UNIVERSIDAD TECNOLÓGICA DE LA MIXTECA

"SISTEMA DE CONSULTA DE INFORMACIÓN EN DOCUMENTOS ESTRUCTURADOS DE XML"

TESIS

PARA OBTENER EL TÍTULO DE
INGENIERO EN COMPUTACIÓN

PRESENTA

CELIA BERTHA REYES ESPINOZA

DIRECTOR DE TESIS

MC LUIS ANSELMO ZARZA LÓPEZ

HUAJUAPAN DE LEÓN, OAX., MAYO DE 2003

ÍNDICE GENERAL

LISTA DE FIGURAS

LISTA DE DIAGRAMAS

INTRODUCCIÓN

Objetivo

El objetivo de esta tesis es presentar un “Sistema de Consulta de Información en Documentos Estructurados de XML”, (SICIDEX), donde, usando documentos estructurados, se puedan realizar consultas de contenido, es decir, usar la ventaja de tener un documento organizado por etiquetas, permitiendo así obtener el contenido de una etiqueta determinada que cumpla con nuestra búsqueda, considerando que el sistema pueda ser usado fácilmente por el usuario.

Otro objetivo que se persigue es permitir usar el sistema de consulta tanto en la plataforma Linux como Windows, ya que son dos plataformas usadas en la Universidad Virtual y de esta manera los alumnos podrían utilizar esta herramienta de consulta.

Justificación

Una de las principales desventajas de la mayoría de las páginas Web que no utilizan un lenguaje de estructuración de documentos en su elaboración es que se preocupan en mayor medida por la presentación final de la información sin darle importancia al contenido (los datos). A diferencia de los documentos estructurados donde los elementos que lo componen pueden dar información sobre lo que contienen, por ejemplo, si se trata de una definición, párrafo, título, etc.

Tradicionalmente, al buscar información en Internet, los motores de búsqueda proporcionan el servicio sobre una base de datos indexada de páginas Web. Las páginas Web son excelentes cuando se está navegando entre diferentes documentos, pero para encontrar una información exacta

que el usuario originalmente intenta encontrar, no es fácil. Cuando el usuario intenta formular una búsqueda sobre páginas Web (documentos no estructurados) lo que recibe es una cadena que coincide con su búsqueda, pero no le ofrece la posibilidad de determinar el tipo de información que desea consultar.

Cada vez toma más importancia el intercambio y manejo de documentos estructurados, principalmente por la gran cantidad de documentos que se desean compartir en un grupo de personas con intereses en común. Por lo tanto, si se utiliza un lenguaje estructurado para crear los documentos, es posible que el documento pueda ser tratado por un mayor número de herramientas, indexadores, visualizadores, etc.

Al referirse a documentos con estructura, se está refiriendo a documentos cuya estructura es declarada explícitamente de algún modo, ya sea asociando etiquetas a elementos de estructura o mediante la sintaxis con la que se escribe el documento.

Actualmente, los archivos de los cursos que se imparten en la Universidad Virtual de la Universidad Tecnológica de la Mixteca (http://virtual.utm.mx) están formateados como páginas Web. Se han estructurado algunos archivos de las materias de la Universidad Virtual en XML, basándose en un conjunto de etiquetas comunes entre ellos, como un ejemplo de lo que podría implementarse a futuro en la Universidad Virtual, si es que se decidiera migrar el formato de los actuales documentos de los cursos. Se pueden encontrar etiquetas que indican qué información contiene un documento ya sea una definición, párrafo, lista, título, autor, fecha de elaboración, etc.

Por ejemplo, en un documento de la materia de Tecnología de Agentes, donde se quisiera encontrar la palabra "agente", la cual obviamente está incluida en las etiquetas de título, párrafo en general, definición, etc, en un archivo no estructurado, los resultados de la búsqueda se incluirían toda coincidencia con la palabra "agente" independientemente de su ubicación. Al realizar la misma consulta sobre un documento estructurado e

indicando que sólo se quiere encontrar la palabra "agente" sobre la etiqueta "definición", no considerará aquellas palabras "agente" incluidas en otras etiquetas, como párrafo, título, etc., y se obtendrá un resultado específico a la consulta.

Este sistema podría servir de apoyo a los alumnos de la Universidad Virtual en la consulta de datos específicos sobre la gran cantidad de documentos que de cada materia reciben.

Organización de la tesis

La organización de la tesis es la siguiente:

En el capítulo 1 “Lenguajes de definición de documentos” se presenta el marco histórico del origen y desarrollo de XML como un lenguaje de estructuración de documentos, presentando las características particulares de cada uno de los lenguajes, así como sus ventajas y desventajas ante la estructuración de documentos.

En el capítulo 2 "Lenguajes de búsqueda en XML", se explican las características de algunos de los principales lenguajes que actualmente se usan para realizar consultas sobre documentos estructurados. Además, en este capítulo se explica la importancia de procesar y analizar un documento estructurado.

En el capítulo 3 "Implementación del sistema", se presentan las características del sistema y de las herramientas usadas para su elaboración.

En el Apéndice A se presenta el glosario de acrónimos.

En el Apéndice B se incluye el manual del usuario, en el cual se explica el funcionamiento del sistema así como las instrucciones a seguir para poder utilizarlo.

En el Apéndice C se presenta un ejemplo simplificado de un documento estructurado en XML incluyendo su hoja de estilo así como su declaración de tipo de documento.

En el Apéndice D se desglosa un análisis de las herramientas consideradas en la realización del sistema SICIDEX, explicando las características, ventajas y desventajas del uso y aplicación de cada una de ellas.

Capítulo 1

LENGUAJES DE DEFINICIÓN DE DOCUMENTOS

Existen diferentes lenguajes de estructuración de documentos; son aquellos que insertan etiquetas en los documentos para delimitar los elementos de su estructura. Entre estos lenguajes destacan tres: SGML (*Standard Generalized Markup Language*), HTML (*HyperText Markup Language*) y XML (*eXtensible Markup Language*).

En 1969, Ed Mosher, Ray Lorie y Charles F. Goldfarb, investigadores de IBM crearon el primer lenguaje de marcado GML (Generalized Markup Language). [Birbeck,01]

GML después de su revisión y aprobación por el ISO (*International Organization for Standarization*), fue publicado como el estándar ISO 8879:1986, denominándose SGML. Este estándar se utilizó para diseñar lenguajes de marcas específicos, cuyos ejemplos más conocidos son el HTML y el RTF (*Rich Text Format*). [Bryan,97]

En 1989, a partir de un simple ejemplo de tipo de documento según la norma SGML se desarrolló una versión de hipertexto llamada HTML, como solución para publicar las investigaciones de muy diversas fuentes y autores que se producían en el CERN (*European Organization for Nuclear Research*). [Golfarb,99]

1.1 Standard Generalized Markup Language

SGML es un metalenguaje con el cual se pueden definir lenguajes para definir la estructura y el contenido de documentos. La definición de la estructura y el contenido de un tipo de documento se realiza en un DTD (*Document Type Definition*), en el cual se definen los elementos que conformarán el tipo de documento y cómo tiene que estar organizado para que sea correcto. Un DTD describe una clase o familia de documentos con ciertas características comunes. [Golfarb,99]

SGML permite el intercambio de documentos entre diferentes plataformas, ya que proporciona una sintaxis que permite describir las partes que componen un documento por medio de etiquetas.

Para SGML una unidad de texto es un elemento. Cada elemento tendrá un nombre determinado, pero SGML no proporciona ninguna forma de expresar el significado de un tipo particular de elemento, si no que expresa su relación con elementos de otros tipos. El lenguaje propone conjuntos de etiquetas que permiten elegir nombres para los elementos y para documentar su uso en el marcado del texto.

En un documento estructurado, cada elemento debe estar etiquetado de alguna forma. El estándar es insertar una etiqueta al inicio del elemento y otra al final. La etiqueta de inicio tiene el nombre del tipo de elemento encerrado entre los símbolos > y <, como por ejemplo **<elemento>**, mientras la etiqueta final incluye una barra inclinada de la forma **</elemento>**.

1.2 HyperText Markup Language

Dan Connolly y Karen Olson Muldrow crearon el HTML 2.0, teniendo tal aceptación y crecimiento que se organiza la *First International WWW Conference* en Mayo de 1994. [Berners,95]

La versión 2.0 de HTML incorpora a la sintaxis ya existente los formularios, los cuales le permiten al usuario enviar información al servidor y que ésta sea recogida y procesada en el mismo, permitiendo que por medio del uso de campos los usuarios puedan escribir y usar menús en páginas Web. [Berners,95]

La forma en que se visualizará un documento HTML, dependerá del navegador que se utilice ya que HTML se encarga de describir el formato de la página y depende del navegador la interpretación del mismo, independientemente del tipo de máquina desde donde se acceda al documento.

HTML contiene una sintaxis muy sencilla que permite que por medio de la aplicación de una serie de etiquetas se controle el aspecto que la información tendrá en un navegador. Para escribir un documento HTML se puede utilizar desde un editor de texto ASCII hasta editores especializados en el diseño de páginas HTML como FrontPage de Microsoft o Netscape Composer. Dado que HTML es una aplicación de SGML, las etiquetas que se utilizan en el diseño de una página HTML son etiquetas basadas en el estándar SGML. Las etiquetas pueden crearse utilizando mayúsculas y minúsculas sin distinción, como por ejemplo las etiquetas <p> y <P> serán identificadas por el navegador como la misma etiqueta. Existen etiquetas que controlan la presentación del texto en el documento. Otras etiquetas definen las partes que forman al documento.

1.3 eXtensible Markup Language

En 1996, el W3C, principal organización en el diseño de tecnologías relacionadas con el WWW (*World Wide Web*), desarrolló el lenguaje de marcado XML. [W3C,98]

Algunos de los objetivos planteados por el Grupo de Trabajo XML y el W3C son:

XML será directamente empleado en Internet.
XML soportará una amplia variedad de aplicaciones.
XML será compatible con SGML.
Será fácil escribir programas que procesen documentos XML.
El número de características opcionales en XML deberá mantenerse en un mínimo absoluto, idealmente cero.
Los documentos XML deberán ser legibles y razonablemente claros.
El diseño de XML deberá ser preparado rápidamente.
El diseño de XML será formal y conciso.
Los documentos XML serán fáciles de crear.
La brevedad en las marcas XML es de mínima importancia.

En 1998 XML se volvió una recomendación del W3C, significa que XML ha sido revisado y aprobado por los miembros del W3C. Por lo tanto, se dice que XML es estable y realmente válido para su desarrollo.

Algunas de las ventajas que aporta la utilización de XML son:

XML es un lenguaje abierto, por lo tanto se pueden crear etiquetas adecuadas a un documento.
Con la utilización de etiquetas para estructurar un documento, los datos se encuentran en total disponibilidad de ser manipulados por algún otro lenguaje.

Se puede utilizar un estándar para la creación de documentos que pueden compartirse con la seguridad de que la información contenida en los mismos, será fácil de clasificar y procesar.

Los principales usos de XML son:

Aplicación en sitios de Web separando contenido y presentación, y que los mismos datos se puedan mostrar de varias formas distintas.
La utilización de XML para la comunicación entre aplicaciones con una representación de los datos muy simple y fácil de transmitir por la red.
XML para la configuración de programas: representando los datos en forma simple y estándar.

Una característica de XML es que el usuario pueda crear sus propios elementos para estructurar los documentos, por lo tanto se requiere que todos los usuarios que compartirán dichos documentos se pongan de acuerdo en el establecimiento de los requerimientos de acuerdo a sus necesidades.

Actualmente existen estándares terminados y algunos en desarrollo que pueden utilizarse en diversas ramas industriales y sociales, permitiendo la aplicación de un estándar en común dentro de un grupo específico de trabajo, como por ejemplo:

Gráficas por computadora
Multimedia
Matemáticas
Comunicaciones
Etc.

De esta forma se puede utilizar un estándar y adaptarlo a las necesidades propias. Además existen varias propuestas para actividades similares que pueden ser complementarias a la actividad de interés.

1.4 Propiedades de los documentos

Un tipo de documento queda definido por sus elementos, por éstos se puede decir que es fácil identificar los documentos que pertenezcan a la categoría de las cartas, novelas o agendas. Por tanto, si dos documentos contienen elementos totalmente distintos o permiten muy diversas combinaciones de elementos, entonces es posible que no se ajusten al mismo tipo de documento.[Golfarb,99]

Un DTD es un conjunto de reglas que definen la forma en que los datos deberán ser estructurados en un documento XML. Al usar un DTD no sólo permite determinar las reglas de sintaxis y estructura que regirán a los documentos XML, sino que también permite que los datos sigan sus propias reglas basándose en su estructura y contenido.

El DTD puede incluirse dentro de la declaración del documento, definirse en un documento externo o se puede utilizar una mezcla de las dos.

En XML existen algunos conceptos respecto a la validez de un documento: Documento válido, Documento no válido y Documento bien formado.

1.4.1 Documentos bien formados

Los documentos que son creados basándose en la sintaxis básica de XML se conocen como documentos bien formados. Tales documentos pueden usarse sin un DTD que describa su estructura. Por ejemplo, si un navegador encuentra un documento sin un DTD, tiene que ser capaz de comprender la estructura del documento. Un documento bien formado contiene uno o más elementos correctamente anidados uno dentro del otro. *Document* es el elemento que contiene a los demás elementos que

conforman el documento. Todos los elementos forman un simple árbol jerárquico de tal forma que mantienen una relación entre los elementos.

Como se puede observar en la Figura-1.1, se declara que el documento conforma una sintaxis XML 1.0 (version=”1.0”) y que no requiere entidades externas (standalone=”yes”).

```
<?xml version=”1.0” standalone=”yes”?>
<articulo>
      <parrafo>Ejemplo de un documento sin DTD</parrafo>
</articulo>
```

Figura-1.1 Ejemplo de un documento XML sin DTD

Los elementos pueden contener atributos cuyos valores deberán ir entre comillas. Si existiera algún elemento vacío, es decir, que no tiene una etiqueta de final como por ejemplo la etiqueta
, deberá terminar con />, de esta forma quedaría la etiqueta como
 o se puede añadir una etiqueta de final para convertirse así en un elemento no vacío,
</br>.

Según la especificación XML del W3C [W3C,98], un documento XML está bien formado si:

La etiqueta de inicio tiene su correspondiente etiqueta de final (o se trata de una etiqueta vacía).
Cumple con la sintaxis de la especificación XML.
Las etiquetas no se traslapan.
Los atributos tienen nombres únicos.
Los elementos forman un árbol jerárquico, con un único nodo raíz.
No contiene referencias a entidades externas (a menos que se proporcione un DTD).

La Figura-1.2 no es un documento XML bien formado ya que no contiene ningún elemento.

```
Mi primer documento XML
```

Figura - 1.2 Ejemplo de un documento XML que no está “bien formado”

La Figura - 1.3 al contener al menos el elemento <articulo> es un documento bien formado.

```
<?xml version=”1.0” encoding=”utf-8” standalone=”yes”?>
<articulo>Mi primer documento XML
</articulo>
```

Figura - 1.3 Ejemplo de un documento XML con un elemento

La Figura - 1.4 no es un documento XML bien formado, ya que no contiene un único elemento raíz, en este caso contiene dos elementos raíz <material>.

```
<?xml version=”1.0” encoding=”utf-8” standalone=”yes”?>
<material>
   <parrafo>Primer elemento del material </parrafo>
   <material>
       <parrafo>Otro elemento del material </parrafo>
   </material>
</material>
```

Figura - 1.4 Ejemplo de un documento XML con dos elementos raíz

En la Figura - 1.5 al convertirse el elemento <documento> en el elemento raíz, ser único y no formar parte del contenido de ningún otro elemento es un documento bien formado.

```
<?xml version=”1.0” encoding=”utf-8” standalone=”yes”?>
<material>
   <parrafo>Primer elemento del material </parrafo>
   <parrafo>Segundo elemento del material</parrafo>
</material>
```

Figura - 1.5 Ejemplo de un documento con un único elemento raíz

La Figura-1.6 es incorrecta, ya que la etiqueta inicio del elemento <enfasis> está dentro del contenido del elemento <parrafo>, pero su etiqueta de final está fuera, es decir las etiquetas no están correctamente anidadas.

```
<?xml version=”1.0” encoding=”utf-8” standalone=”yes”?>
<articulo>
   <parrafo>Mi primer <enfasis> documento XML </parrafo></enfasis>
   <parrafo>Mi primer documento XML </parrafo>
</articulo>
```

Figura-1.6 Ejemplo de etiquetas mal relacionadas

En la Figura-1.7 se muestra la forma correcta de relacionar las etiquetas

```
<?xml version=”1.0” encoding=”utf-8” standalone=”yes”?>
<articulo>
   <parrafo>Mi primer <enfasis>documento XML </enfasis></parrafo>
   <parrafo>Mi primer documento XML </parrafo>
</articulo>
```

Figura-1.7 Ejemplo con etiquetas correctamente relacionadas

XML, trata a las mayúsculas y las minúsculas como elementos diferentes. Si un elemento de XML está definido como <imagen>, no se puede usar <Imagen> para referirse a él. Normalmente se suelen usar únicamente minúsculas para los nombres de las marcas y de sus atributos.

1.4.2 Documentos válidos

Un documento XML válido cumple con la sintaxis, estructura y reglas que son definidas en un DTD, de tal manera que se verifica que se respete la relación entre los elementos que conforman el documento.

Actualmente existen varios editores que permiten la creación de documentos XML, mostrando la lista de los elementos que conforman el documento, así como las delimitaciones de los mismos. Si el documento XML es creado por un medio que no identifica la sintaxis XML, se capturan las etiquetas manualmente y se validan una vez terminado el documento.

Se puede validar un documento contra un DTD especificándolo en la declaración del documento. Un navegador o un editor de documentos XML puede acceder al DTD ya sea localmente o a través de la red mediante un identificador público precedido por la trayectoria donde se encuentra el DTD.

La Figura-1.8 contiene un identificador público además del URL donde se encuentra el DTD.

```
<?xml version=”1.0”?>
<!DOCTYPE material SYSTEM ”http://virtual.utm.mx/material.dtd”>
<material>
<parrafo>Primer párrafo de un documento XML</parrafo>
</material>
```

Figura-1.8 Ejemplo de una declaración de DTD externo

En la Figura-1.9 se indica la trayectoria donde se encuentra el DTD en el mismo sistema de archivos.

```
<?xml version=”1.0”?>
<!DOCTYPE material SYSTEM ”material.dtd”>
<material>
…
</material>
```

Figura-1.9 Ejemplo de una declaración de DTD en el mismo sistema de archivos

En la Figura-1.10 se incluye el DTD dentro del documento.

```
<?xml version=”1.0”?>
<!DOCTYPE material
[
<!ELEMENT material (titulo,fecha)>
<!ELEMENT titulo (#PCDATA)>
<!ELEMENT fecha (#PCDATA)>
]>
<material>
…
</material>
```

Figura-1.10 Ejemplo de DTD incluido en el documento XML

En la Figura-1.10 las instrucciones que se encuentran entre los corchetes son una parte interna del documento y serán válidos únicamente para el documento que lo contiene.

