- endCDATA
- startEntity
- endEntity

5.2.5. Clase extractXML

La clase extractXML llama al procedimiento almacenado xml_serialise_nodes y muestra el documento XML con el ID deseado.

Para reconstruir un documento XML del repositorio, se necesita el procedimiento almacenado xml_serialise_nodes para obtener todos los datos juntos de las tablas. Esencialmente, esto implica una consulta grande sobre la unión de todas las tablas. Este procedimiento también necesita incluir los pocos datos de carácter que delimitan los diferentes componentes de XML (por ejemplo, '<">álrededor de nombres del elemento). También necesita asegurar de que todo sea regresado en el orden correcto (esto implica un poco de más esfuerzo que solo ordenarlo por valores de x).

5.2.6. Clase dtd2xsd

La clase dtd2xsd se encarga de realizar la conversión de una DTD a un Esquema XML para poderla almacenar dentro de la Base de Datos.

Específicamente esta clase debe:

- Verificar que la DTD exista.
- Crea un archivo con el mismo nombre de la DTD pero con terminación .xsd para especificar que es un Esquema XML.
- Convertir la DTD a Esquema XML y almacenarlo dentro del archivo creado.

5.2.7. Clase userXML

La clase userXML es la encargada de verificar los datos del usuario que desea utilizar la aplicación.

Específicamente esta clase debe:

- Revisar que los datos del usuario que desea usar la aplicación sean correctos.
- En caso de que alguno de los datos sea incorrecto, no permite la entrada.
- Si los datos son correctos permite la entrada del usuario a la aplicación para poder utilizarla.

Conclusiones

La problemática planteada en la introducción de la tesis de ¿Cómo almacenar dentro de una base de datos relacional documentos XML? Fue abordada a lo largo del desarrollo de esta tesis cumpliendo con el objetivo principal de diseñar y crear una arquitectura genérica que permita almacenar, manipular y extraer documentos XML de de una base de datos relacional.

A lo largo de esta tesis se hizo énfasis en la importancia y los beneficios de ligar las bases de datos relacionales con XML, fundamentando la realización de esta tesis.

En un principio la idea sólo fue diseñar una arquitectura genérica que fuera capaz de almacenar documentos XML dentro de una base de datos relacional y de esta manera poder verificar, consultar y manipular los datos contenidos en estos documentos. Conforme se fue avanzando en el diseño y desarrollo de esta arquitectura se presentó la idea de que también fuera capaz de almacenar DTDs y Esquemas XML, para lo cual en el caso de las DTDs fue necesario realizar algunas conversiones para poderlas almacenar dentro de la base de datos.

Uno de los objetivos planteados desde el principio de esta tesis fue que la arquitectura fuera diseñada y desarrollada con software libre con la finalidad de no estar atada a algún proveedor, dando como resultado una arquitectura independiente de cualquier SMBD.

La aplicación se hizo siguiendo el Proceso Unificado, el cual es un proceso de desarrollo de software configurable que se adapta a través de los proyectos variados en tamaño y complejidad. Se basa en muchos años de experiencia en el uso de la tecnología orientada a objetos en el desarrollo de software de misión crítica.El Proceso Unificado guía a los equipos de proyecto en cómo administrar el desarrollo iterativo de un modo controlado mientras se balancean los requerimientos del negocio, el tiempo al mercado y los riesgos del proyecto. El proceso describe los diversos pasos involucrados en la captura de los requerimientos y en el establecimiento de una guía arquitectónica lo más pronto, para diseñar y probar el sistema hecho de acuerdo a los requerimientos y a la arquitectura. El proceso describe qué entregables producir, cómo desarrollarlos y también provee

patrones. El proceso unificado es soportado por herramientas que automatizan entre otras cosas, el modelado visual, la administración de cambios y las pruebas.

A pesar de la necesidad de almacenamiento eficiente de XML, las bases de datos nativas XML todavía son solamente un porcentaje minúsculo de todas las bases de datos instaladas. ¿Por qué? Principalmente porque las bases de datos relacionales existentes son infraestructura crítica en la mayoría de las organizaciones. Sin embargo, almacenar datos XML en las tablas relacionales es menos directo porque los enfoques XML y relacionales se construyen sobre diversos principios. Las diferencias en los dos enfoques crean una necesidad de herramientas que pueden mapear XML a los datos relacionales.