1.4.3 Declaración de documentos

La primera línea de un documento XML debe ser una declaración XML, como se muestra en la Figura-1.11:

```
<?xml version=”1.0” encoding=”utf-8” standalone=”yes”?>
```

Figura-1.11 Ejemplo de la primera línea de un documento XML

En la Figura-1.11 los primeros carácteres **<?XML** indican que es una instrucción de proceso. Todas las instrucciones están reservadas para el lenguaje. En esta instrucción se encuentran definidos tres parámetros:

La versión, su valor actualmente tiene que ser de “1.0” (no se ha definido ninguna otra versión hasta el momento). Este parámetro se utilizará para especificar futuras versiones de XML.

La declaración encoding identifica el conjunto de carácteres con que esta codificado el documento, por ejemplo, “UTF-8”, “UTF-16” o “ISO-8859-1” (Latin-1), que permite identificar los carácteres incluidos en el documento.

La declaración standalone es opcional y su valor puede ser “yes” o “no”. Un “yes” significa que todas las declaraciones de entidades requeridas están contenidas dentro del documento, y “no” significa que se requiere de un DTD externo.

1.4.3.1 Declaración de tipo de documento en XML

Los documentos XML tienen una estructura lógica y física. En la estructura física el documento XML está formado por entidades, donde una entidad puede hacer referencia a otras para ser incluidas en el documento. En la estructura lógica, el documento está compuesto por elementos, comentarios, referencias e instrucciones de proceso.

En el Diagrama-1.1 se muestra un ejemplo de la estructura de un DTD, con un único elemento principal <material>, el cual puede contener a otros elementos.

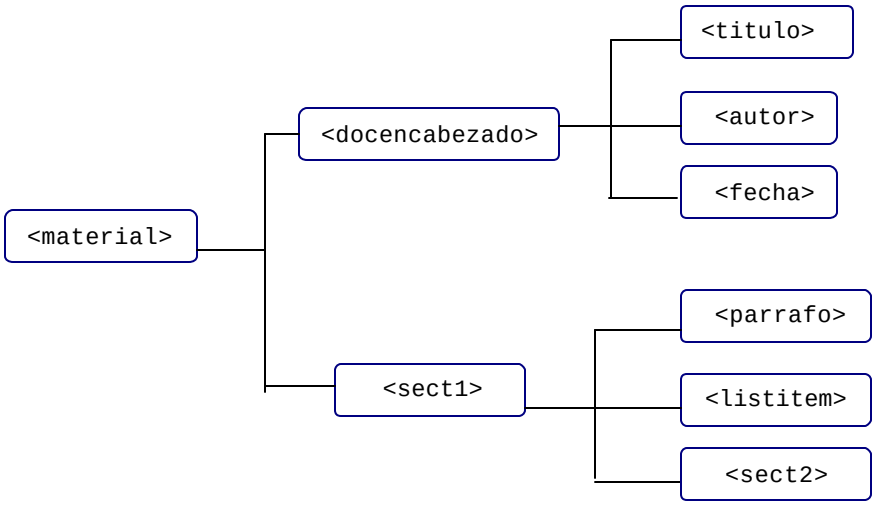

Diagrama-1.1 Estructura de un DTD vista en forma gráfica

El DTD define el número de veces que un elemento puede aparecer y el orden de las mismas dentro de la estructura del documento. Los elementos pueden ser creados de acuerdo a las necesidades propias de los usuarios.
Un documento XML puede asociarse con un DTD (el cual puede ser un subconjunto interno o externo), usando la declaración DOCTYPE. Esta declaración debe aparecer sólo una vez dentro del documento XML, tiene que seguir a la declaración del documento XML y preceder a cualquier elemento o declaración de datos.

En la Figura-1.12 se muestra la declaración de un DTD como subconjunto externo del documento, es decir, el DTD se encuentra en un archivo externo el cual puede ser compartido por diversos documentos.

```
<!DOCTYPE material SYSTEM ”material.dtd”>
```

Figura-1.12 Ejemplo de declaración de un DTD como subconjunto externo

Una o más declaraciones del DTD pueden ser incluidas en un subconjunto interno, el cual está contenido dentro de la declaración DOCTYPE. Estas declaraciones son limitadas con los corchetes [ ], como se muestra en la Figura-1.13.

```
<!DOCTYPE articulo
[
<!ELEMENT articulo (titulo,autor)>
<!ELEMENT titulo (#PCDATA)>
<!ELEMENT autor (#PCDATA)>
]>
<articulo>
<titulo>Material de la primera semana</titulo>
<autor>Jose Rivera</autor>
</articulo>
```

Figura-1.13 Ejemplo de un DTD como subconjunto interno

Si en un documento se utiliza tanto un subconjunto interno y uno externo, el subconjunto interno domina sobre el externo. En la Figura-1.14 se muestra un ejemplo de un documento con un DTD externo.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<?xml-stylesheet type="text/css" href="css-material.css"?>
<!DOCTYPE articulo SYSTEM ”material.dtd”>
<articulo>
   <artencabezado>
      <titulo>Teoría avanzada de las bases de datos</titulo>
        <autor>
             <nombre>Yuri</nombre>
             <apellido>Paramonov</apellido>
        </autor>
   </artencabezado>
   <orderedlist>
      <listitem>Fundamentos subyacentes de las bases
      relacionales;</listitem>
          <itemizedlist>
                  <listitem>Modelos de los datos relacionales;
                  </listitem>
          </itemizedlist>
      </listitem>
   </orderedlist>
</articulo>
```

Figura-1.14 Ejemplo de un documento XML con un DTD externo

En la Figura-1.15 se muestra un ejemplo de un DTD.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!ELEMENT articulo (artencabezado, (parrafo | sect1 |%listtype;)*) >
<!ELEMENT artencabezado (titulo, autor, (fecha | resumen)*) >
<!ELEMENT titulo (#PCDATA | %chardatos;)* >
<!ELEMENT resumen (parrafo+) >
<!ELEMENT autor (nombre,apellido,email?)>
<!ELEMENT nombre (#PCDATA) >
<!ELEMENT apellido (#PCDATA) >
<!ELEMENT email (#PCDATA) >
<!ELEMENT fecha (#PCDATA) >
<!ELEMENT orderedlist (listitem+) >
<!ELEMENT listitem (#PCDATA | %listtype;)* >
<!ENTITY % listtype " itemizedlist | orderedlist " >
```

Figura-1.15 Ejemplo de un DTD

En la Figura-1.15 se muestra un ejemplo de un DTD, donde la definición de los elementos se delimitan usando los signos < y >, seguido del signo de admiración (!) y de una palabra clave además de la especificación de sus parámetros. En este caso se indica que el elemento principal es articulo, el cual contiene un artencabezado y puede contener (cero o más veces) parrafo, sect1 o listtype. En la siguiente línea se especifica que el elemento artencabezado debe contener titulo y autor y puede contener o no fecha o resumen. En las siguientes líneas se continúan especificando los elementos requeridos así como el tipo de contenido de los mismos.

Como se puede observar, los usuarios pueden crear su propia estructura para definir un documento adecuado a sus necesidades, pero esto también puede convertirse en un inconveniente, ya que para documentos de una misma actividad pueden existir varios DTDs. Por este motivo se han desarrollado DTDs estándares, que pueden ser adoptados por un grupo de personas con intereses similares y de esta forma compartir las mismas etiquetas.

A continuación se describen algunos DTDs; varios continúan en desarrollo.

CML (Chemical Markup Language), Metodología para manipular información molecular. www.xml-cml.org
MathML (Mathematical Markup Language), Descripción de notación matemática. www.w3.org/TR/REC-MathML
GedML (Genealogical Data in XML), Manipulación de datos genealógicos. users.iclway.co.uk/mhkay/gedml
SMIL (Synchronized Multimedia Integration Language), Presentación sincronizada de multimedia. www.w3.org/TR/REC-smil
SVG (Scalable Vector Graphics), Candidato a recomendación, descripción de gráficos de dos dimensiones. www.w3.org/TR/2000/CR-SVG-20001102
OSD (Open Software Description), Usado para describir paquetes de software y sus dependencias para clientes heterogéneos. www.w3.org/TR/NOTE-OSD.html
QAML (Question and Answer Markup Language), Lenguaje de marcas para preguntas y respuestas. www.ascc.net/xml/resource/gaml-xml.dd
WML (Wireless Markup Language), Se usa para desplegar en navegadores WAP. www.w3schools.com/wap/wap-intro.asp

El DTD DocBook contiene un conjunto de características y reglas que permiten la escritura de documentación técnica (por ejemplo, libros, artículos, etc.). Originalmente fue escrito en SGML, en la actualidad se encuentra disponible en formato XML, cuenta con páginas de estilo para producir HTML, TeX y otros formatos [Walsh,99].

Se utilizó DocBook para diseñar un ejemplo de estructura para los documentos de las clases de la Universidad Virtual de la Universidad Tecnológica de la Mixteca. Debido a que no se utilizarían todas las características que contiene DocBook, se creó una estructura especial para el diseño de los documentos de prueba para SICIDEX (Apéndice C.2).
La estructuración de los documentos se realizó en base a los elementos más comunes entre ellos. Son varios los elementos que comprenden la

estructura algunos de los elementos más representativos son: título, nombre, fecha, sección, párrafo, lista, gráfico, comando, etc.

En el Apéndice D se explica la forma en la que se crearon los documentos de prueba así como las herramientas que se utilizaron.

Capítulo 2

LENGUAJES DE BÚSQUEDA EN XML

2.1 Lenguajes de búsqueda en XML

Consultar documentos XML representa un nuevo desafío, dada la perspectiva de que grandes volúmenes de datos estarán disponibles en este formato, generando la necesidad de herramientas de consulta que sean lo suficientemente flexibles para entender la heterogeneidad de los documentos y poder extraer información con rapidez y exactitud.

Un lenguaje de búsqueda se usa para extraer información de documentos XML. Idealmente, se espera que dicha extracción de información se realice mediante un lenguaje de búsqueda que se caracterice por su alto poder de expresión y facilidad de uso. Todavía no hay un lenguaje de búsqueda estándar para XML. Actualmente existen diversos desarrollos de algunos lenguajes que pueden convertirse en muy poco tiempo en un estándar.

A continuación se presentan algunas características de un lenguaje de búsqueda[Deutsch,99]:

- Semántica precisa. Un lenguaje de búsqueda XML debería tener una semántica formal. La formalización necesita ser suficiente para soportar argumentos sobre búsquedas XML, tales como determinar la estructura del resultado, equivalencia y contención.

- Habilidad para re-escribir. Los datos XML generalmente son generados automáticamente desde otros formatos: relacionales, orientados a objetos, formatos de propósito especial. Cuando los datos XML son nativos y son procesados por un procesador de búsqueda, las búsquedas XML necesitan ser optimizadas, como las búsquedas SQL sobre datos relacionales.
- Operaciones de búsqueda. Las diferentes operaciones que tienen que ser soportadas por un lenguaje de búsqueda XML son: selección (eligiendo un elemento de un documento basado en el contenido, estructura o atributos), extracción (atrayendo determinados elementos de un documento), reducción (eliminando subelementos seleccionados de un elemento), reconstrucción (construyendo un nuevo conjunto de elementos para mantener los datos encontrados) y combinación (reuniendo dos o más elementos dentro de uno).
- El significado de una expresión debería ser la misma independientemente de donde aparezca.
- Un lenguaje de búsqueda debería poder ser usado en datos XML donde no exista un DTD conocido con anterioridad. Los datos están estructurados de forma en que se describen a sí mismos, y debería ser posible para una búsqueda XML confiar en su evaluación. Esta capacidad significa que las búsquedas XML pueden ser usadas contra un código fuente con limitados conocimientos de la precisa estructura de sus documentos.
- Inversamente, cuando el DTD está disponible, debería ser posible determinar si una búsqueda XML está correctamente formada relativo a los DTDs. Esta capacidad puede detectar errores en tiempo de compilación más fácilmente que en tiempo de ejecución.
- Las búsquedas XML deberían ser capaces de preservar el orden y la asociación de elementos en datos XML, si es necesario. El orden de los elementos en un documento XML puede contener información importante que una búsqueda no debería perder. En forma similar, el agrupamiento de subelementos dentro de elementos es usualmente significativo. Por ejemplo, si una búsqueda XML extrae los subelementos <titulo> y <autor> de los elementos <libro> en unos

datos bibliográficos, se debería de preservar las asociaciones <titulo><autor>.

- Las búsquedas XML deberían ser mutuamente incrustadas con XML. Esto es, una búsqueda XML debería ser capaz de contener arbitrariamente datos XML, y un documento XML debería ser capaz de soportar búsquedas arbitrarias.
- Soportar nuevos tipos de datos. Un lenguaje de búsqueda debería tener un mecanismo de extensión para condiciones y operaciones específicas para un particular tipo de datos. Las operaciones especializadas para seleccionar diferentes tipos de contenidos multimedia son un ejemplo.
- Una búsqueda XML debería ser adaptable para su procesamiento del lado del servidor. Mientras una búsqueda XML pueda incorporar variables desde un contexto local, debería tener la forma de "obligar" a la búsqueda XML para que pueda ser ejecutada en forma remota sin necesidad de comunicarse con su punto de origen.
- Las búsquedas XML deberían ser sensibles al ser creadas y manipulados por los programas. La mayoría de las búsquedas no han sido escritas directamente por los usuarios o los programadores. Más bien, serán construidas a través de interfaces de usuario o herramientas en ambientes de desarrollo de aplicaciones.

2.1.1 XQL

XQL le ofrece a los programadores características de funcionalidad de búsqueda que se han estado usando en el mundo de las bases de datos, incluyendo[Jansz,00]:

- Funcionalidad equivalente a la sentencia WHERE de SQL.
- Operadores lógicos (por ejemplo AND, OR, NOT).
- Operadores de comparación
- Operadores "comodines" (por ejemplo *).

XQL. La sintaxis básica de XML imita la sintaxis de navegación de directorio de hipervínculos, pero en lugar de especificar la navegación a través de una estructura física de archivos, la navegación es a través de elementos del árbol XML.[Derksen,99]

- Una simple cadena es tomada para ser un nombre de elemento. Por ejemplo, la búsqueda *titulo* regresa todos los elementos <*titulo*>.
- El operador ("/") indica la jerarquía. La búsqueda *personal/director* regresa todos los elementos <*director*> que son hijos del elementos <*personal*>.
- El elemento raíz de un documento XML puede ser indicado por el operador "/", por ejemplo /listadelibros/libro/titulo.
- El contenido de un elemento o el valor de un atributo se puede especificar usando el operador ("="). La búsqueda reparto/celebridad/actriz='Julia Roberts' regresa todas las actrices con el nombre "Julia Roberts".
- Los nombres de atributos inician con "@". Son tratados como hijos de los elementos a los cuales pertenecen:
- /pelicula/reparto/carácter/actor/@genero='masculino'.
- El operador descendiente ("//") indica cualquier número de niveles intermedios. La siguiente sentencia muestra a los actores en cualquier lugar dentro del reparto: reparto//actor.
- Cuando el operador descendiente es encontrado en el inicio de una ruta, eso significa: todos los nodos descendientes del documento. La siguiente búsqueda encontrará cualquier actor en el documento: //actor.
- El operador filtro("[]") filtra un conjunto de nodos basados en las condiciones indicadas dentro de los corchetes. La siguiente búsqueda regresa a los actores. Cada uno de los actores tiene que tener un atributo llamado "género" con el valor "masculino": //actor[@genero='masculino']. Se pueden combinar múltiples condiciones usando operadores booleanos.

2.1.2 Lorel

Lorel fue diseñado para realizar búsquedas de datos semi-estructurados. Fue desarrollado como un lenguaje de búsqueda para el prototipo del sistema administrador de bases de datos Lore en Stanford. En una búsqueda en Lorel, el constructor aparece en la cláusula SELECT, las normas aparecen en la cláusula FROM, y ambos padrones y filtros aparecen en la cláusula WHERE. En las figuras 2.1, 2.2 y 2.3 se muestran tres diferentes expresiones en Lorel: búsqueda, unión y actualización, respectivamente.[Abiteboul,96]

```
Lorel query
 select C.director, C.writer
 from movielist.movie{M}.crew C
 where M.rating = ”8.0”
```

Figura-2.1 Ejemplo de búsqueda usando Lorel

```
Lorel join
 select C.writer, A.actor, A.role
 from movielist.movie.crew C, movielist.movie.cast.character A
 where C.director = A.actor
```

Figura-2.2 Ejemplo de unión usando Lorel

```
Lorel update
 update C.producer := A.actor
 from movielist.movie.crew C, movielist.movie.cast.character A
 where A.actor = ”Selena”
```

Figura-2.3 Ejemplo de actualización usando Lorel

El sistema Lorel tiene dos interfaces: una interfaz de texto y la interfaz gráfica, que proporciona diversas herramientas para revisar los resultados de las búsquedas.

2.1.3 XQuery

En febrero de 2001, el grupo de trabajo XML Query publicó su primer trabajo: “XQuery: A Query Language for XML”. XQuery probablemente se convertirá en un lenguaje de búsqueda estándar para XML. Este esfuerzo de W3C inició en enero de 2000 con los requerimientos que especifican los objetivos, escenarios de uso, y requerimientos para modelos de datos de búsqueda de XML del W3C, álgebra y lenguajes de búsqueda.[Birbeck,01]

XQuery se deriva de un lenguaje de búsqueda de XML llamado Quilt, el cual a su vez toma características de otros lenguajes. Como XQL, XQuery usa expresiones con sintaxis apropiadas para documentos jerárquicos. Pero la característica interesante de XQuery es el soporte FLWR. Esta característica permite al usuario formular búsquedas conteniendo: FOR, LET, WHILE y RETURN, como se muestra en la Figura-2.4.[Chamberlin,01]

```
expresiones FLWR
          FOR $b IN document("movies.xml")//movie
          WHERE $b/crew/director = "Alfred Hitchcock"
          AND $b/releaseyear = "1959"
          RETURN $b/title
```

Figura-2.4 Ejemplo de una búsqueda usando XQuery

Además, XQuery soporta expresiones incluyendo operadores y funciones (por ejemplo, UNION, INTERSECCION y EXCEPCION). Expresiones condicionales (IF THEN ELSE) que se pueden usar cuando la estructura de la información a ser regresada dependa de alguna condición. Los tipos de datos pueden ser examinados o modificados en expresiones XQuery.

El proyecto XQuery del W3C continúa su estudio y desarrollo, aún no es un estándar aprobado.

2.1.4 Otros lenguajes de búsqueda

Otros lenguajes de búsqueda se describen a continuación:

a) XML-GL. Sus búsquedas son realizadas visualmente usando un formalismo basado en gráficos cerca de la estructura de documentos XML. Sin embargo, XML-GL no es una interfase visual más allá de lo convencional, textual, pero es un lenguaje de búsqueda basado en gráficos con una sintaxis y semántica definidos en términos de estructuras gráficas y operaciones.[Ceri,98]

b) XML-QL. Es un lenguaje de búsqueda para datos XML desarrollado en AT&T. XML-QL proporciona soporte para búsquedas, construcción, transformación e integración de datos XML. En una búsqueda las variables son usadas para transferir datos desde la cláusula where a la cláusula construct.[Deutsch,98]

c) Xtract. Es una herramienta de línea de comandos para búsquedas en documentos XML. La herramienta es usada como el comando de UNIX: matches/match/player, con este comando se seleccionan los elementos player que contengan los elementos match que se encuentran dentro del elemento matches.[Wallace,02]

d) Quilt. Es una simple forma de una búsqueda que consiste de cláusulas FOR, WHERE y RETURN. La cláusula LET puede también usarse para atar una variable al valor de una expresión.[Robie,00]

e) XMAS. Es un lenguaje de búsqueda de XML declarativo basado en reglas. La idea básica es usar reglas XMAS las cuales están de la forma “construct encabezado where cuerpo”. El cuerpo especifica los datos que serán extraídos de las fuentes XML bajo la consideración de que el encabezado describe como serán extraídos los datos que son colocados en un nuevo documento XML de respuesta.[Baru,98]

2.2 Analizadores XML

En los siguientes capítulos, se usará la palabra “analizador” al referirse a un “*parser*”. Un analizador XML es una herramienta que permite comprobar si la estructura de un documento es correcta de acuerdo al estándar XML, es decir, se encarga de verificar errores y que el documento esté correctamente estructurado de acuerdo al DTD al que esté relacionado, se puede incluir en alguna aplicación de forma tal que ésta pueda manipular documentos XML.

Existen diversos analizadores programados en diversos lenguajes y para varias plataformas: C, Java, Phyton, Visual Basic, Perl, Delphi, etc. La utilización de un analizador en especial dependerá de las necesidades de la aplicación a desarrollar, ya que existen analizadores que comprueban que el documento esté bien formado y los que analizan un documento con respecto a un DTD.

Se han visto en el Capítulo 1 los fundamentos de XML, así como la sintaxis básica para la estructuración de un documento XML. Ahora se analizarán las herramientas para llevar a cabo el procesamiento de la información contenida en documentos XML.

Existen dos modelos de procesamiento que pueden usarse para acceder datos XML: SAX y DOM, en la Figura-2.5 se muestra en forma gráfica la relación entre SAX y DOM con documentos XML.

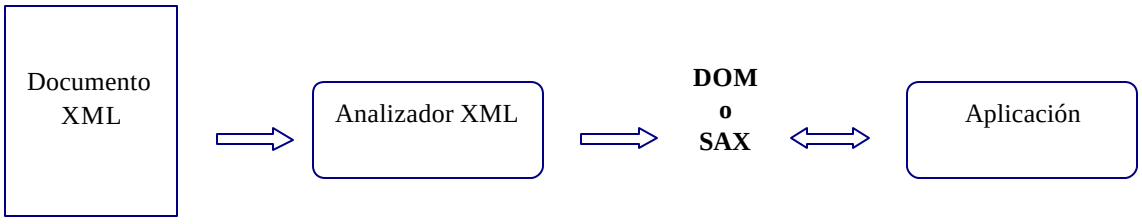

Figura-2.5 Ejemplo de procesamiento de XML

API (*Application Programming Interface*). Interfaz que proporciona llamadas a funciones y procedimientos que proporcionen métodos y mecanismos para manipular, en este caso documentos XML. Son independientes del lenguaje de programación. En la Figura-2.6 se muestra la representación gráfica de SAX y DOM.

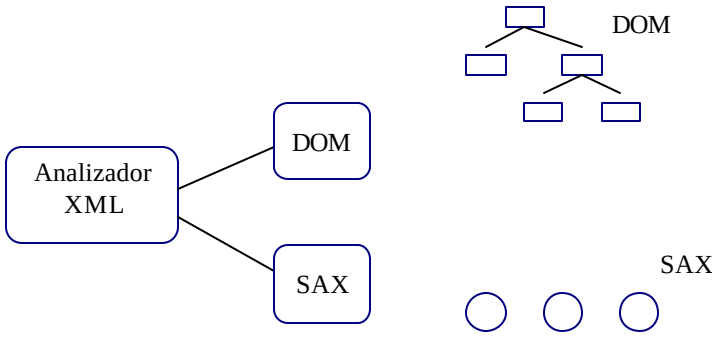

Figura-2.6 Ejemplo de representación gráfica de SAX y DOM

2.2.1 Simple API for XML (SAX)

SAX (*Simple API for XML*) lee los datos XML en forma secuencial identificando cada uno de los elementos que conforman el documento como son las etiquetas de inicio, etiquetas final, datos, etc. Contiene una metodología basada en eventos que pueden ser usados por una aplicación para procesar los datos. Puede manejar documentos XML muy grandes sin requerir grandes cantidades de memoria para almacenar los datos, dando como resultado una mayor rapidez de procesamiento. Este ahorro de memoria se consigue porque SAX tiene en cuenta en cada momento únicamente el punto actual del documento sin considerar el resto.

Una característica de SAX es que identifica la estructura del documento que ha sido especificada a través de un DTD o un esquema.

2.2.2 Document Object Model (DOM)

El API DOM (*Document Object Model*) accede a los elementos de un documento y permite manipular y modificar la estructura de los datos debido a que mantiene en memoria la estructura de árbol del documento.

Como se mantiene y gestiona en memoria en todo momento la estructura de árbol del documento, el consumo de recursos es considerable, sobre todo al manipular documentos muy grandes.

Una de las ventajas de DOM es que se puede modificar, eliminar o incluir elementos a la estructura del documento.

Veamos en forma gráfica en la Figura-2.7 y Figura-2.8 la estructura de árbol de un documento XML.

```
<?xml version=”1.0”>
<articulo>
  <artencabezado>
    <titulo>Material de la primera semana</titulo>
    <autor>Lucia Venta</autor>
    <fecha>17 marzo 2001</fecha>
  </artencabezado>
  <sect1>
    <titulo>Agentes</titulo>
    <parrafo>
    Un agente es un sistema computacional, situado en algún
    ambiente, capaz de actuar autónomamente y dotado de
    flexibilidad para alcanzar sus objetivos
    </parrafo>
  </sect1>
</articulo>
```

Figura-2.7 Ejemplo de un documento XML

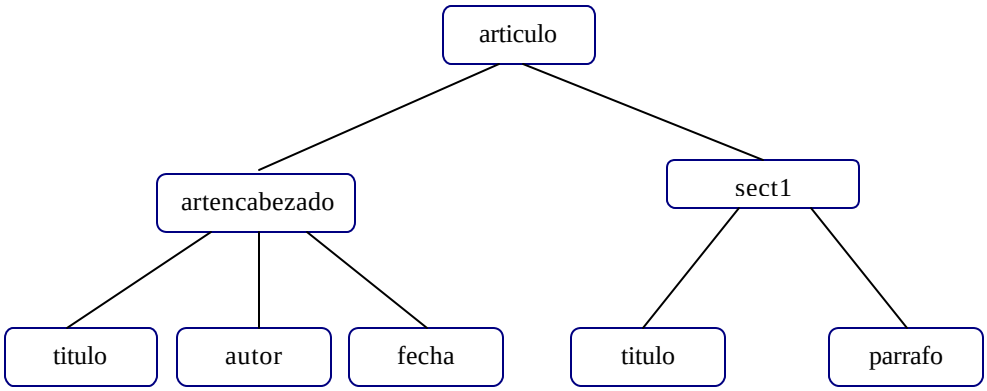

Figura - 2.8 Representación gráfica de una estructura XML

DOM identifica en todo momento el punto actual donde se encuentra cada uno de los nodos que conforman la estructura de árbol del documento. Contiene funciones que permiten desplazarse a través del árbol. Dichas funciones permiten modificar el árbol, es decir, permiten cambiar el contenido de un elemento, mover un elemento de una posición del árbol a otra, eliminar o añadir elementos, etc.

2.3 Tipos de analizadores

Existen dos tipos de analizadores: validadores y no-validadores. Ambos tienen la característica de que deben informar las violaciones de las restricciones de documento bien formado dadas en su especificación. Permiten separar los elementos de su contenido e identificar las diferentes partes del documento. [Birbeck,01]

a) Sin Validación. El analizador sólo se asegura de que los datos estén "bien formados" basándose en la sintaxis de XML, por ejemplo, que exista sólo una etiqueta raíz, que estén bien cerradas, etc.

b) Con Validación. El analizador verifica que el documento se adapte a un esquema específico de un DTD (ya sea interno o externo a el documento XML a ser analizado), además de comprobar que el documento esté bien formado según la sintaxis de XML.

Existen algunas reglas mínimas que un analizador verifica para no rechazar un documento XML:

- Sólo se permite un elemento raíz. Es imprescindible cumplir esta norma para que el analizador pueda saber que el documento esta completo, como se muestra en la Figura-2.9.

```
<?xml version=”1.0”>
<articulo>
    <titulo>
    Documento XML con un elemento raíz
    </titulo>
</articulo>
```

Figura-2.9 Ejemplo de un documento XML con un elemento raíz

- En la Figura-2.10 se muestra un documento con dos elementos raiz.

```
<?xml version=”1.0”>
<articulo>
    <titulo>
    Documento XML con dos elementos raíz
    </titulo>
<articulo>
    <titulo>
     Este es el segundo elemento raíz en un mismo documento
</titulo>
</articulo>
</articulo>
```

Figura-2.10 Ejemplo de un documento XML con dos elementos raíz

- Los elementos deben iniciar con una etiqueta de inicio y terminar con una etiqueta de fin, como se muestra en la Figura-2.11.

```
<?xml version=”1.0”>
<articulo>
    <titulo>
    Las etiquetas están correctamente declaradas
    </titulo>
</articulo>
```

Figura-2.11 Ejemplo de la correcta declaración de etiquetas en un documento XML

- En la Figura-2.12 se muestra un ejemplo de la incorrecta declaración de etiquetas por estar incompletas.

```
<?xml version=”1.0”>
<articulo>
    <titulo>
    no se declaró la etiqueta de cierre del titulo
</articulo>
```

Figura-2.12 Ejemplo de etiquetas incompletas en un documento XML

- Las etiquetas deben estar bien relacionadas, es decir colocar en orden las etiquetas de inicio y de fin, veamos un ejemplo en la Figura-2.13.

```
<?xml version=”1.0”>
<articulo>
    <titulo>
    Las etiquetas no están bien relacionadas
    <autor>
    </titulo>
    Luis González
    </autor>
</articulo>
```

Figura-2.13 Ejemplo de etiquetas anidadas incorrectamente

? XML distingue entre mayúsculas y minúsculas, como se muestra en la Figura-2.14.

```
<?xml version=”1.0”>
<articulo>
    <titulo>
    Las etiquetas son diferentes al estar en mayúsculas y
    minúsculas
    </TITULO>
    <AUTOR>
    Luis González
    </autor>
</ARTICULO>
```

Figura-2.14 Ejemplo de etiquetas incorrectas al mezclar mayúsculas y minúsculas

Si un documento cumple con las reglas anteriores (existen otras reglas que conforman la sintaxis básica de XML) se considera que el documento está bien formado. Si el documento se basa en un DTD y cumple con sus restricciones se considera como un documento XML válido.

Hay varios analizadores XML disponibles, algunos de ellos se describen a continuación:

a) XERCES. Es un analizador que soporta SAX y DOM, fue liberado bajo la licencia de código abierto, está compilado sólo para Linux, pero se dispone de los archivos para usar Visual C++ o Borland C++ para Windows .[Apache,00]

b) Oracle XML Parser. Oracle liberó su analizador XML para Java, un componente XML que permite analizar documentos XML a través de las interfaces DOM y SAX usando los modos de validación y no validación.[Oracle,01]

c) JAXP (Java API for XML Processing). Desarrollada por Sun Microsystems, soporta SAX y DOM. Permite a las aplicaciones analizar y transformar documento XML usando un API independiente.[Sun,01]

d) XML4J (XML Parser for Java). Es un analizador XML escrito completamente en Java, es una librería para analizar y generar documentos XML. Este analizador le da a una aplicación la habilidad de leer y escribir datos XML. Un simple archivo JAR proporciona las clases para analizar, generar, manipular y validar documentos XML. Las características de generación y validación le permiten al analizador XML4J ser usado para construir servidores Web XML, validación en tiempo de ejecución usado en los editores XML.[IBM,00]

Como se puede observar con los ejemplos anteriores los analizadores tienen características en común. Por ejemplo, se espera que puedan identificar los elementos de un documento estructurado, los atributos que contengan dichos elementos, el texto de cada elemento, la identificación de las instrucciones de proceso propios de XML así como los comentarios incluidos en el documento.

2.4 XML y las búsquedas

La característica principal de XML es que se puede usar para representar datos estructurados (registros) así como datos no estructurados (texto). Por ejemplo, XML se puede usar para representar (estructurar) información de un documento de texto de una materia de la Universidad Virtual de la Universidad Tecnológica de la Mixteca, identificando el título del documento, autor, párrafos, listas de elementos, imágenes, etc. Para aprovechar esta característica, sin embargo, es importante tener herramientas que puedan trabajar adecuadamente con ambos tipos de datos. Es muy importante tener un lenguaje de búsqueda el cual pueda seleccionar los registros de la parte estructurada de un documento XML y buscar información en el texto. Por ejemplo, debería de ser posible plantear una búsqueda que encontrara todos los párrafos que incluyeran la palabra “objetos”.

Es muy importante la búsqueda de datos en documentos XML, ya que puede explotarse la característica de que los documentos XML cuentan con una estructura, un ejemplo de un documento estructurado puede verse en la Figura-2.15. Veamos una situación que se presenta frecuentemente cuando se tienen varios documentos con grandes cantidades de información en cada uno y se desea encontrar una palabra específica. Si los documentos sobre los cuales se desea realizar la consulta no cuentan con una estructura, los resultados de la búsqueda incluirían toda palabra que coincidiera con la búsqueda realizada, dándonos en muchos casos información que no es relevante, como en el caso de una búsqueda en Internet (por ejemplo al utilizar algún buscador en Internet). Pero si dichos documentos están estructurados de tal forma que se identifica cada uno de sus elementos, se puede especificar sobre cuál elemento en especial se desea realizar la consulta, eliminando aquellos elementos de los que no se desea ninguna información. Como se puede observar en la Figura.2-15, donde se ha identificado cada una de las partes del documento con una etiqueta, de esta forma si se desea encontrar alguna palabra contenida en el título del documento, la búsqueda se realizaría únicamente en los elementos identificados con la etiqueta <titulo>, sin incluir el resto del contenido del documento.

```
<?xml version=”1.0”>
<articulo>
      <titulo>Tecnología de objetos</titulo>
      <autor>Oscar Atriano</autor>
      <sect1>
         <titulo>Primera parte</titulo>
         <parrafo>
        La programación orientada a objetos (POO) encapsula datos
        (atributos) y métodos (comportamientos) en objetos. Los
        datos y métodos están íntimamente relacionados entre sí. Los
        objetos tienen la propiedad de ocultamiento de información:
        los objetos saben cómo comunicarse entre sí a través de
        interfaces bien definidas, normalmente no se permite que un
        objeto sepa cómo están implementados otros objetos; los
        detalles de la implementación están ocultos dentro de los
        objetos mismos.
        </parrafo>
     </sect1>
</articulo>
```

Figura-2.15 Ejemplo de un documento estructurado

2.4.1 Consulta de documentos

La consulta de documentos XML representa un nuevo cambio, dada la perspectiva de los grandes volúmenes de datos que están disponibles, generando una necesidad de herramientas de consulta que sean flexibles y capaces de comprender la heterogeneidad de los documentos y capaces de extraer información con rapidez y exactitud.

Un lenguaje de consulta de documentos XML puede o no producir como resultado un documento XML, o bien las aplicaciones pueden trabajar con los datos originales o con resultados de consultas anteriores, dando mayor flexibilidad a las consultas.

Un lenguaje de consulta debería ofrecer ciertas características, por ejemplo que permita extraer elementos específicos de un documento o ignorar aquellos elementos que no cumplan con los requerimientos solicitados.