En un futuro, se cree que un gran número de desarrolladores cambiará a un paradigma de desarrollo basado en XML. Pero este cambio no traerá alrededor una revolución en la tecnología de las bases de datos. De lo contrario, se espera que la tecnología relacional siga siendo la tecnología de almacenamiento dominante, con extensiones importantes a XML y que coexistan con la nueva tecnología de bases de datos nativas XML. Por supuesto, la tecnología de bases de datos nativa XML se presentó debido a las diferencias entre el paradigma de bases de datos relacionales y el paradigma XML. Pero también se cree que así como las técnicas de mapeo objeto-relacional simplifican la persistencia de objetos en los depósitos relacionales, la tecnología de mapeo XML puede tender un puente sobre la brecha para recuperar correctamente datos XML de bases de datos relacionales y datos XML sobre bases de datos relacionales.

Mientras que la industria se pone de acuerdo en como XML y los datos relacionales llegarán a ser intercambiables, convertir XML a/y desde datos relacionales es a menudo difícil. De hecho, cuando un proyecto requiere que los datos relacionales estén convertidos a/o desde formatos complejos de XML (los esquemas o DTDs de XML), no hay una forma definida de hacerlo.

Contribuciones

Toda la problemática antes expuesta, fue abordada en el trabajo realizado a lo largo de la tesis. Se implementó una aplicación que almacena documentos XML dentro de una base de datos relacional basada en una arquitectura genérica y elaborada con software de distribución gratuita.

Otro punto importante que se debe resaltar es que la arquitectura propuesta no ata a un producto o a un vendedor específico.

El diseño de la base de datos planteado en este trabajo permite almacenar cualquier documento XML, lo que hace incecesario crear una base de datos para cada documento XML, solución que proponen algunos SMBD o algoritmos a-

nalizados anteriormente.

Con este sistema se pueden realizar prácticas en cursos que utilicen XML y en donde sea necesario consultar, manipular y actualizar los datos contenidos en un documento XML.

Perspectivas

Aunque la arquitectura planteada permite realizar consultas ad hoc, dependiendo de lo que se desee con instrucciones simples SQL, una de las extensiones que se puede realizar a este trabajo es la parte de las consultas y la manipulación de los datos utilizando la recomendación XPath[1] de W3C.

[1]XPath: Es un lenguaje (basado en XML) que permite seleccionar subconjuntos de un documento XML. La idea es parecida a las expresiones regulares para seleccionar partes de un texto plano. XPath permite buscar y seleccionar teniendo en cuenta la estructura jerarquica del XML.

Apéndice A

Implementación del Sistema

A.1. Ejemplo

Para poder conocer la aplicación y ver su funcionamiento se trabajará con el documento XML Equipos.xml cuyo contenido se muestra a continuación:

```
<?xml version = "1.0"?>
<!-- File name: Equipos.xml -->
<TIENDA>
  <EQUIPO TIPO="Sobremesa" COLOR="Negro">
    <NOMBRE>FX300 UM</NOMBRE>
    <PRECIO>
      <PESETAS>166386</PESETAS>
      <EUROS>1000</EUROS>
    </PRECIO>
    <CARACTERISTICAS CPU="Pentium III" VELOCIDAD="800">
      <MEMORIA>128</MEMORIA>
      <DISCO>15</DISCO>
    </CARACTERISTICAS><
    OPCIONES>Multimedia, Grabadora de CD</OPCIONES>
  </EQUIPO>
  <EQUIPO TIPO="Portátil" COLOR="Plateado">
    <NOMBRE>LT150 HX</NOMBRE>
    <PRECIO>
      <PESETAS>166386</PESETAS>
      <EUROS>1000</EUROS>
    </PRECIO>
    <CARACTERISTICAS CPU="Pentium III" VELOCIDAD="800">
      <MEMORIA>128</MEMORIA>
      <DISCO>15</DISCO>
    </CARACTERISTICAS>
    <OPCIONES>Multimedia, Grabadora de CD</OPCIONES>
  </EQUIPO>
</TIENDA>
```

Para ejecutar la aplicación desde la consola, sólo basta con situarse en el directorio en el que se encuentra el archivo *EnterSystemXML.class* y ejecutarlo para lo cual se tiene que escribir *java EnterSystemXML* como se muestra en la Figura A.1.