Existen diversos programas que ofrecen la posibilidad de realizar consultas sobre documentos estructurados, a continuación se presentan algunos:

a) X-Cubed Search Engine. Es un sistema de búsqueda que indexa documentos XML de acuerdo a las etiquetas de cada documento. Requiere de un servidor MS-SQL en Windows 2000 con IIS instalado. [X3,02]

b) GoXML Search. Es una aplicación servidor que indexa documentos XML y proporciona interfaces de búsqueda estándar.[GoXML,02]

c) SAIC SIM Server. Contiene una base de datos XML propia, fue diseñado para entender la estructura jerárquica de los documentos. [Saic,02]

d) XYZFind. Es una base de datos XML, a diferencia de las bases de datos tradicionales que descomponen y almacenan XML en tablas, XYZFind descompone un documento XML en una simple representación de datos. Facilita el almacenamiento de los documentos.[XYZFind,02]

2.4.2 Procesadores de búsquedas

Un procesador de búsqueda de XML usando un lenguaje de búsqueda XML permite realizar consultas en los elementos y atributos que componen un documento XML. Hay varios procesadores de búsqueda de XML implementados en Java, algunos de ellos son:

a) Kweelt. Es un motor de búsqueda de datos XML, que entre otras cosas ofrece un motor de evaluación para el lenguaje de búsqueda XML Quilt. Kweel ha hecho una adaptación personal del Quilt de tal forma que algunas características de éste han sido eliminadas, algunas otras han sido adaptadas, y se han agregado algunas más. La finalidad de Kweelt es proporcionar un esquema extendido de búsqueda de XML. Se ha separado completamente el motor de búsqueda del módulo de almacenamiento. El primero es responsable de analizar y evaluar las búsquedas, el segundo es el encargado de la manipulación del XML. El enlace entre los dos es a través de varios APIs.[Kweelt,01]

b) XML Query Engine (XQEngine). Es un motor de búsqueda de texto completo para XML. Permite realizar búsquedas en documentos XML, usando el lenguaje de búsquedas XQL. XML Query Engine puede indexar múltiples documentos usando cualquier analizador SAX disponible. El índice, una vez construido, se puede accesar usando XQL. Las búsquedas expresadas en XQL son más expresivas y poderosas que las interfaces de búsqueda estándar disponibles a través de los motores de búsquedas basados en el Web. XML Query Engine es un componente Java que permite indexar y después realizar la búsqueda en documentos XML por elemento, atributo o búsqueda de texto completo. Tiene una interfaz de programación que le permite ser llamado fácilmente desde la propia aplicación Java. El esquema puede trabajar como una herramienta personal de productividad en una computadora personal o en un servidor con un bajo a medio nivel de acceso. Los documentos a ser buscados se limitan a 32,768, cada uno de los cuales puede contener hasta 32,768 elementos.[Howart,02]

c) Xset es un motor de búsqueda XML. La meta fundamental es llevar la funcionalidad de búsqueda XML a todas las aplicaciones con un tiempo mínimo de respuesta. Xset tiene también una base de datos en la memoria principal, que crea un índice en memoria de un conjunto de datos. El índice es una combinación de los directorios del sistema de archivos y de las secuencias de entrada XML del usuario. En una operación normal como un servicio RMI (*Remote Method Invocation*), el XSetService puede ser inicializado en segundo plano (como una tarea secundaria) y las operaciones ser ejecutadas por medio de unas llamadas de los clientes al Java RMI. [Zhao,00]

d) GMD-IPSI XQL Engine. Es una aplicación basada en Java para el almacenamiento y búsqueda de grandes documentos XML. La funcionalidad puede ser accedida por medio de la invocación en línea de comando o un API de Java. El motor consiste en dos partes principales: una implementación persistente del W3C-DOM y una completa implementación del lenguaje XQL. El motor XQL implementa la sintaxis XQL W3C-QL 98. Usa un algoritmo de indexación XML, el cual indexa el documento mientras se procesa la primera búsqueda. Las siguientes búsquedas al mismo documento son aceleradas considerablemente. Implementa la interfaz W3C-DOM al indexar archivos XML binarios. Los documentos son analizados una vez y almacenados en esta forma. Una arquitectura caché adicional incrementa el rendimiento. Durante este tiempo el acceso es sólo de lectura. El formato de salida siempre es en XML. GMD-IPSI XQL Engine, anteriormente PDOM, es una implementación persistente de Java DOM para el almacenamiento de documentos XML de tamaño arbitrario en una forma que estén listos para el desarrollo, búsqueda y extracción. [GMD,01]

e) Niagara. Está diseñado para permitir a los usuarios plantear búsquedas XML sobre Internet. Se puede usar para recuperar datos XML, búsquedas y monitorización de éstos. Estas tres funcionalidades son implementadas por tres principales componentes: Search Engine, Query Engine y Trigger Manager. Niagara usa una novedosa colaboración entre su Search Engine XML y su Query Engine XML. La búsqueda es expresada en el lenguaje de

búsqueda Search Engine Query Language (SEQL). El sistema Niagara tiene dos principales componentes el Search Engine, el cual regresa los enlaces (URLs) de los archivos XML relacionados a la búsqueda, y el Query Engine, el cual evalúa las búsquedas XML-QL. Niagara Query Engine ha sido diseñado desde su inicio con la finalidad de ocuparse de fuentes de datos remotos al mismo tiempo que proporciona a los usuarios la flexibilidad de ver resultados parciales en cualquier momento. [Niagara,01]

Como se puede observar en las descripciones anteriores estas herramientas brindan la posibilidad de realizar búsquedas de texto sobre documentos estructurados, pero dadas las características de XML Query Engine y GMD-IPSI XQL, por ejemplo, estar basados en Java, permitir la manipulación de un gran número de documentos y elementos, la utilización de SAX y DOM, se consideran como las mejores opciones para realizar pruebas en el desarrollo de un sistema de consulta.

2.4.3 Tipos de aplicaciones en Java

Hay diferentes tipos de software que se pueden escribir en Java para usar XML. Existen tres categorías principales, para describir ciertos tipos de aplicaciones (actualmente son las más populares) las cuales son realmente las más convenientes para el uso de XML. Esto no significa que sea un único conjunto de categorías; se pueden crear las propias, y pueden surgir mucho mejores categorías dado que XML se vuelve cada vez más popular. [Idris,99]

Estas categorías son:

? Aplicaciones gráficas Java, del lado del cliente.
? Aplicaciones servidor, del lado del cliente y servidor.
? Aplicaciones basadas en el Web.

En el caso de SICIDEX, pertenece a la primera categoría, es una aplicación gráfica del lado del cliente. Este tipo de aplicaciones XML Java se ilustra en la Figura-2.16

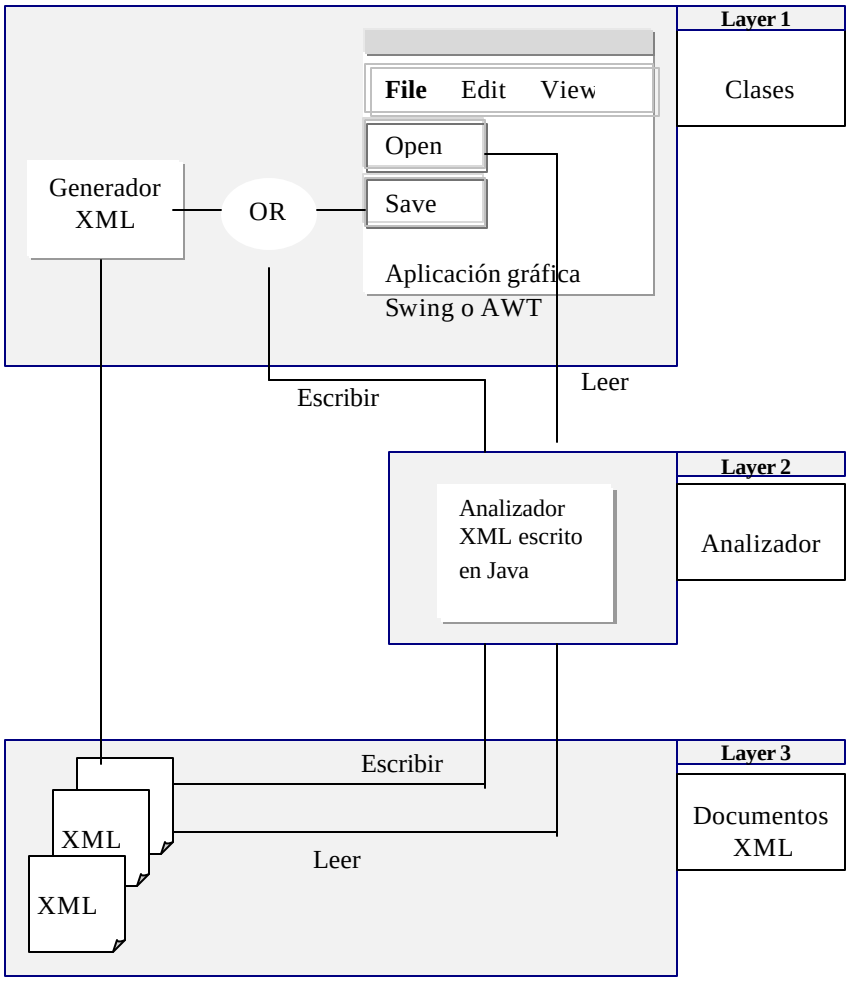

Figura-2.16 Representación de una aplicación gráfica Java [Idris,99]

En las aplicaciones que pertenecen a esta categoría, se tienen que escribir clases que importen y exporten información desde los documentos XML (usando la validación con un DTD si es que se tiene alguno). Se requieren además clases que creen la interfaz de usuario de la aplicación.

Las clases que importen y exporten información desde la aplicación tienen que usar un analizador y el API DOM o SAX. Estas clases pueden acceder esta información usando alguna de las siguientes estrategias:

- Usar DOM para manipular directamente la información almacenada en el documento (DOM crea un árbol de nodos). Este documento es creado por el analizador DOM XML después de ser leído desde documento XML.
- Crear un modelo propio de objetos Java que importen información desde el documento XML usando SAX o DOM. Este tipo de modelo sólo usa SAX o DOM para inicializarse a sí mismo con la información contenida en el o los documentos XML. Una vez que se ha inicializado y realizado el análisis, DOM o SAX no se vuelve a usar. Se puede usar un modelo propio para acceder o modificar información sin usar SAX o DOM. Así que se puede manipular la información usando objetos propios, y confiar en el API SAX o DOM para importar la información desde la aplicación en memoria (como una agrupación de objetos Java).
- Crear un modelo propio de objetos Java (adaptar) que use DOM para manipular la información en el árbol de objetos del documento (que es creado por el analizador). Este modelo difiere ligeramente de la segunda opción, ya que se está usando el API DOM para manipular la información del documento como un árbol de nodos, pero sólo se está envolviendo la aplicación con un API específico alrededor de los objetos DOM.

Dependiendo de cuál de las tres opciones se elija para acceder a la información, esta información es almacenada en memoria, así como el árbol de nodos (DOM), o en el propio modelo de objetos que se haya creado.

Existen ventajas y desventajas en el uso de alguna de las estrategias para importar y exportar XML. La complejidad de la aplicación y los recursos del sistema de los que se disponga son factores que determinarían que estrategia debería usarse.

Al utilizar XML para crear un archivo de formato propio para la información, no se tiene que usar un formato propietario.

Usar XML sobre un formato propietario le permite a la aplicación tener una inmensa interoperabilidad a través de plataformas, aplicaciones y lenguajes de programación.

Para crear aplicaciones gráficas Java del lado del cliente, se puede definir un DTD para la información, como se ha hecho en el caso del sistema de consulta aquí descrito. El usuario de la aplicación puede ver información usando una interfaz gráfica y puede leer información desde archivos XML.

2.4.4 Java y SAX

Una de las características principales del API SAX es que extrae la información de la aplicación según procesa el archivo XML, sin crear ninguna estructura adicional en memoria más que la estrictamente necesaria para la lectura del archivo. SAX trabaja respondiendo a eventos, eventos que se producen al procesar el archivo XML, como puede ser el fin o principio del documento, el fin o principio de un elemento, etc. Por ello es una excelente opción para realizar el mapeo entre documentos XML y Java.

El SICIDEX se planteó como una opción de fácil uso, la utilización de SAX en el sistema garantizaba una mejor administración de los elementos que constituyen un documento estructurado de XML.

SAX no permite agregar o modificar el contenido de los documentos (opciones de DOM), característica que no está contemplada en el diseño del sistema ya que los documentos serán sólo de consulta, el contenido de dichos documentos es responsabilidad del autor.

En el SICIDEX se ha utilizado el paquete org.xml.sax, el cual define todas las interfaces que usará el analizador SAX.

Capítulo 3

IMPLEMENTACIÓN DEL SISTEMA

3.1 Arquitectura del sistema

El sistema SICIDEX es una herramienta que permite realizar consultas sobre documentos estructurados de XML, lo integran diversas partes como son: un área de recepción de información en la cual se ingresan los documentos sobre los cuales se desea realizar las consultas, el elemento que se desea consultar así como la palabra a buscar. Una vez que se han ingresado los datos de entrada, se realiza el procesamiento de la información donde los documentos son indexados y procesados de tal forma que los elementos sean identificados con su interdependencia entre elementos. Para la presentación de resultados se pueden identificar cada uno de los resultados y dentro de todo el contenido de los documentos ubicar el cursor en el lugar donde se encuentra el elemento que se desea consultar, se pueden realizar diversas consultas sobre diferentes elementos.

A continuación se presentan los diagramas de clases y actividades del sistema, así como los diagramas que presentarán una vista funcional del mismo.

En el Diagrama-3.1 se muestra el Diagrama de clases de SICIDEX, mostrando un diagrama estático, es decir un diagrama de clases que muestra la relación entre ellas, en el cual se puede observar como la clase BusquedaTexto tiene características similares de la clase JFrame, así como el Refinamiento con respecto a ChangedListener y ResultListener; se muestran las instancias como son BackKey, CheckNumber, InstanceTree, NumberChekIntro, algunas de las cuales comparten la característica de la agregación con respecto a su clase de donde fue heredada y la composición con la clase BusquedaTexto.

La clase BusquedaTexto contiene diversas funciones que permiten obtener los datos de entrada como son las etiquetas sobre las cuales se realizarán las consultas, la palabra que se desea consultar así como si se desea incluir en la consulta las etiquetas que se encuentren dentro de la etiqueta principal.

Las clases BackKey, CheckNumber y NumberChekIntro analizan los datos y verifican si el formato de entrada es el adecuado, en caso de que no sea el correcto se solicita otra vez la introducción de la información y nuevamente es verificada.

Las clases FucTextHandler, funcTextHandler_all y FuncTwoTextHandler_all analizan los documentos, mediante la identificación del inicio y final de un documento, inicio y final de una etiqueta, inicio y final de texto, hasta encontrar las coincidencias con las etiquetas y la palabra a consultar.

La clase InstanceTree crea el árbol de etiquetas de cada documento para poder identificar la jerarquía de las mismas.

La clase Ayuda despliega los tópicos de ayuda para el uso del sistema.

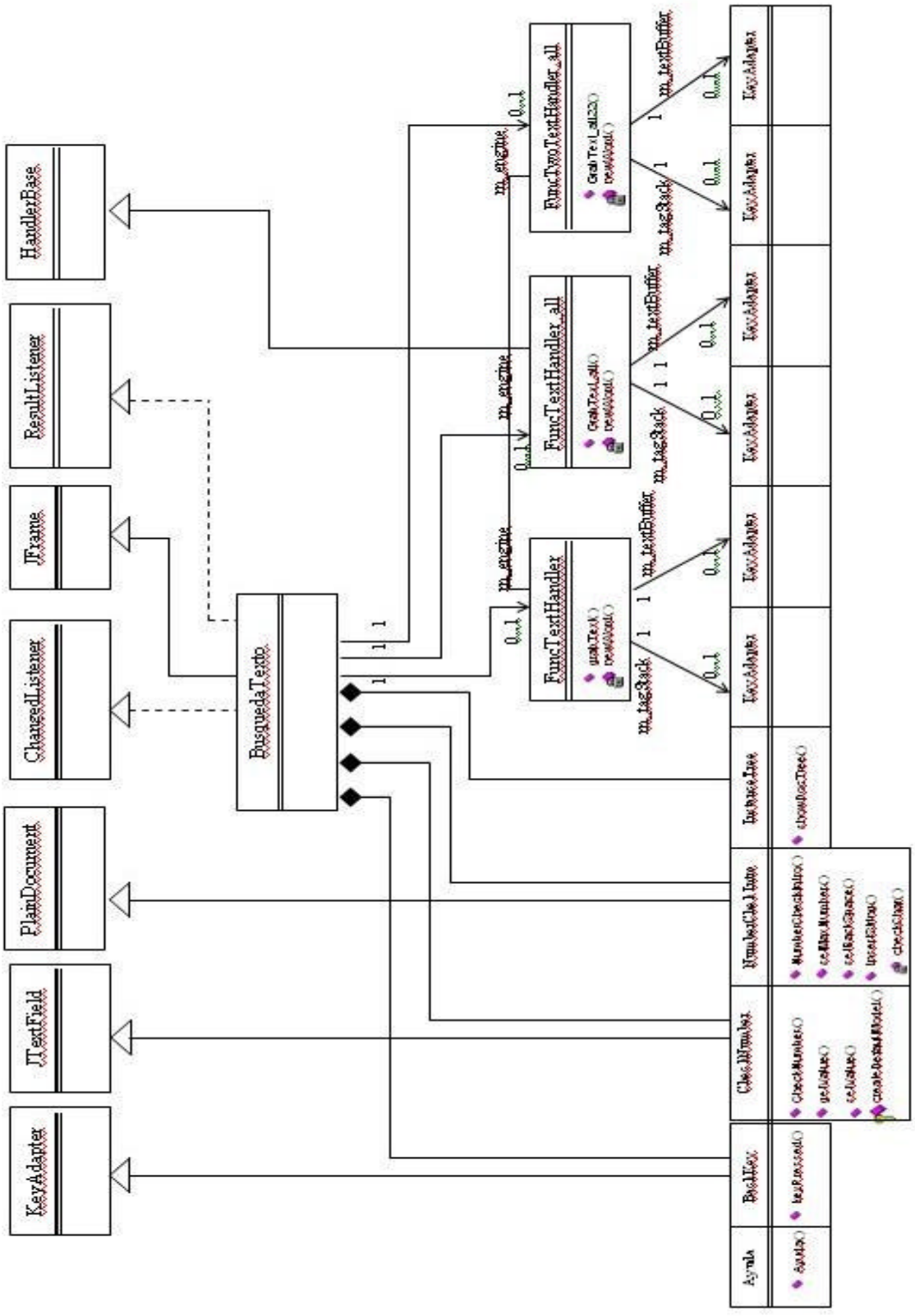

Diagrama - 3.1 Diagrama de clases del sistema

El Diagrama-3.2 y Diagrama-3.3 muestran el Diagrama de Actividades, el cual describe el comportamiento de los procesos entre sí, incluyendo las actividades en paralelo como es el caso de la presentación de las ventanas tanto de recepción de datos como de presentación de los resultados.

El diagrama de actividades indica además los procesos que dependen de la respuesta del usuario, es decir, cuando se decide que acción tomar dependiendo de la información recibida.

Como se puede observar se indican los procesos desde la selección de los documentos sobre los cuales se quiere realizar la consulta hasta la presentación de los resultados.

Se pueden identificar procesos tales como la creación de las ventanas de recepción de datos en la cual se solicitará se indique la etiqueta sobre la cual se desea realizar una consulta, así como si se incluirán en la consulta todas las etiquetas que formen parte de la etiqueta principal, por ejemplo, si la etiqueta <parrafo> incluyera etiquetas tales como <definicion> y <ejemplo> la consulta se podría realizar en las tres.

Los documentos que se van a consultar son indexados de tal forma que se identifiquen todos los elementos que los componen. Creando así una lista de todas las etiquetas identificadas.

Una vez que se introduce la palabra a consultar y que se ha verificado que es un texto se construye una sentencia de XQL.

Cuando ya se han encontrado los elementos que coinciden con la consulta, éstos son clasificados para presentarlos en la ventana de resultados, es decir se identifica a cada uno de ellos con un número y se especifica el documento en el cual fueron encontrados.

Si se desea ubicar alguno de los resultados dentro del o de los documentos consultados, se especifica el número que lo identifica o bien se navega entre todas las coincidencias y automáticamente el cursor se coloca en el

lugar en el que se encuentra el elemento dentro de todo el documento, resaltando el texto en un color diferente para su rápida identificación.

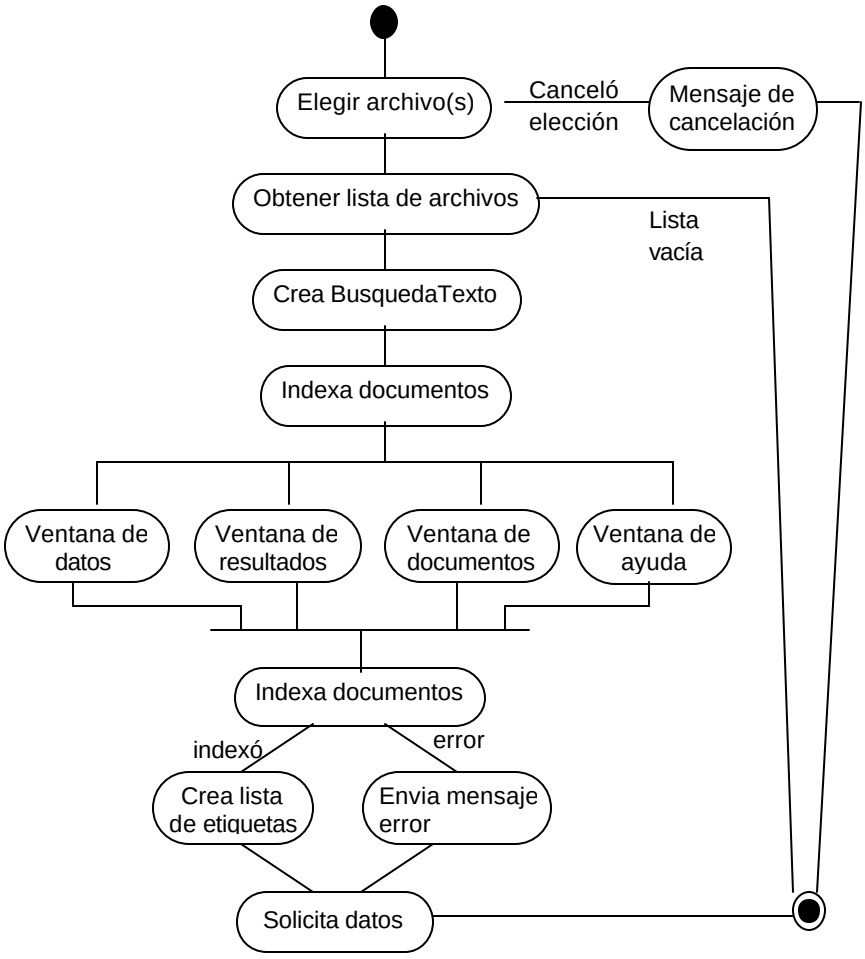

Diagrama - 3.2 Diagrama de actividades

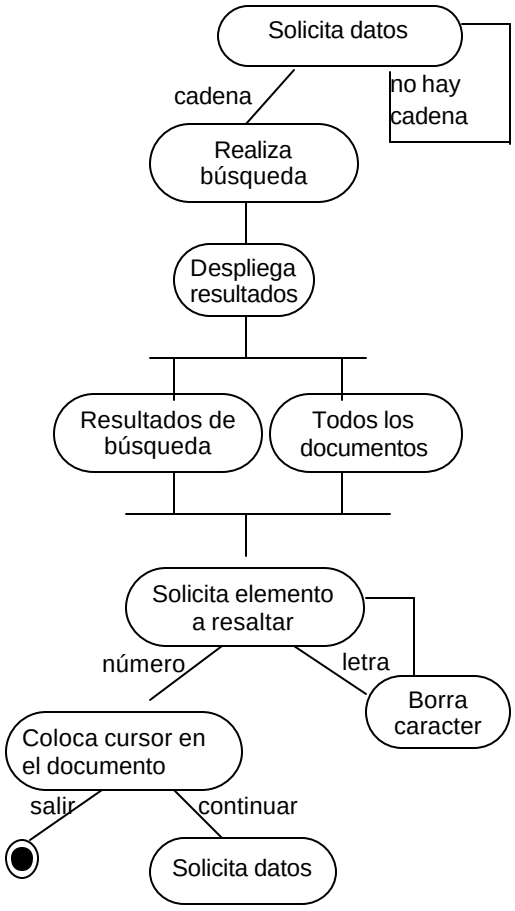

Diagrama-3.3 Diagrama de actividades

A continuación se presentan los diagramas de estado del SICIDEX, los cuales nos muestran una vista funcional del mismo.

En el Diagrama-3.4 se describe lo siguiente:

- Se establece una lista de elementos únicos identificados en la estructura del documento XML sobre los cuales se pueda realizar la consulta.
- Se puede consultar no solamente en un documento sino en varios documentos.

- No sólo se identifica la palabra a consultar en el documento original si no también su ubicación dentro del documento de cada una de las instancias encontradas.
- Pueden reconocerse los elementos de la estructura de cada uno de los documentos.
- El cursor se coloca en el lugar donde se encuentra la palabra consultada dentro de todos los documentos.
- Se identifica el elemento donde se encuentra la palabra consultada dentro de todos los documentos con un color diferente.

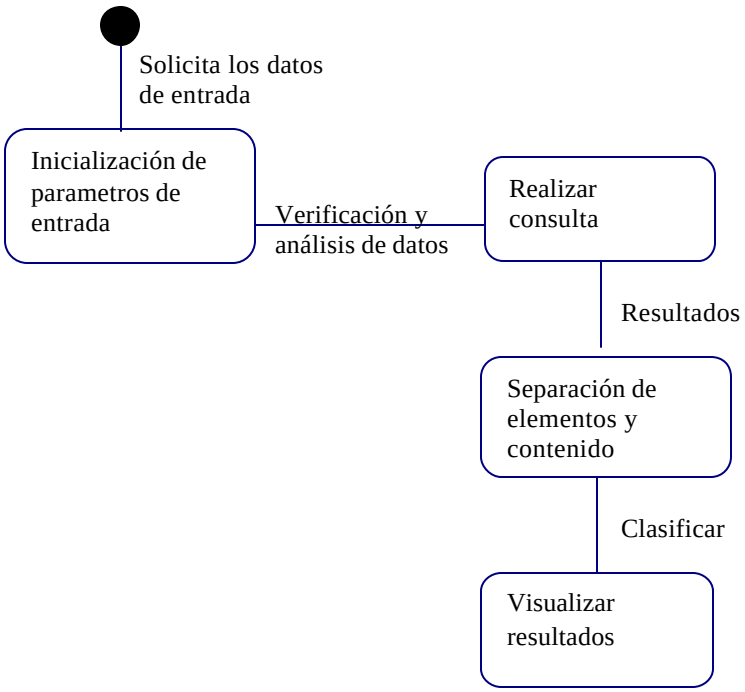

Diagrama-3.4 SICIDEX

En el Diagrama-3.5 se muestran los estados donde una vez que se ha establecido la lista de documentos estructurados sobre los cuales se desea realizar la consulta, éstos deben ser indexados utilizando un analizador.

El procesador de búsquedas XML Query Engine, recibe como parámetro de entrada una sentencia XQL, la cual procesa generando así un resultado. Dicho resultado tiene que ser procesado según los requerimientos del sistema.

Al leer la palabra a consultar se valida, verificando que sea una palabra y se eliminan los espacios en blanco. Se lee el elemento sobre el cual se realizará la consulta.

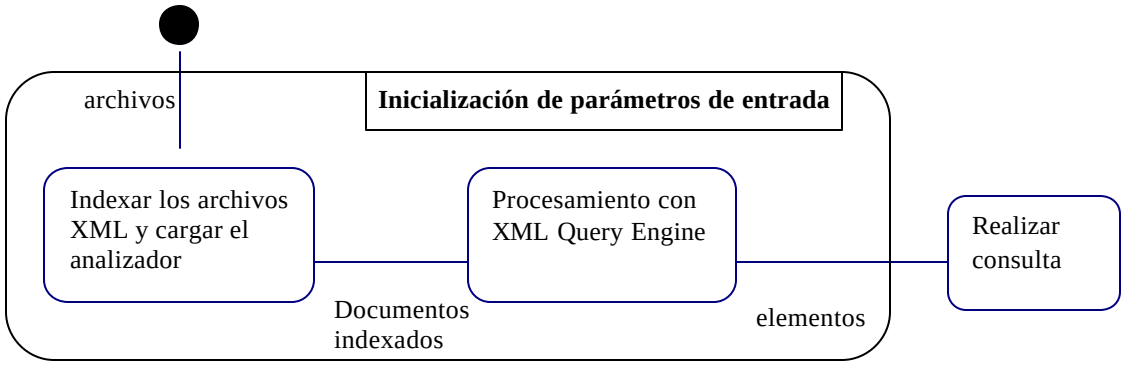

Diagrama-3.5 Inicialización de parámetros de entrada

La palabra a consultar verificada junto con el elemento sobre el cual se realizará dicha consulta, son enviados al XMLQuery Engine, el cual genera un resultado dependiendo del tipo de consulta que se haya especificado.

Los resultados obtenidos deberán ser identificados entre elementos, atributos y contenido, de tal forma que sean útiles para cumplir con los requerimientos del sistema.

En el Diagrama-3.6 muestra el comportamiento del sistema ante los resultados de la consulta, los cuales serán desplegados en forma gráfica, permitiendo identificar cada uno de los resultados mediante un número consecutivo el cual servirá para elegir un elemento en especial dentro de todos los documentos consultados.

El contenido de cada uno de los documentos será desplegado en forma consecutiva, identificando cada uno de los documentos y su contenido, de esta manera si el usuario desea encontrar el resultado de una consulta dentro de dichos documentos, el elemento elegido se visualizará en un color diferente al del resto del texto y el cursor se colocará en la ubicación del mismo.

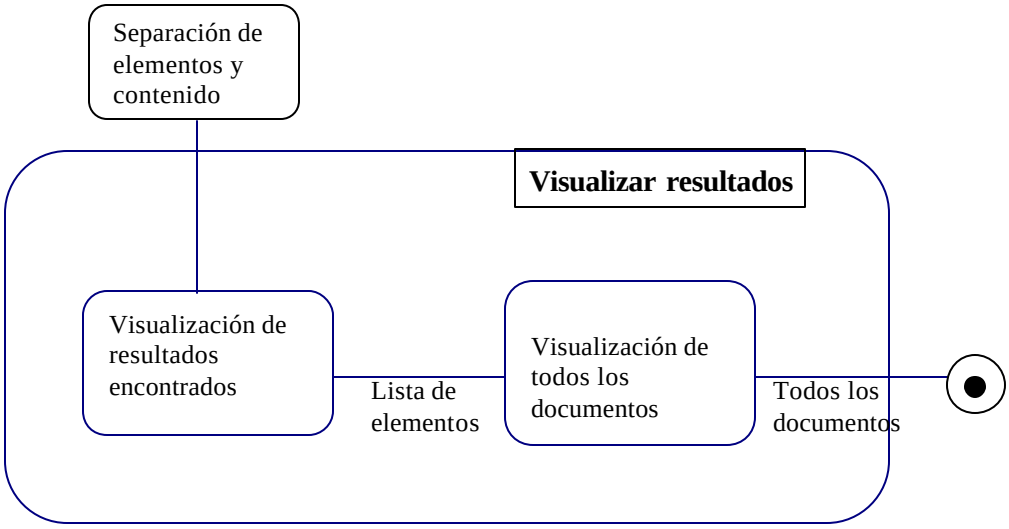

Diagrama - 3.6 Visualización de resultados encontrados en los documentos consultados

3.2 Características finales del sistema

El sistema SICIDEX tiene las siguientes características:

- Consulta de datos en documentos XML: el propósito principal del sistema es trabajar como un formulador de búsquedas sobre un conjunto de documentos XML.
- Interfaz gráfica.

- No es necesario conocer la forma en que está estructurado un documento o cuál es su DTD para poder utilizar el sistema.
- El número de documentos XML a consultar no es fijo.
- El programa trabaja en los sistemas operativos Windows y Linux.
- Actualización del sistema: el código se puede volver a usar en futuros desarrollos.

En la Figura-3.1 se muestra el flujo de la información cuando un usuario interactúa con el "Sistema de consulta de información en documentos estructurados de XML".

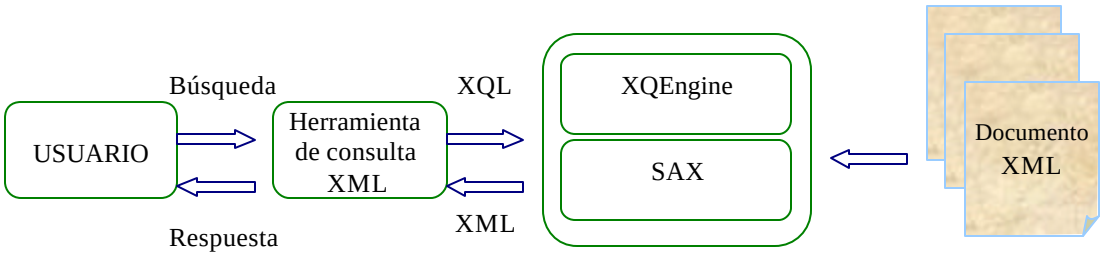

Figura-3.1 Diagrama de flujo de la información en el sistema de consulta

3.3 Ambiente de Desarrollo

La computadora que se usó es una Pentium III a 550 Mhz con 192Mb RAM y Linux Red Hat 6.4 como Sistema Operativo. Algunas pruebas de ejecución del sistema se han llevado a cabo en una computadora con procesador AMD-K6 a 133Mhz con 32Mb RAM y Microsoft Windows 98.

XML y Java se complementan. La tecnología Java proporciona la portabilidad, el reuso de código para procesos portables y el reuso de datos XML. XML y Java tienen un número de características compartidas

que hacen de estos el par ideal para programar en el Web, la independencia de plataforma, expandibilidad y rehusabilidad.

La implementación del sistema se basa en dos librerías GUI de Java muy importantes: AWT (Abstract Windowing Toolkit) y Swing. Se ha usado Java Development Kit (JDK 1.3) para desarrollar la interfaz gráfica del sistema.

3.4 Implementación de Interfaz *DocumentHandler*

La Interfaz *ContentHandler* es el que define todos los eventos que se producen a lo largo del procesamiento de los documentos, así que las clases del proceso de XML deberán implementar esta interfaz para que el analizador haga lo que requiera la aplicación. Esta interfaz contiene un conjunto de métodos extendibles o sobreescribibles según las necesidades.

El API SAX se basa en eventos, eventos que se producen al procesar un archivo, como por ejemplo, inicio/fin del documento, inicio/fin de un elemento, datos entre etiquetas, etc.

Los métodos que invoca SAX en respuesta a diferentes eventos de análisis entre ellos y que se han configurado para este proyecto son: startDocument(), endDocument(), startElement(), endElement(), y characters().

Con el método startElement() se detecta cuando empieza una nueva etiqueta.

El método endElement(), detecta el final de los elementos y de esta manera se pueda obtener su contenido.

El método characters() obtiene los valores contenidos entre las diferentes etiquetas de inicio y de fin de un elemento.

Cada uno de estos métodos debe lanzar una SAXException. Esta excepción es devuelta al analizador, quien la reenvía al código que llamó al analizador.

Cuando se encuentra una etiqueta de inicio o de final, el nombre de la etiqueta se pasa como una cadena los métodos startElement() o endElement(), según corresponda. Cuando se encuentra una etiqueta de inicio, así como cualquier atributo que defina, son pasados en un AttributeList. Los carácteres encontrados dentro del elemento se envían como un arreglo de carácteres, junto con el número de carácteres (longitud) y un desplazamiento dentro del arreglo que apunta al primer carácter.

De esta manera se recorre por completo el o los documentos consultados, identificando sus elementos, atributos y contenido.

3.5 Búsquedas en documentos HTML y XML

A continuación se presenta una búsqueda en un documento tanto en formato HTML como en formato XML, que nos permitirá comparar los resultados en uno u otro formato.

Iniciaremos con una búsqueda en el documento objetos.html que esta en formato HTML, objetos.html, que contiene información de programación orientada a objetos, en el cual se busca la definición de objetos.

Para realizar esta búsqueda se utilizó un CGI, que simula el comportamiento de un buscador común en Internet.

Como se puede observar en la Figura-3.2, se introduce la palabra a buscar, en el caso de este CGI se ha especificado el directorio donde se encuentran varios documentos en formato HTML incluyendo el documento objetos.html sobre el cual se quiere realizar la consulta.

Figura-3.2 Búsqueda en documento HTML

Una vez que se ha especificado la palabra objetos que es la que se desea buscar, el buscador muestra una lista de los documentos en los cuales se ha encontrado dicha palabra. Los documentos son ordenados por el documento con mayor número de coincidencias con la búsqueda. En la Figura-3.3 se muestra la pantalla de resultados. Como se puede observar el buscador ha incluido los archivos software.html y Modelado.html y los presenta como posibles soluciones.

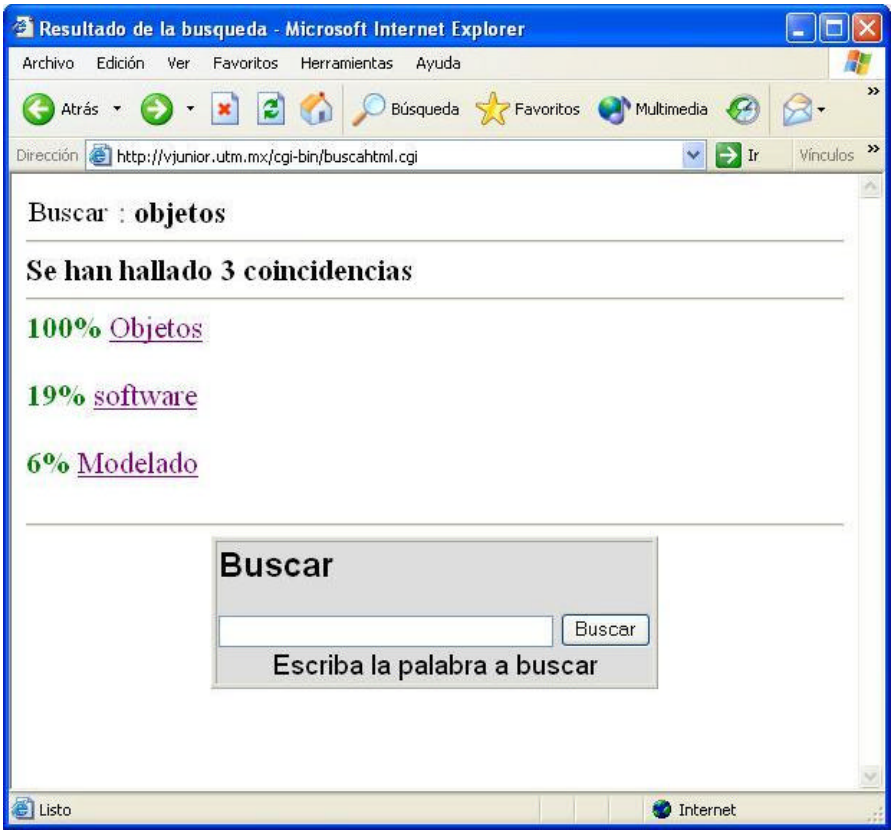

Figura-3.3 Resultados de la búsqueda en HTML

El resultado nos muestra una lista de enlaces de todo aquel documento dentro del cual se encuentre la palabra que se busca. En este caso, la primera opción muestra al documento objetos.html el cual obviamente contiene el mayor número de palabras que coinciden con la palabra buscada.

Al acceder el enlace objetos.html se mostrará el contenido del documento, como se ilustra en la Figura-3.4.