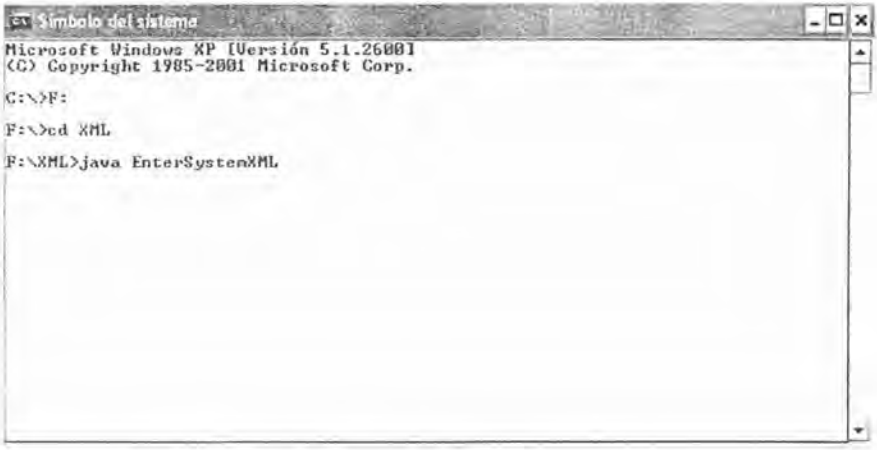


Figura A.1: Ejecución de la aplicación

Una vez ejecutado el archivo EnterSystemXML aparecerá la ventana de inicio de la aplicación (ver Figura A.2).

Figura A.2: Ventana de Inicio

Es necesario introducir un nombre y una contraseña válidos por razones de seguridad, para no permitir el acceso a cualquier usuario, por lo cual la ventana de inicio muestra en la parte inferior dos campos de texto para introducir los datos. Una vez introducidos los datos se debe oprimir el botón *Ingresar Sistema* para poder utilizar esta aplicación (Ver Figura A.3).

Figura A.3: Introducción de Datos

Si el nombre o la contraseña no son válidos aparecerá una ventana con un mensaje indicando que los datos son incorrectos. Para poder volver a intentar el acceso se debe oprimir el botón *Aceptar* de esta ventana (ver Figura A.4).

Figura A.4: Datos incorrectos

Si el nombre y la contraseña son válidos se permite el acceso, en la consola se muestra el número de usuario y aparece una ventana principal desde donde se pueden realizar las funciones principales de la aplicación como se observa en las Figura A.5 y A.6.

Figura A.5: Acceso a la aplicación

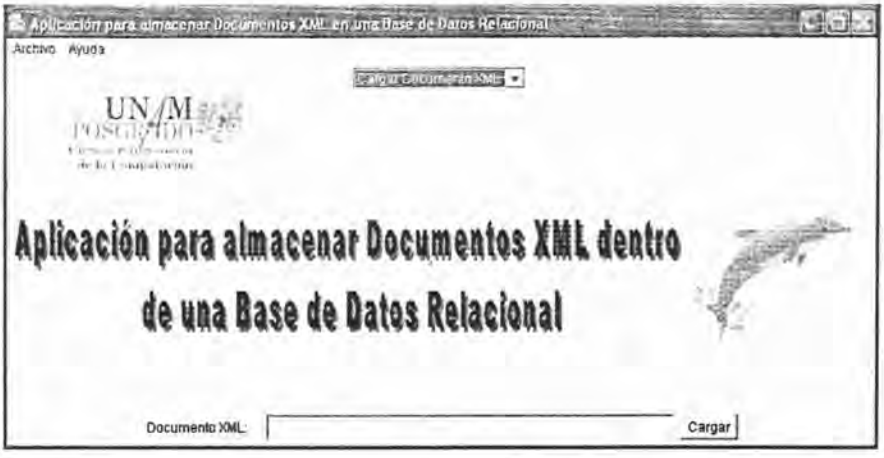


Figura A.6: Ventana Principal

Como se ha mencionado anteriormente, esta aplicación realiza dos funciones principales que son Cargar un documento XML dentro de la base de datos y Extraer un documento XML de la base de datos. Para poder ejecutar cualquiera de estas funciones se debe seleccionar del combo que se encuentra en la parte

superior de la ventana la opción deseada (ver Figura A.7).

Figura A.7: Seleccionar opción

Si lo que se desea hacer es almacenar un documento XML dentro de la base de datos se selecciona del combo la opción *Cargar Documento XML*. En la parte inferior de la ventana aparecerá un campo de texto donde se debe introducir el nombre del documento XML que se desea almacenar y posteriormente se debe oprimir el botón *Cargar* (ver Figura A.8).