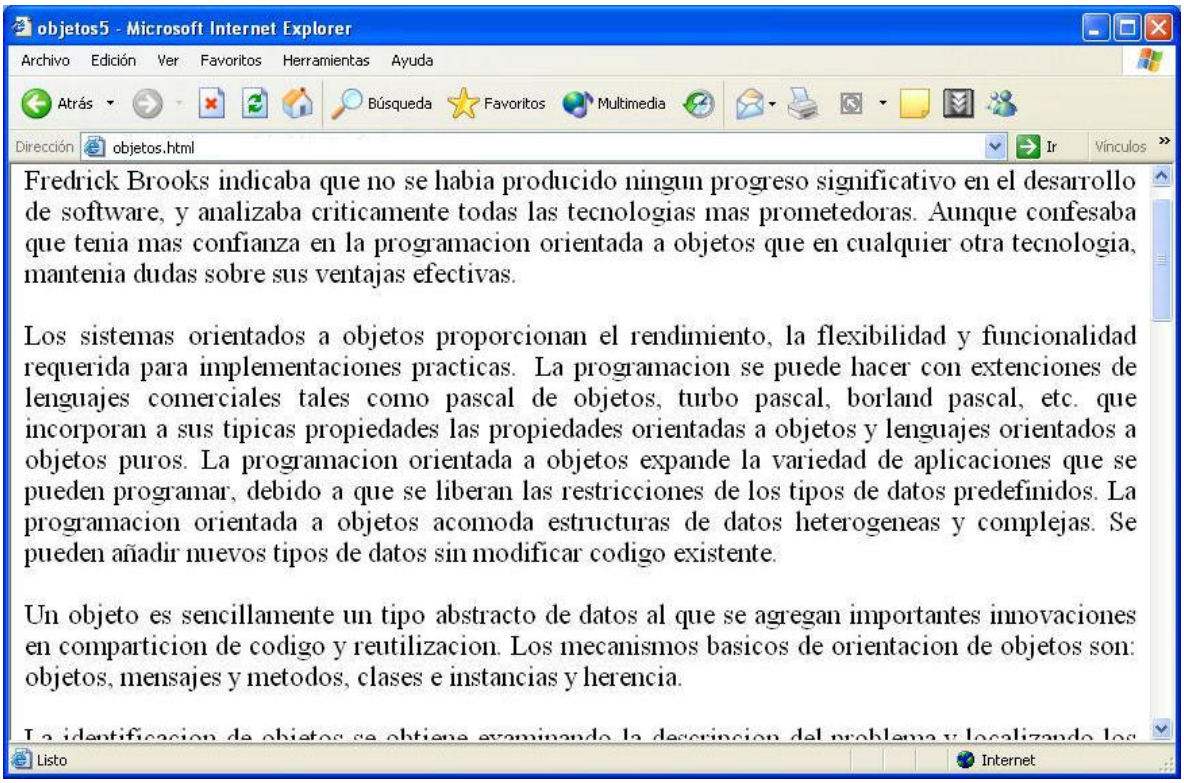

Figura-3.4 Visualización del documento HTML consultado

Como se puede observar en la Figura-3.4 el documento es desplegado en su totalidad y le corresponde al usuario buscar la definición de objetos dentro del documento, es decir es necesario leer línea por línea hasta encontrar la definición de objetos, que es lo que se desea encontrar, incluso si se está leyendo uno de los otros documentos que contienen sólo una vez la palabra objetos.

Ahora se realizará la misma búsqueda pero en el sistema SICIDEX, en la Figura-3.5 se muestra la pantalla en la cual se especificarán los documentos sobre los cuales se realizará la búsqueda.

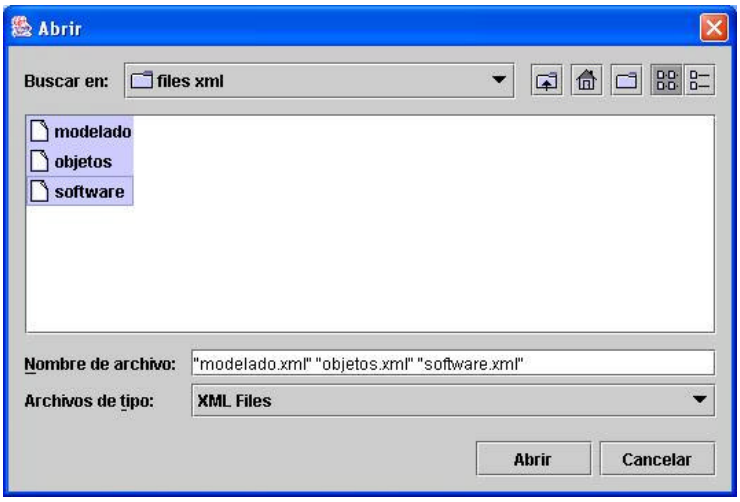

Figura - 3.5 Archivos sobre los cuales realizar una consulta

Para ejemplificar la búsqueda igual a la realizada en los documentos HTML, se seleccionarán los mismos documentos pero en formato XML. Una vez que se han seleccionado los archivos se especifica cual es el elemento sobre el cual se desea realizar la búsqueda; debido a que se esta realizando una búsqueda de la definición de objetos se elige el elemento <definicion> de entre los elementos que componen a los documentos, como se muestra en la Figura - 3.6.

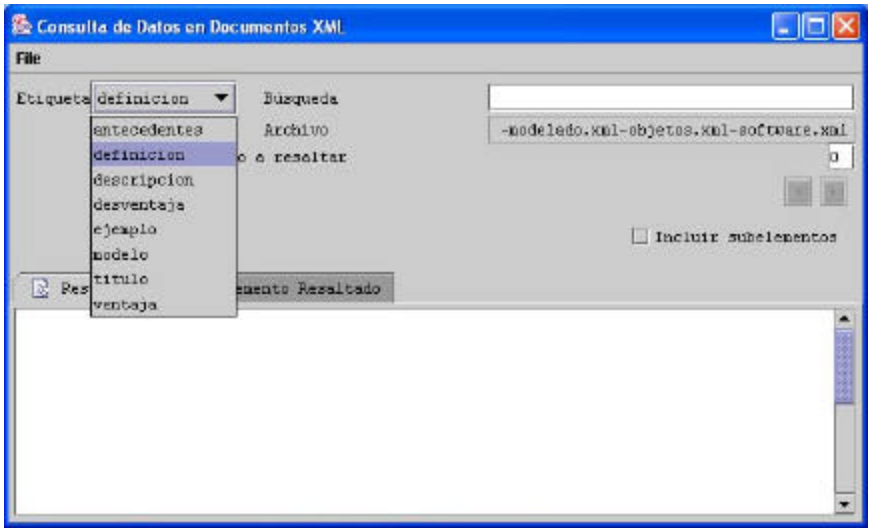

Figura - 3.6 Especificar el elemento sobre el cual consultar

Una vez que se ha especificado el elemento, se escribe la palabra objetos que es la búsqueda que se desea realizar, como se muestra en la Figura-3.7

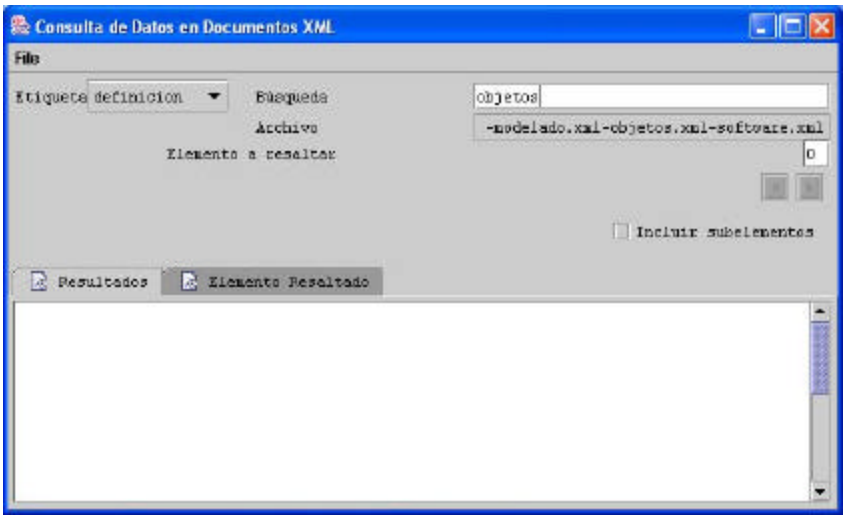

Figura-3.7 Escribir la palabra objetos para su consulta

El sistema mostrará todas aquellas coincidencias con la palabra especificada, identificándolas con un número consecutivo e indicando el documento en el cual se encontró, como se puede ver en la Figura-3.8

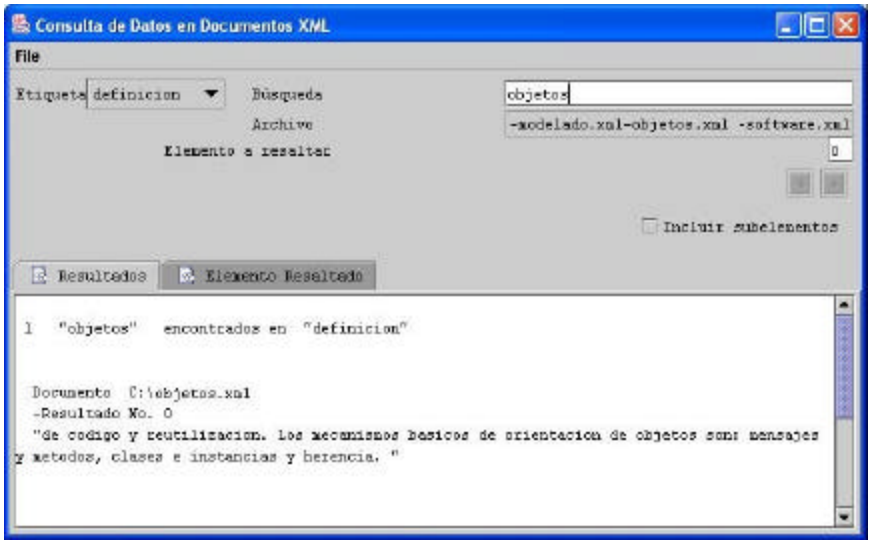

Figura-3.8 Visualización de palabras encontradas en el documento XML

Como se puede observar en la Figura-3.8 se muestra un único resultado ya que existe sólo una definición de objetos en los documentos.

Se puede ubicar el elemento buscado en el contenido total de los documentos consultados, como se muestra en la Figura-3.9

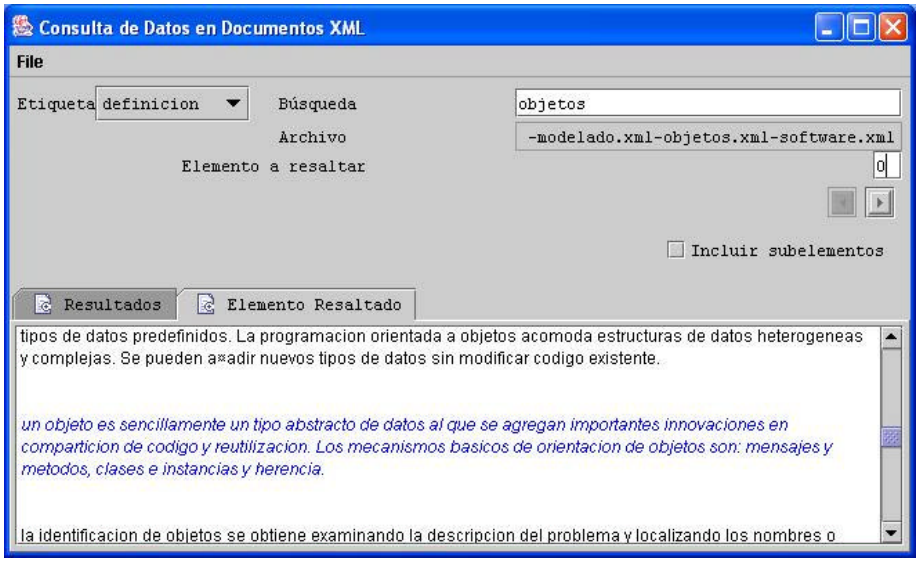

Figura-3.9 Ubicación de las coincidencias encontradas

Como se puede observar no fue necesario recorrer el contenido del documento objetos.xml ya que el cursor se colocó automáticamente en el lugar donde se encuentra la definición de objetos, además el párrafo donde se encuentra el elemento buscado es de un color diferente al del resto del texto, con esto se permite ubicar rápidamente el texto, en caso de que se desplazara por el resto del documento.

Cuando se realizó esta misma búsqueda en el documento en formato HTML se requirió leer línea por línea el documento hasta ubicar el lugar donde se encontraba la definición de objetos, además si se continuaba desplegando el resto del documento, se pierde la ubicación de la definición de objetos y nuevamente se tendría que leer el documento para encontrar la definición buscada.

Cuando se realizan consultas a un documento buscando un tópico especial, se requiere de una herramienta que permita realizar las consultas en la forma más simple y rápida posible, ya que el tiempo de consulta es muy valioso, sobre todo cuando se requiere consultar varios temas y diversos documentos.

Al crear una herramienta como SICIDEX la cual permite realizar consultas sobre documentos estructurados de diversos temas y con diferentes estructuras, se le está brindando al usuario la oportunidad de acceder más rápidamente a información específica y que pueda consultar varios documentos a la vez con mayor facilidad, sin que tenga que leer todos los documentos consultados para encontrar por ejemplo una definición.

Por ejemplo en la universidad Virtual de la Universidad Tecnológica de la Mixteca, los alumnos reciben un gran número de documentos de diversas áreas y si requirieran encontrar una definición necesitarían revisar documento por documento hasta encontrar la definición requerida.

Con la utilización de una herramienta de apoyo como SICIDEX se espera que la búsqueda de un elemento sea más rápida y fácil de realizar.

El sistema SICIDEX puede servir como base para un futuro desarrollo de un sistema en red, es decir permitir a los usuarios acceder a una base de datos integrada por todos los documentos de las diferentes materias que integran los cursos y realizar las consultas en línea.

CONCLUSIONES

El sistema SICIDEX permite realizar consultas sobre uno o varios documentos XML, integrando diferentes herramientas. Esto se logra al integrar un lenguaje de búsqueda como XQL, un procesador de búsqueda como XML Query Engine, un analizador XML y un lenguaje de programación como Java. Dando como resultado una aplicación fácil de utilizar donde toda la interacción se realiza mediante una interfaz gráfica.

El desarrollo de SICIDEX es un ejemplo de lo que puede llegar a realizarse con el uso de XML dentro de los motores de búsqueda que permitan al usuario buscar el contenido de datos XML y su estructura.

SICIDEX es una herramienta que permite realizar consultas sobre documentos estructurados, no se requiere que los documentos tengan la misma estructura, es decir pueden estar integrados por diversos elementos. Pueden consultarse varios documentos a la vez y las consultas se realizan en un ambiente gráfico. Los resultados y el contenido de los documentos se presentan en forma separada, permitiendo al usuario revisar el contenido de todos los documentos consultados sin perder la ubicación del elemento producto de su búsqueda.

Si se llegara a utilizar XML como lenguaje de estructuración de los documentos que se utilizan actualmente en la universidad virtual y los cuales actualmente se encuentran en formato HTML, se requerirá de la utilización de un DTD que permita crear documentos estructurados, como por ejemplo el documento que se presenta en el Apéndice C.1, en el Apéndice C.2 se incluye un ejemplo de un DTD, en el Apéndice C.3 se incluye un ejemplo de un documento XSL y en el Apéndice C.4 se incluye un ejemplo de un documento CSS, estos últimos se utilizarían para la presentación de la información en diferentes formatos, HTML entre ellos.

La estructura puede adecuarse según las necesidades de cada materia, es decir no es una estructura única la que se puede utilizar.

Una vez que los documentos hayan sido estructurados en XML, se pueden generar diferentes formatos de presentación de la información, por ejemplo para continuar con el formato HTML el cual se utiliza para colocar los documentos en línea o el formato PDF para enviarlos a los estudiantes.

Actualmente los navegadores más usados como el Internet Explorer y el Netscape Navigator ya soportan el XML, así que también podría colocarse la información en línea en formato XML.

En el Apéndice D se explica que programas se utilizaron y el procedimiento que se siguió para estructurar los documentos de pruebas que se utilizaron en el desarrollo del sistema.

Una siguiente etapa de desarrollo del sistema de consulta podría ser la de tener acceso al mismo por medio del Web mediante un sistema cliente-servidor, el cual podría dar acceso al usuario a una base de datos integrada por los documentos de las materias que integran los diferentes cursos mediante una conexión en línea.

XML podría ser el lenguaje que nos garantizará el intercambio de cualquier información, sin que ocasione problemas ya sea de tipo contenido o de tipo presentación.