Figura A.8: Cargar Documento XML

Para que el documento XML pueda ser almacenado dentro de la base de datos la aplicación debe verificar que el documento XML exista, que esté bien formado y en su caso que sea válido. En la consola se va mostrando el proceso de verificación (Ver Figura A.9).

Figura A.9: Proceso de verificación del documento XML

Si la carga del documento XML fue exitosa en la consola se muestra un mensaje indicando el ID asignado al documento junto con el número de nodos que fueron almacenados dentro de la base de datos (ver Figura A.10).

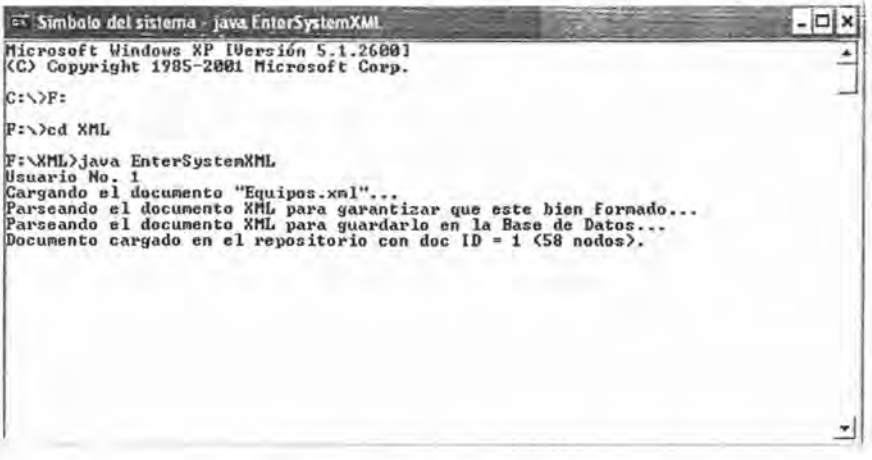


Figura A.10: Carga Exitosa

También se muestra una ventana indicando que la carga fue exitosa junto con el ID del documento. Para poder seguir trabajando con la aplicación es necesario oprimir el botón *Aceptar* de la ventana del mensaje (ver Figura A.11).

Figura A.11: Ventana de Carga Exitosa

Si lo que se desea es extraer un documento XML de la base de datos, se selecciona del combo la opción *Extraer Documento XML* (ver Figura A.12).

Figura A.12: Extraer Documento XML

En la parte inferior de la ventana aparecerá un combo donde se debe seleccionar el documento XML que se desea extraer. Una vez seleccionado el documento se debe oprimir el botón *Extraer* (ver Figura A.13).

Figura A.13: Selección de documento XML

Si la extracción es exitosa en la consola se mostrará el documento XML seleccionado (ver Figura A.11).

Figura A.14: Extracción exitosa

Además de las funciones principales, la aplicación también muestra una pequeña barra de menús en la parte superior con la opción *Archivo* y la opción *Ayuda*.

Si se selecciona la opción *Ayuda* aparecerán dos opciones: *Ayuda!* y *Acerca de...* (ver Figura A.15).

Figura A.15: Menú de Ayuda de la aplicación

Si se selecciona la opción *Ayuda!* se muestra una ventana donde brevemente se explica la función y las instrucciones de uso de la aplicación (ver Figura A.16).

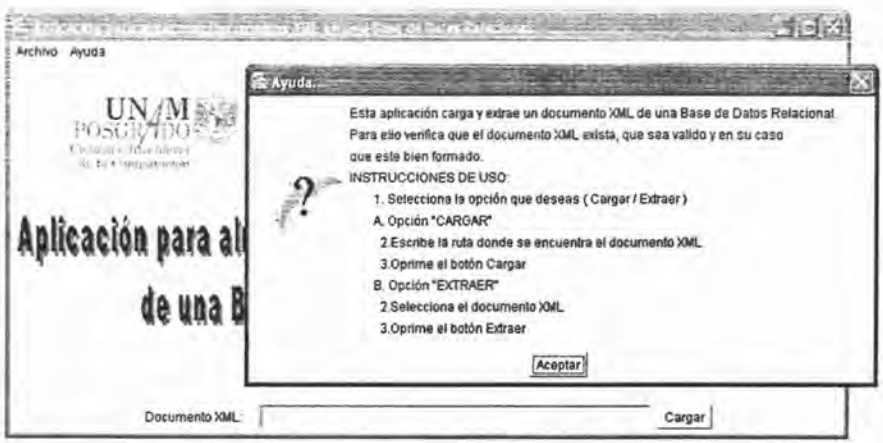


Figura A.16: Ventana de Ayuda

Si se selecciona la opción *Acerca de...* aparecerá una ventana con los datos de creación de la aplicación (ver Figura A.17).