REFERENCIAS

[Abiteboul,96] Abiteboul, S., Quass, D., Mchugh, J., Widom, J., Wiener, J.L. (1996). *The Lorel Query Language for Semistructured Data.* Consultado Agosto, 2002, de, http://www-db.stanford.edu/pub/papers/lorel96.ps

[Apache,00] The apache software foundation (2000), *The apache XML project.* Consultado Agosto, 2002, de, http://xml.apache.org/xerces-j/index.html

[Baru,98] Baru, C., Ludascher, B., Papakinstantinou, Y., Velikhov, P., Vianu, V. (1998), *Features and Requirements for an XML View Definition Language: Lessons from XML Information Mediation.* Consultado Agosto, 2002, de, http://www.w3.org/Tands/QL/QL98/pp/xmas.html

[Berners,95] Berners-Lee, T.; Connolly, D.W. (1995), *Hypertext Markup Language-2.0.* Consultado Agosto, 2002, de, http://www.w3.org/MarkUp/html-spec/ html-spec_12.html

[Birbeck,01] Birbeck, M., Diamond, J., Duckett, J., Gauti, O., Kibak, P., Lenz, E., Livingstone, S., Marcus, D., Mohr, S., Pinnock, J., Visco, K., Watt, A., Williams, K., Zaev, Z., Ozu, N.: *Professional XML.* Wrox, Canada (2001)

[Bryan,97] Bryan M. (1997), *SGML and HTML Explained.* Consultado Agosto, 2002, de, http://www.sgml.u-net.com/book/sgml-1.htm

[Ceri, 98] Ceri, S., Comai, S., Damiani, E., Fraternali, P., Paraboschi, E., Tanca, L. (1998), *XML-GL: A Graphical Language for Querying and Reshaping XML Document.* Consultado Agosto, 2002, de, http://www.w3.org/Tands/QL/QL98/pp/xml-gl.html

[Chamberlin,01] Chamberlin, D., Florescu, D., Robie, J., Siméon, J., Stefanescu, M. (2001), *XQuery: A Query Language for XML.* Consultado Agosto, 2002, de, http://www.w3.org/TR/2001/WD-xquery-20010215

[Deutsch,98] Deutsch, A., Fernández, M., Florescu, D, Levy, A., Suciu, D. (1998), *XML-QL: A Query Language for XML.* Consultado Agosto, 2002, de, http://www.w3.org/TR/NOTE-xml-ql19980819

[Deutsch,99] Deutsch, A., Fernández, M., Florescu, D., Levy, A., Suciu, D.(1999),*Querying XML Data.*Consultado Agosto,2002,de, http://www.informatik.uni-trier.de/~ley/db/journals/ /debu/DeutschFFLMS99.html

[Derksen,99] Derksen, E., Fankhauser, P., Howland, E., Huck, G., Macherius, I., Murata M., Resnick, M., Schöning, H. (1999), *XQL (XML Query Language).* Consultado Agosto, 2002, de, http://www.ibiblio.org/xql/xql-proposal.html

[Fernández,98] Fernández, J.M. (1998), *Java Parte I.* Consultado Agosto, 2002, de, http://es.tldp.org/LinuxFocus/pub/mirror/ LinuxFocus/Castellano/July1998/article57.html

[Fowler,99] Fowler, M., Scout, K.:*UML gota a gota.* Pearson, México (1999)

[GMD,01] GMD (2001), *GMD-IPSI XQL Engine*. Consultado Agosto, 2002, de, http://xml.darmstadt.gmd.de/xql

[Golfarb,99] Golfarb, C., Prescod, P.: *Manual de XML*. Prentice Hall, Madrid (1999)

[GoXML,02] XML Global Technologies Inc. (2002), *GoXML*. Consultado Agosto, 2002, de, http://www.xmlglobal.com

[Howart,02] Howart, K. (2002), *XML Query Engine*. Consultado Agosto, 2002, de, http://www.fatdog.com

[IBM,00] IBM Software (2000), *XML Parser for Java*. Consultado Agosto, 2002, de, http://www.alphaworks.ibm.com/aw.nsf/techmain/xml4j

[IBM,01] IBM Software (2001), *IBM Alpha Works*. Consultado Agosto, 2002, de, http://www.alphaworks.ibm.com/tech/xml4j

[Idris,99] Idris, A. (1999), *Java XML Applications Categories*. Consultado Agosto, 2002, de, http://www.developerlife.com/appoverview/default.htm

[Jansz,00] Jansz, K., Jim S.W., Indurkhya N., Maming C. (2000), *Using XSL y XQL for efficient, customised access to dictionary information*. Consultado Agosto, 2002, de, http://ausweb.scu.edu.au/aw2k/papers/jansz/paper.html

[Kweelt,01] Sahuget, A., Dupont, L. (2001), *KWEELT*. Consultado Agosto, 2002, de, http://kweelt.sourceforge.net/Kweelt

[Mecca,98] Mecca, G., Merialdo, P., Atzeni, P. (1998), *Do we really need a new query language for XML?*. Consultado Agosto, 2002, de, http://www.w3.org/TandS/QL/QL98/pp/MeMAql98.html

[Niagara,01] National science Foundation (2001), *Niagara Query Engine.* Consultado Agosto, 2002, de, http://www.es.wisc.edu/niagara

[Oracle,01] Oracle Inc. (2001), *Oracle XML Developer´s Kits*. Consultado Agosto, 2002, de, http://technet.oracle.com/tech/xml/parser_java2/content.html

[Robie,00] Robie, J., Chamberlin, D., Florescu, D. (2000), *Quilt: an XML Query Language.* Consultado Agosto, 2002, de, http://www.almaden.ibm.com/es/people/chamberlin/quilt_euro.html

[Robie,98] Robie, J., Lapp, J., Schach, D. (1998), *XML Query Language (XQL)*. Consultado Agosto, 2002, de, http://www.w3.org/TandS/QL/QL98/pp/xql.html

[Saic,02] Science Applications International Corp. (2002), *Saic SIM Server.* Consultado Agosto, 2002, de, http://www.saic.com

[Sun,01] Sun Microsystems Inc. (2001), *Java API for XML Processing*. Consultado Agosto, 2002, de, http://java.sun.com /xml/jaxp/index.html

[Wallace,02] Wallace, M., Runciman, C. (2002), *Xtract: a query language for XML documents*. Consultado Agosto, 2002, de, http://www.cs.york.ac.uk/fp/Xtract/xtract.html

[Walsh,99] Walsh, N., Muellner, L. (1999), *DocBook: The Definitive Guide,* [versión electrónica] Capitulo 1, http://www.docbook.org/tdg/en/html/docbook.html

[W3C,98] World Wide Web Consortium (1998), *eXtensible Markup Language (XML).* Consultado Agosto, 2002, de, http://www.w3.org/TR/REC-xml-19980210

[W3C,99] World Wide Web Consortium (1999), *HTML 4.01 specification*. Consultado Agosto, 2002, de, http://www.w3.org/TR/REC-html40

[W3C,02] World Wide Web Consortium (2002), *eXtensible Style Language (XSL).* Consultado Agosto, 2002, de, http://www.w3.org/Style/XSL

[Xql,98] World Wide Web Consortium (1998), *XML-QL: A Query Language for XML*. Consultado Agosto, 2002, de, http://www.w3.org/TR/NOTE-xml-ql-19980819

[Zhao,00] Zhao, B. Y. (2000), *XSet Enabling XML Applications*. Consultado Agosto, 2002, de, http://www.cs.berkeley.edu/~ravenben/xset

[XYZFind,02] XYZFIND CORPORATION (2002), *XYZFind.* Consultado Agosto, 2002, de, http://www.interwoven.com/news/press/2002/0327xyzpr.html

[X3,02] Docsoft Inc. (2002), *X-Cubed Search Engine.* Consultado Agosto, 2002, de, http://www.docsoft.com/x3Info.htm

APÉNDICE A
GLOSARIO DE ACRÓNIMOS

API	Application Programming Interface
CERN	European Organization for Nuclear Research
CGI	Common Gateway Interface
CSS	Cascading Style Sheets
DOM	Document Object Model
DTD	Document Type Definition
GML	Generalized Markup Language
HTML	HyperText Markup Language
ISO	International Organization for Standardization
NCSA	National Center for Supercomputing Applications
RTF	Rich Text Format
SAX	Simple Api for XML
SGML	Standard Generalized Language
W3C	World Wide Web Consortium
XML	eXtensible Markup Language
XSL	eXtensible Stylesheet Language

APÉNDICE B
MANUAL DE USUARIO

Bienvenidos al manual de usuario del "Sistema de consulta de información en documentos estructurados de XML" (SICIDEX), el cual es una herramienta de consulta sobre documentos estructurados de XML. En este manual se encuentra una explicación de cada una de las fases que componen el sistema.

Puesta en ejecución

SICIDEX es una aplicación independiente de las plataformas Linux y Windows. Todos los archivos de las clases compiladas, con XQEngine y el analizador SAX de Sun, están empaquetadas en un archivo Java JAR (Java Archive). Para ejecutar el archivo JAR se debe tener instalado el JRE (Java Runtime Environment) en la computadora. Si la variable CLASSPATH está correctamente direccionada, teclee:

java –jar Busqueda.jar

en la línea de comando se iniciará el sistema SICIDEX.

Formulación de una búsqueda

La formulación de una consulta en SICIDEX se puede realizar sobre uno o varios documentos XML para lo cual primero debe seleccionar los archivos XML a usar.

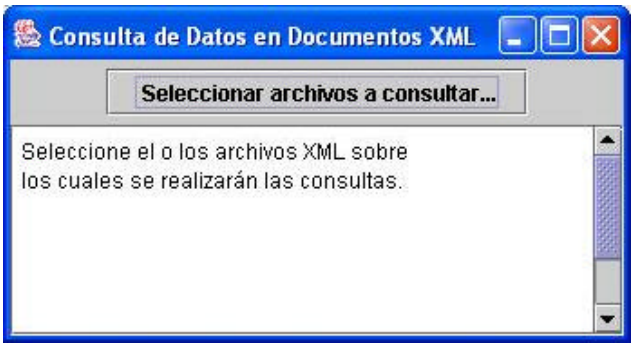

Figura-B.1 Elija los archivos a consultar

Permitiendo seleccionar en una estructura de directorios los archivos XML que se desean consultar, la búsqueda de los archivos en el árbol de directorios está diseñada para desplegar todos los documentos con extensión .xml que encuentre.

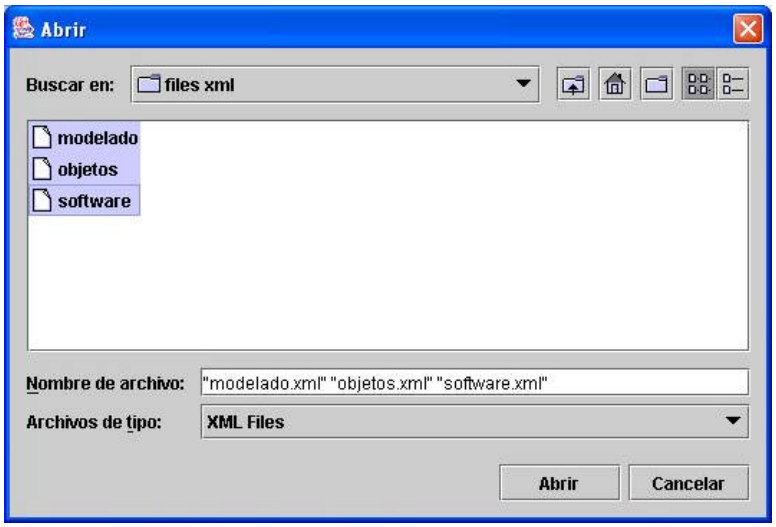

Figura-B.2 Selección de archivos en un árbol de directorios

La consulta en el SICIDEX se basa en el lenguaje de búsqueda XQL. En la Figura-B.3 se muestra la pantalla principal para realizar una búsqueda. Elija de una estructura jerárquica el elemento sobre el cual desea realizar la búsqueda, como se muestra en la Figura-B.4.

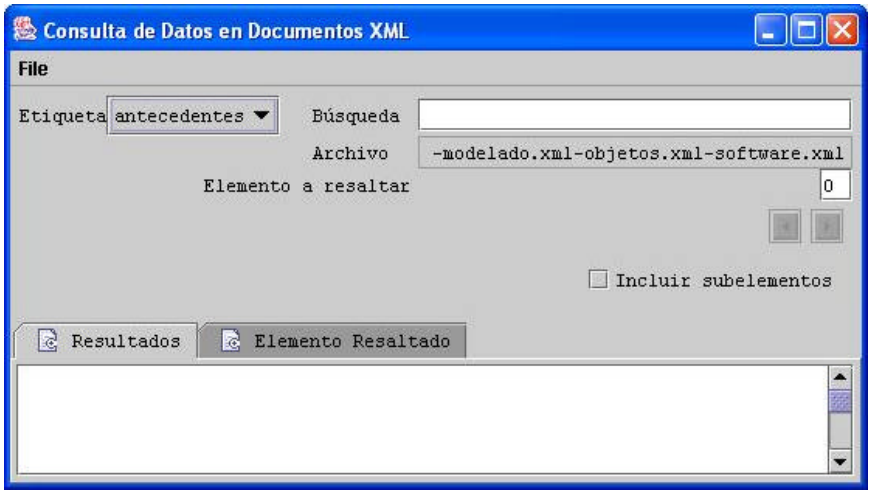

Figura-B.3 Pantalla principal para realizar una búsqueda

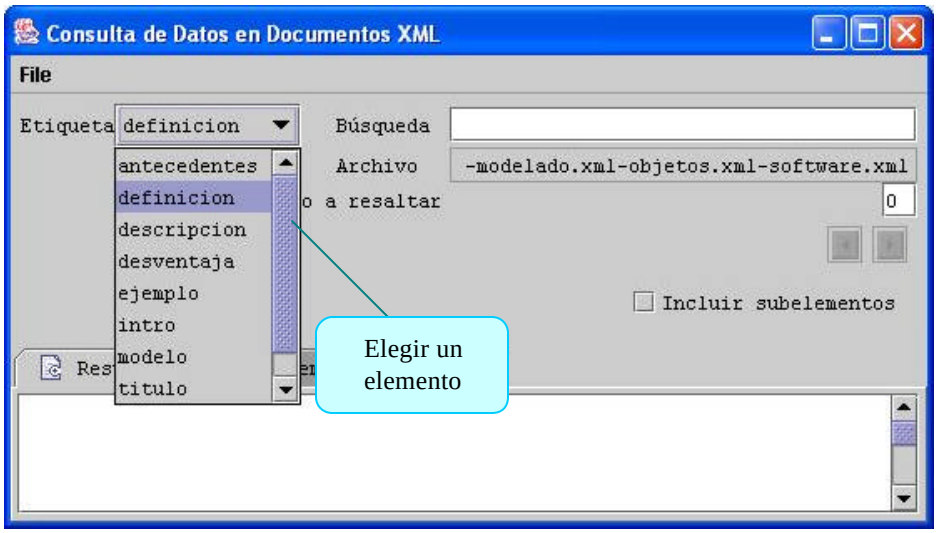

Figura-B.4 Selección de un elemento de la estructura del documento XML

Los elementos son identificados automáticamente de cada documento XML, independientemente de la estructura de cada uno de ellos. Para realizar una consulta teclee las palabras a buscar en la opción Búsqueda, como se muestra en la Figura-B.5, se puede indicar que la consulta se realice en los subelementos de la etiqueta elegida, por medio de activar la opción incluir subelementos.

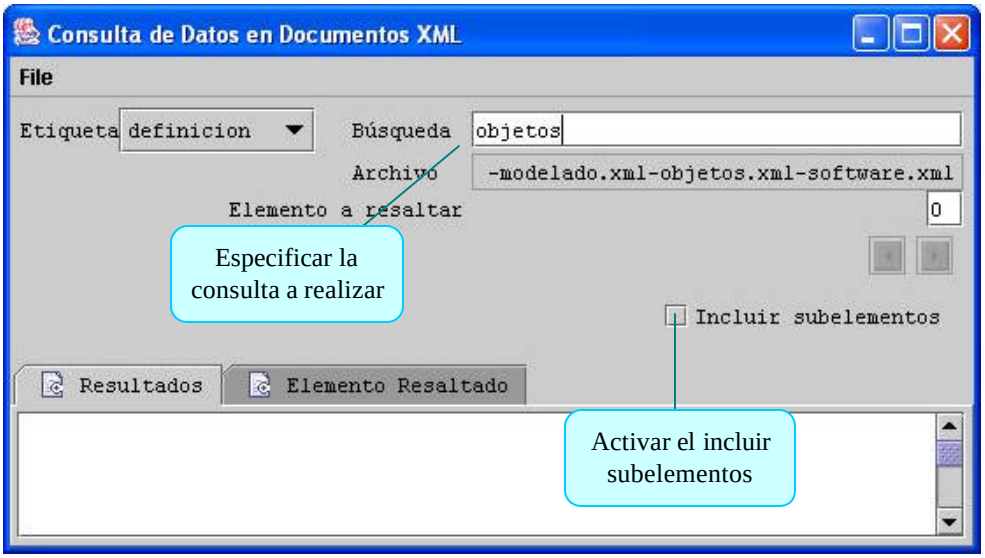

Figura-B.5 Ejemplo de una consulta

Visualización de los resultados

Una vez que se ha determinado el elemento y lo que se desea consultar, la aplicación muestra los resultados en forma numerada, permitiendo así ver la totalidad de los elementos encontrados, como se muestra en la Figura-B.6.

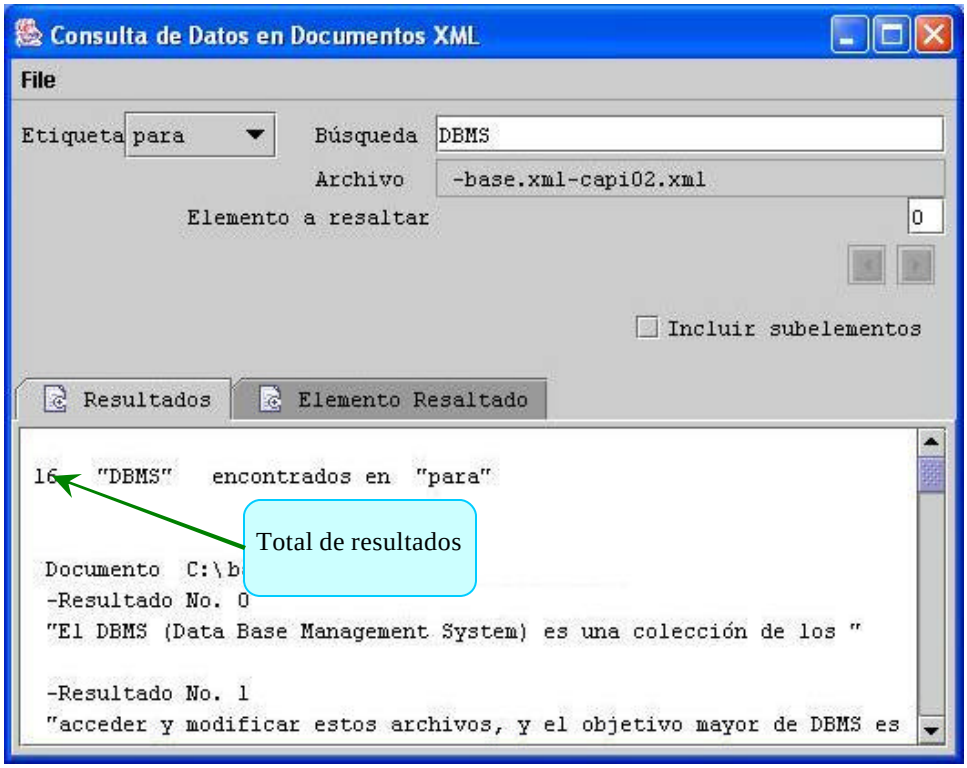

Figura-B.6 Resultados encontrados en los documentos XML

Como se puede observar en la Figura-B.7 se indica el documento en el cual se ha encontrado un elemento que coincida con la consulta así como el total de elementos encontrados y cada uno de ellos identificados con un número.