Figura A.17: Ventana Acerca de...

Si lo que se desea es salir de la aplicación se debe seleccionar de la barra de menús la opción *Archivo* y seleccionar la única opción que es *Salir* (ver Figura A.18).

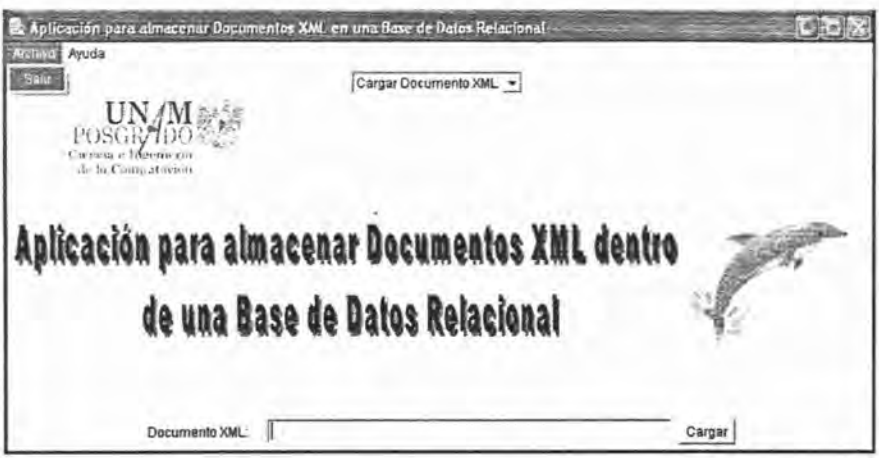


Figura A.18: Menú de Archivo de la aplicación

Aparecerá una ventana para confirmar la salida de la aplicación y si es así se debe oprimir el botón *Aceptar* (ver Figura A.19).

Figura A.19: Salir de la aplicación

Si no se desea salir de la aplicación se debe oprimir el botón *Cancelar* (ver Figura A.20).

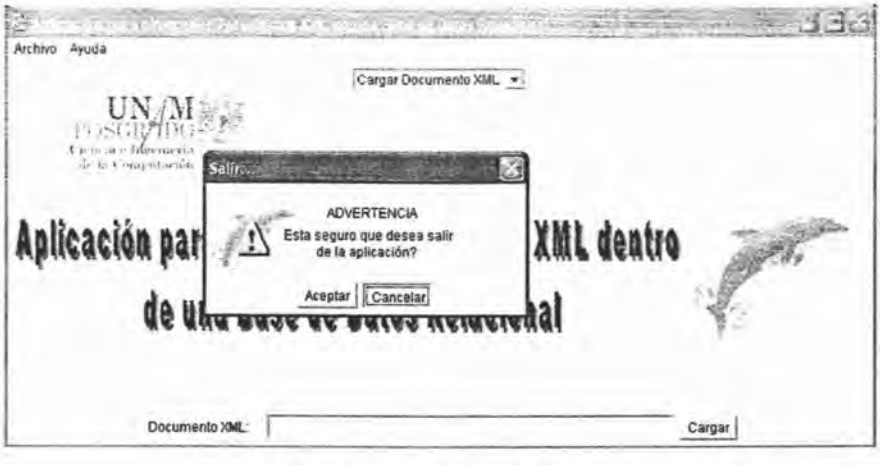


Figura A.20: Cancelar salida de la aplicación

En caso de querer almacenar un documento XML que no esté bien formado, en la consola se mostrará un mensaje de error y aparecerá una ventana indicando que surgió un error durante el análisi; por ejemplo si al documento Equipos.xml se le elimina la etiqueta <TIENDA> el documento no estará bien formado. Ver Figura A.21 y Figura A.22.

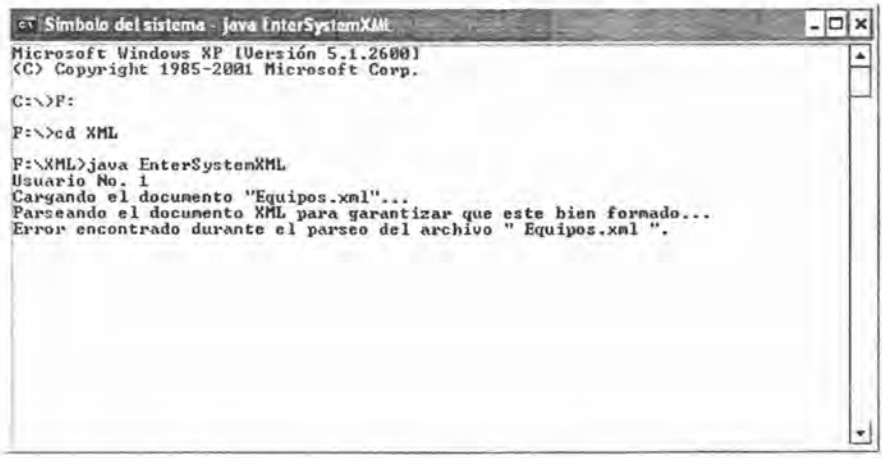


Figura A.21: Documento no bien formado

Figura A.22: Ventana de Error

Para saber cuál fue el error ocurrido, se debe oprimir el botón *Aceptar* y en la consola se mostrará un breve descripción de cuáles fueron las causas que provocaron el error durante el análisis (ver Figura A.23).

Figura A.23: Información del Error

Como se mencionó anteriormente una de las características de la aplicación es poder almacenar los documentos que definen el contenido de un documento XML; en el caso de una DTD se realizá una conversión a Esquema XML.

Para revisar el funcionamiento de la aplicación con una DTD se utilizará el documento XML mensajes.xml con la DTD mensajes.dtd que se muestran a continuación:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE mensajes SYSTEM "mensajes.dtd">
<!--comentario inutil-->
<mensajes>
   <mensaje>
      <para alias="tumismo">tu@tu.casa.es</para>
      <de>yo@mi.casa.es</de>
      <asunto>XML es un lenguaje de marcado</asunto>
      <urgente/>
      <texto>
         <!--comentario inutil-->
         Pues eso, que XML es un lenguaje de marcado...
     &amp;
      </texto>
   </mensaje>
</mensajes>
```

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!-- DTD para mensajes -->
<!ELEMENT mensajes (mensaje+)>
<!ELEMENT mensaje (para,de,asunto?,urgente?,texto?)>
<!ELEMENT para (#PCDATA)>
<!ATTLIST para
alias CDATA "">
<!ELEMENT de (#PCDATA)>
<!ELEMENT asunto (#PCDATA)>
<!ELEMENT urgente EMPTY>
<!ELEMENT texto (#PCDATA)>
```

Se selecciona la opción *Cargar Documento XML* y se introduce el nombre del documento que se desea almacenar en este caso *mensajes.dtd* y se oprime el botón *Aceptar* (Ver Figura A.24). En la consola se mostrará el proceso de conversión a Esquema XML como se ve en la Figura A.25.

Figura A.24: Ventana Cargar DTD

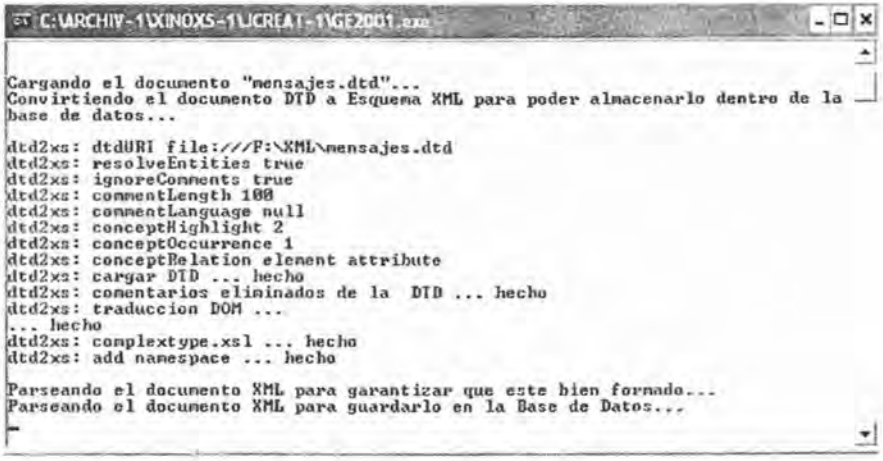


Figura A.25: Cargar DTD

Si la conversión y el almacenamiento fueron exitosos se mostrará el ID del documento y el documento se almacena con el mismo nombre pero con la extensión .xsd, en este caso *mensajes.xsd* Ver Figuras A.26, A.27 y A.28.

Figura A.26: Ventana Carga Exitosa de DTD

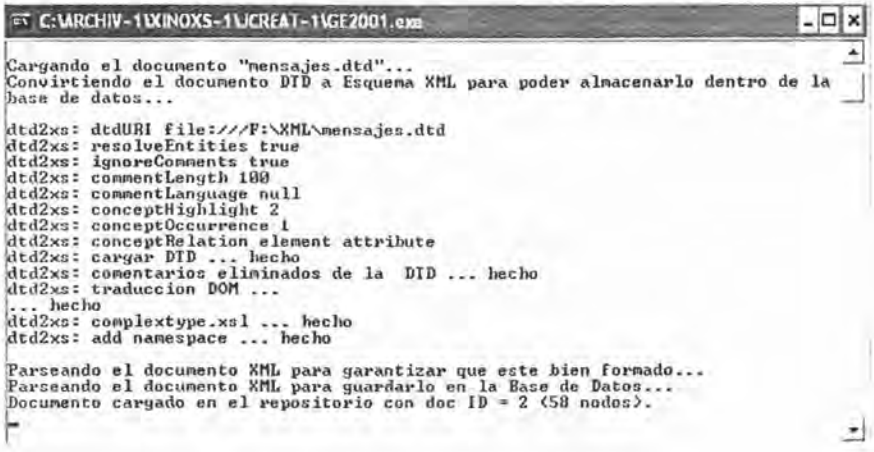


Figura A.27: Cargar Exitosa de DTD

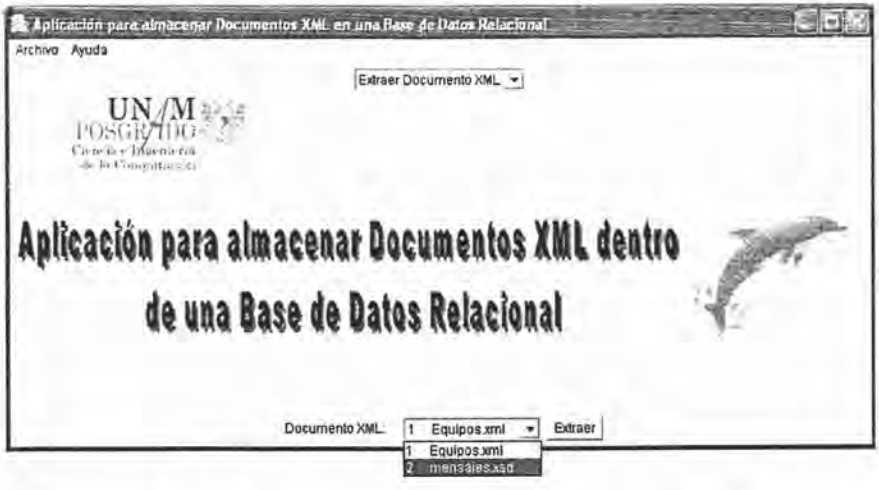


Figura A.28: Almacenar documento con extensión .xsd

A.2. Procedimientos Almacenados

A continuación se presentan algunos de los procedimientos almacenados más representativos utilizados en la aplicación, con una breve explicación y con la definición SQL para PostgreSQL:

- **xml_i_en**: inserta una fila en la tabla *element_name* para un nodo elemento; el *node_id* hace referencia a la entrada en la tabla *node*. Los otros parámetros son el *prefijo del espacio de nombre* y el *nombre local* del elemento.

  La definición de este procedimiento almacenado en PostgreSQL con el leguaje plpgsql queda de la siguiente manera:

```
 CREATE OR REPLACE FUNCTION xml_i_en(t_node_id, t_ns_prefix, t_element_name)
RETURNS int4 AS
$BODY$
BEGIN
INSERT INTO element_name (node_id, ns_prefix, local_name)
VALUES (node, ns, localname);
return 1;
END;
$BODY$
```

- **xml_s_cdlid**: regresa el último valor de *leaf_id* de la tabla *cdata_leaf* para el nodo especificado.

  La definición de este procedimiento almacenado en PostgreSQL con el leguaje plpgsql queda de la siguiente manera:

```
 CREATE OR REPLACE FUNCTION xml_s_cdlid(t_node_id)
RETURNS int4 AS
$BODY$
BEGIN
RETURN max(leaf_id)FROM cdata_leaf WHERE node_id = node;
END;
$BODY$
```

- **xml_u_n_y**: coloca el valor y para el nodo especificado.

  La definición de este procedimiento almacenado en PostgreSQL con el leguaje plpgsql queda de la siguiente manera:

```
 CREATE OR REPLACE FUNCTION xml_u_n_y(t_node_id, t_xy_index)
RETURNS int4 AS
$BODY$
```

```
BEGIN
UPDATE node SET y = y_index WHERE node_id = n;
RETURN 1;
END;
$BODY$
```

Bibliografía

[1] Document type definiton (dtd). Consultado en el World Wide Web: http://www.ulpgc.es/otros/tutoriales/xml/DTD.html.

[2] Introducción a xml. Consultado en el World Wide Web: http://www.desarrolloweb.com/manuales/18/.

[3] Sax. Consultado en el World Wide Web: http://www.saxproject.org/.

[4] Triggers y procedimientos almacenados. Consultado en el World Wide Web: http://www.virtual.unal.edu.co/cursos/sedes/manizales/4060029/lecciones/cap7-4.html.

[5] Tutorial de xml. Consultado en el World Wide Web: http://dat.etsit.upm.es/~abarbero/curso/xml/xmltutorial.html.

[6] APPELQUIST, D. K. *XML and SQL: developing Web applications.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[7] CAMPBELL, C. E., EISENBERG, A., AND MELTON, J. Xml schema. *SIGMOD Rec. 32*, 2 (2003), 96–101.

[8] CHAUDHRI, A., ZICARI, R., AND RASHID, A. *XML Data Management: Native XML and XML Enabled DataBase Systems.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[9] EMERICK, J. Managing xml data storage. *Crossroads 8*, 4 (2002), 6–11.

[10] GROUP, T. P. G. D. Pl/pgsql - sql procedural language, 2000. Consultado en el World Wide Web: http://www.sobl.org/traducciones/postgresql-develdoc/pl-pgsql.html.

[11] HAROLD, E. R. *Processing Xml with Java.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[12] HAUGHEY, T. Modeling hierarchies, 2005.

[13] HILLYER, M. Managing hierarchical data in mysql. Consultado en el World Wide Web: http://www.vbmysql.com/articles/database-design/managing-hierarchical-data-in-mysql.html#part6.

[14] INTERNET, A. Comparativa de bases de datos, 2004. Consultado en el World Wide Web: http://www.arsys.es/soporte/programacion/comparativa.htm.

[15] LEE, D., AND CHU, W. W. CPI: Constraints-preserving inlining algorithm for mapping XML DTD to relational schema. *Data Knowledge Engineering 39*, 1 (2001), 3–25.

[16] LEE, D., MANI, M., CHIU, F., AND CHU, W. W. "Nesting-based Relational-to-XML Schema Translation". In *Int'l Workshop on the Web and Databases (WebDB)* (Santa Barbara, CA, 2001).

[17] LEE, D., MANI, M., CHIU, F., AND CHU, W. W. Net & cot: translating relational schemas to xml schemas using semantic constraints. In *CIKM '02: Proceedings of the eleventh international conference on Information and knowledge management* (New York, NY, USA, 2002), ACM Press, pp. 282–291.

[18] MANI, M., AND LEE, D. Xml to relational conversion using theory of regular tree grammars. In *Proceedings of the VLDB 2002 Workshop EEXTT and CAiSE 2002 Workshop DTWeb on Efficiency and Effectiveness of XML Tools and Techniques and Data Integration over the Web-Revised Papers* (London, UK, 2003), Springer-Verlag, pp. 81–103.

[19] MANI, M., LEE, D., AND MUNTZ, R. R. Semantic data modeling using xml schemas. In *ER '01: Proceedings of the 20th International Conference on Conceptual Modeling* (London, UK, 2001), Springer-Verlag, pp. 149–163.

[20] MARCHAL, B. *XML by Example.* Que Corp., Indianapolis, IN, USA, 1999.

[21] MARUYAMA, H., TAMURA, K., URAMOTO, N., AND TAMURA, K. *XML and Java: Developing Web Applications.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[22] MCLAUGHLIN, B. *Java and XML.* O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2000.

[23] MURATA, M., LEE, D., AND MANI, M. Taxonomy of xml schema languages using formal language theory. In *Extreme Markup Languages* (2001).

[24] THURAISINGHAM, B. M. *Xml Databases and the Semantic Web.* CRC Press, Inc., Boca Raton, FL, USA, 2002.

[25] TULDER, G. V. Storing hierarchical data in a database, April 2003. Consultado en el World Wide Web: http://www.sitepoint.com/print/hierarchical-data-database.