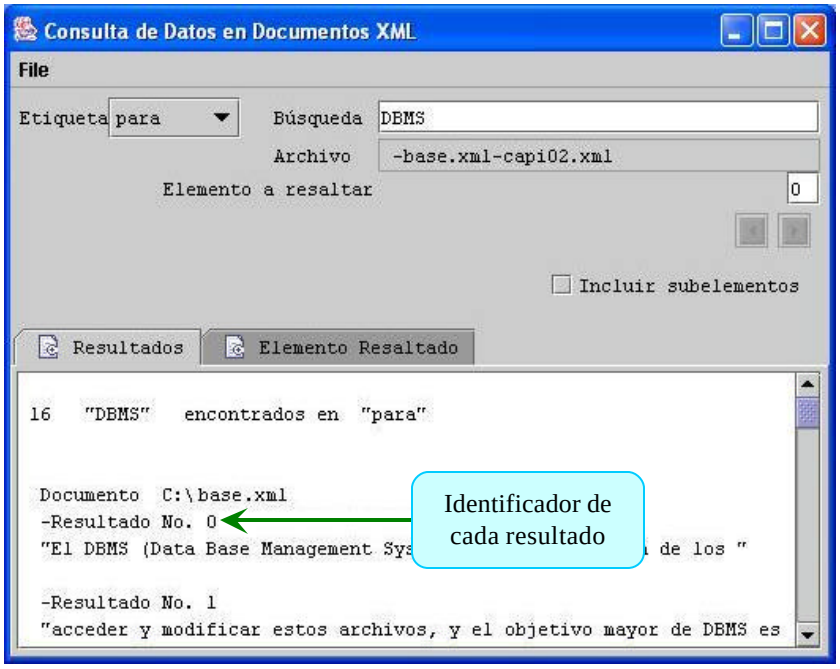

Figura-B.7 Identificación de resultados en cada documento

Si desea ubicar un elemento en particular dentro de todo el documento o documentos consultados, se debe especificar el elemento que se desea ubicar y la aplicación colocará el cursor en el elemento seleccionado, como por ejemplo lo que se muestra en la Figura-B.8:

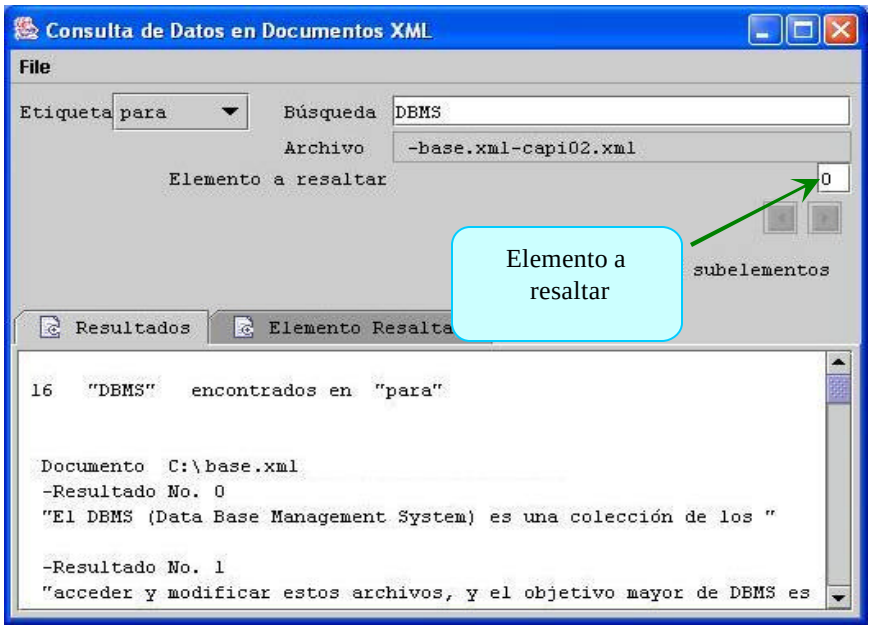

Figura-B.8 Indicar un resultado a ubicar dentro del documento

Como se puede observar en la Figura-B.9 se despliegan el o los documentos completos colocando el cursor en la ubicación exacta del elemento que se desea consultar, si se requiere ubicar algún otro elemento se especifica cual es el que se desea y el cursor es nuevamente colocado en la nueva selección.

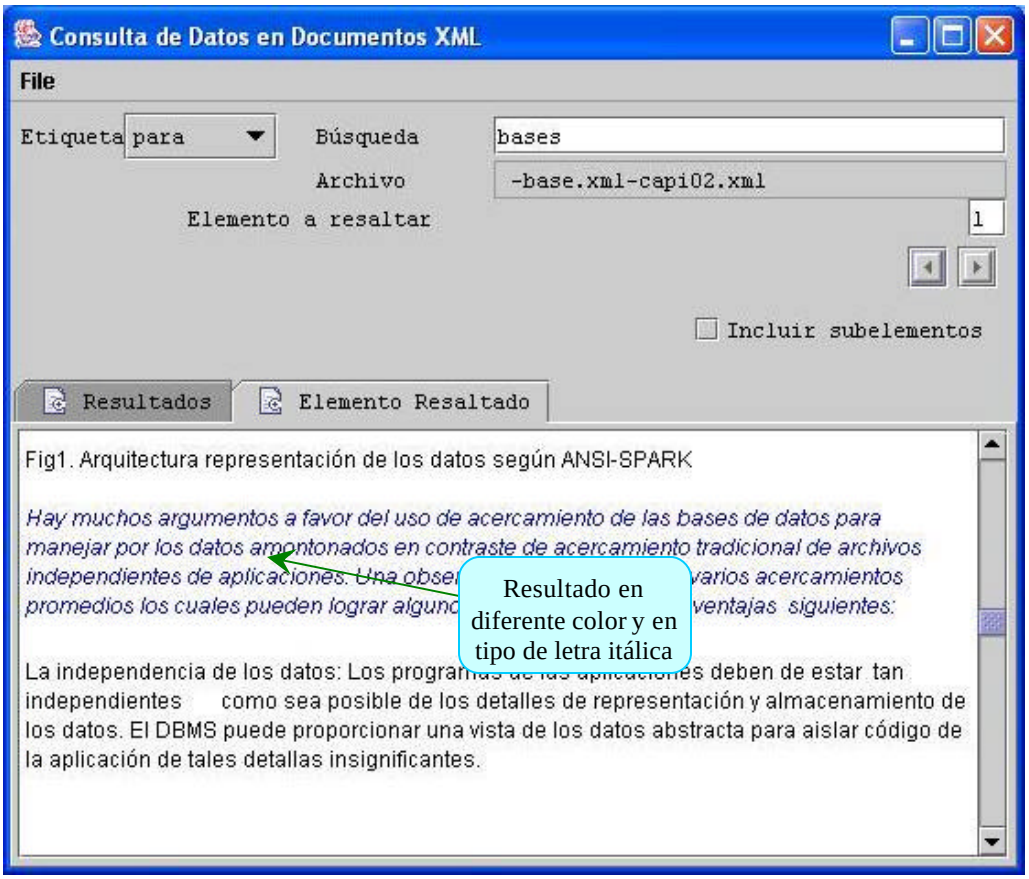

Figura-B.9 Visualización de todo el documento con un elemento resaltado

El SICIDEX despliega con un color de texto diferente los elementos elegidos, de forma que sea más fácil su ubicación dentro del despliegue de todos los documentos consultados.

Para navegar entre los diversos resultados se han colocado dos flechas que permiten pasar de resultado en resultado, como se muestra en la Figura-B.10.

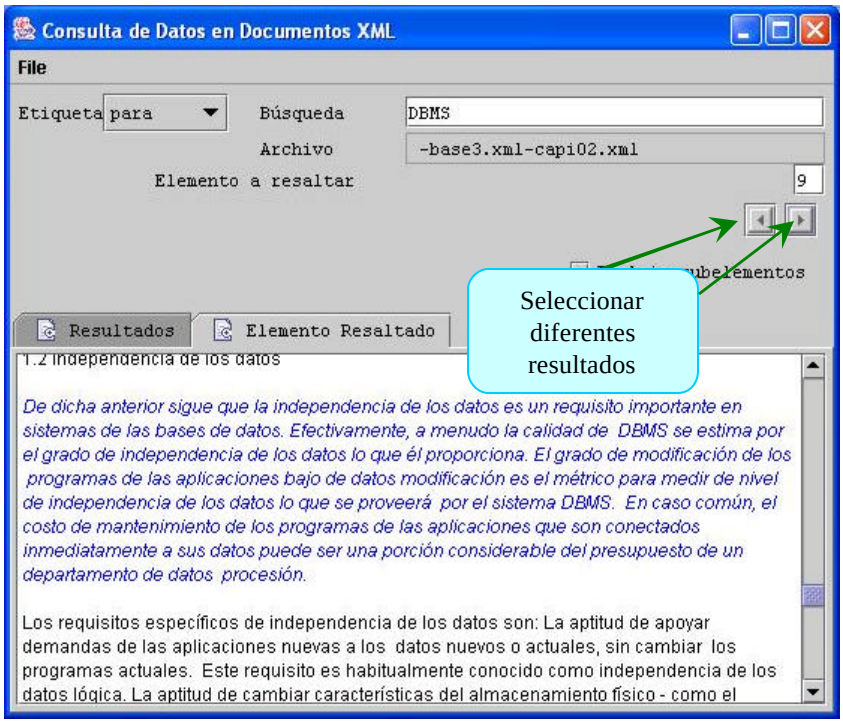

Figura-B.10 Indicar por medio de flechas el anterior o siguiente resultado

Se ha incluido un sistema de ayuda con los tópicos principales del sistema, como se muestra en la Figura-B.11

Figura-B.11 Seleccionar la ventana de ayuda

Una vez que se selecciona la opción de ayuda, se despliega una ventana conteniendo la explicación de los principales tópicos de manejo del sistema, por ejemplo en que consiste el sistema, que se debe hacer para realizar una búsqueda, como elegir un elemento sobre el cual realizar la consulta, etc, como se muestra en la Figura-B.12.

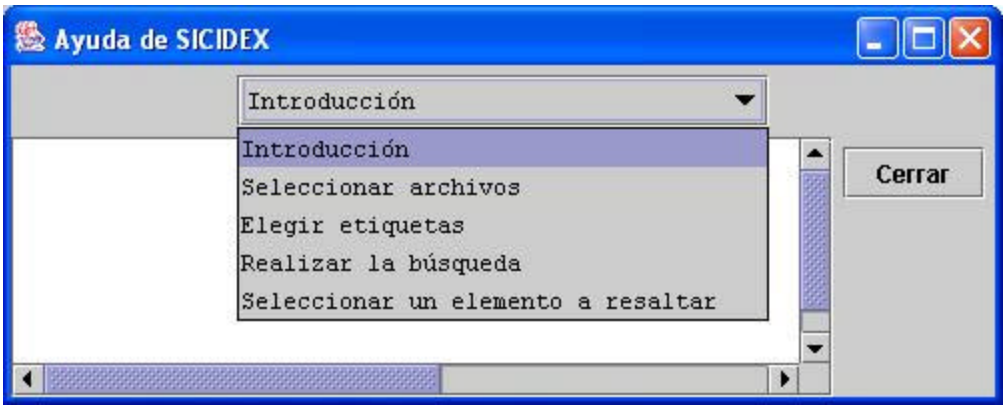

Figura-B.12 Ventana de ayuda

Se espera que este manual de usuario haya sido de utilidad en la utilización del sistema SICIDEX.

APÉNDICE C
EJEMPLO DE DOCUMENTO XML

C.1 Documento XML

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<?xml-stylesheet type="text/css" href="css-material.css"?>
<articulo xmlns:html="http://www.w3.org/TR/REC-html40" >
  <artencabezado>
    <titulo> Programa del curso Teoria avanzada de las bases de datos </titulo>
    <autor>
      <nombre>Yuri</nombre>
      <apellido>Paramonov</apellido>
    </autor>
  </artencabezado>
  <orderedlist>
    <listitem>Fundamentos subyacentes de las bases relacionales; </listitem>
      <itemizedlist>
          <listitem>Modelos de los datos jerárquicos; </listitem>
          <listitem>Modelos de los datos de la red; </listitem>
          <listitem>Modelos de los datos relacionales;</listitem>
      </itemizedlist>
      <listitem>Dependencias funcionales y formas normales </listitem>
    <listitem>Esquemas de refinación</listitem>
    <itemizedlist>
          <listitem>Descomposición;  </listitem>
          <listitem>Normalización  </listitem>
    </itemizedlist>
    <listitem>Seguridad y vistas en las bases de datos  </listitem>
    <listitem>Arquitectura de la base de datos  </listitem>
```

<listitem>Modelo físico de los datos </listitem>
<listitem>Índices (ISAM, B+Arbol) y algoritmos de datos manipulación en nivel físico </listitem>
<listitem>Concepto de transacción en las bases de datos y algoritmos de control de concurrencia </listitem>
<listitem>Estructura de bitácora y algoritmos de recuperación de los datos</listitem>
<listitem>Bases de datos extendidos y arquitecturas principales en tecnología ?cliente -servidor </listitem>
<listitem>Bases de datos paralelos </listitem>
<listitem>Bases de datos orientados a objetos </listitem>
<listitem>Bases de datos distribuidas </listitem>
</orderedlist>
<parrafo>Libros:</parrafo>
<sect1>
<titulo>Estructura de datos y arquitecturas correspondientes de las bases de
datos; fundamentales subyacentes de las bases de datos relacionales</titulo>
<parrafo>En caso común cada base de datos se puede considerarse como un modelo abstracto de un parte del mundo real. Esta representación abstracta a un objeto del mundo real se realiza por medio de un conjunto de valores de atributos pertinentes al objeto y agrupados en los archivos relacionados (todos objetos del mundo real son relacionados, porque son partes de un sistema global).</parrafo>
<itemizedlist>
<listitem>El nivel físico. El nivel mas bajo de abstracción describe como los datos complejos de nivel bajo efectivamente se guardan en memoria;
</listitem>
</itemizedlist>
<parrafo><enfasis>Las ventajas principales de las bases de datos.</enfasis></parrafo>
<figura>

<grafico xml:link="simple" show="replace"><html:img src="figura1.jpg" /></grafico>
<titulo>Fig1. Arquitectura representación de los datos según ANSI-SPARK</titulo>
</figura>
<parrafo>Mientras el acercamiento de la base de datos proveerá las ventajas numerosas al usuario, el también crea o hace mas duras problemas pertinentes en la área de seguridad y integridad de los datos. En el acercamiento tradicional, donde cada sistema de la aplicación tiene sus propios archivos, la cantidad limita de datos compartidos se logra por medio de transición de los archivos específicos de un sistema de aplicación a otro. En contraste, en el ambiente de la base de datos un juego grande de usuarios con requisitos de seguridad diferentes - algunos de ellos aun en régimen on-line - acceden a un conjunto de los datos común con niveles de seguridad diferentes. Este crecimiento en cantidad de los datos compartidos significa que el nivel de aseguración de la seguridad y privacidad de los datos correspondiente puede ser logrado por medio de control de acceso mas sofisticado. </parrafo>
<parrafo> En caso común, una base de datos (DB) puede ser considerada como u juego de objetos de la información u entidades y relaciones entre estas entidades (un modelo abstracto de una parte del mundo real). Relaciones u entidades se representan de alguna manera física en la base de datos, por ejemplo, por medio de los archivos y punteros, dependiendo del tipo de DBMS concreto. Debido al costo grande del diseño y desarrollo de las aplicaciones para las bases de datos, es urge deseable aumentar el ciclo de vida de las aplicaciones pertinentes a las bases de datos tanto como posible. Pero el mundo real está evolucionando continuamente y la base de datos, como modelo abstracto de una parte de ese mundo, deba de evolucionarse correspondiente, además, aparecen nuevos requisitos a la base de datos porque aplicaciones actuales se cambian durante del tiempo y nuevas aplicaciones se introducen. En ese contexto es obvio que el modelo abstracto de una parte del

mundo real que se elija para colocar en base de datos, debe de representar entidad de tal parte de mundo real la que es estable o se cambia más despacio durante del mundo evaluación real. </parrafo>

<sect2>

<titulo>1.2 independencia de los datos </titulo>

<parrafo>Entre los tres esquemas de ANSI/SPARK solo el esquema conceptual define el modelo conceptual de los datos, que es una representación abstracta de los datos en base de datos, y su semántica. El esquema conceptual también puede pensarse de como una descripción de información de la empresa que es simultáneamente implementación-independiente y aplicación-independiente. </parrafo>

<parrafo>El esquema interno especifica las detalla adicionales del almacenamiento, tales como la organización de los archivos usadas y las estructuras de los datos auxiliares (por ejemplo, índices) diseñados para la recuperación rápida. Esencialmente, el esquema interior refleja la idea principal cómo los datos descritos en el esquema conceptual realmente se guardan en dispositivos del almacenamiento secundarios como discos o cintas. La decisión terminal sobre el esquema interior se basa en la comprensión de los métodos típicos del acceso a los datos. DBMS, como el sistema del software, proveerá el acceso para los usuarios o para los programas de la aplicación a la base de datos, y también lleva a cabo las funciones y servicios necesarios internos, tales como: </parrafo>

</sect2>

<parrafo><enfasis>Modelos de los datos</enfasis></parrafo>

<parrafo> El concepto de modelo de datos es subyacente para estructura de la base de datos y el se representa como una colección de los métodos de descripción de los datos, relaciones entre datos, semántica de los datos y restricciones de su consistencia. La variedad de modelos de los datos los cuales han sido propuestos se divide por tres grupos diferentes:</parrafo>

<itemizedlist>

```
                <listitem> modelos lógicos objeto-basado; </listitem>
                <listitem> modelos lógicos registro-basado;
        </listitem>
                <listitem> modelos de datos físicos; </listitem>
        </ itemizedlist>
        <parrafo>El modelo objeto-orientado incluye mucho conceptos del
        modelo entidad-relación, pero  representa códigos ejecutables a la
        par de los datos. Este modelo fue aceptado muy rápido para el uso
        practico.</parrafo>
        <parrafo>El modelo relacional el que ha sido recibido el favor
        encima los dos modelos últimos será discutido detalladamente
        durante el curso. Los modelos de red y modelos jerárquicos todavía
        se usan en varias  bases envejecidas y por lo tanto les consideramos
        aquí brevemente</parrafo>
        <parrafo> <enfasis>Modelo de datos jerárquico. </enfasis> Varias
        variantes de modelos jerárquicos han sido implementadas en DBMS
        pero para nuestra descripción breve es suficiente considerar como
        el ejemplo solo DBNS de IBM – IMS (Information Management
        System).</parrafo>
    </sect1>
</articulo>
```

C.2 DTD del documento

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!ENTITY % chardatos " enfasis | comando | cita" >
<!ENTITY % listtype " itemizedlist | orderedlist | variablelist |programlisting
|figura " >
<!ENTITY % link     " ulink " >
<!ELEMENT articulo (artencabezado, (parrafo | sect1 | %listtype;)*) >
<!ELEMENT artencabezado (titulo, autor, (fecha | resumen)*) >
<!ELEMENT titulo (#PCDATA | %chardatos;)* >
<!ELEMENT resumen (parrafo+) >
<!ELEMENT autor (nombre,apellido,email?)>
<!ELEMENT nombre (#PCDATA) >
<!ELEMENT apellido (#PCDATA) >
<!ELEMENT email (#PCDATA) >
<!ELEMENT fecha (#PCDATA) >
<!ELEMENT sect1 (titulo, (parrafo | sect2 | %listtype; )*) >
<!ELEMENT sect2 (titulo, (parrafo | sect3 | %listtype; )*) >
<!ELEMENT sect3 (titulo, (parrafo | sect4 | %listtype; )*) >
<!ELEMENT sect4 (titulo, (parrafo | sect5 | %listtype; )*) >
<!ELEMENT sect5 (titulo, (parrafo | %listtype;)*) >
<!ELEMENT figura (titulo, img+) >
<!ELEMENT img EMPTY >
<!ELEMENT itemizedlist (listitem+) >
<!ELEMENT orderedlist (listitem+) >
<!ELEMENT variablelist (titulo?, varlistentry, varlistentry*) >
<!ELEMENT varlistentry (termino, listitem) >
<!ELEMENT programlisting (#PCDATA)* >
<!ELEMENT parrafo (#PCDATA | %chardatos; )* >
<!ELEMENT ulink (#PCDATA | %chardatos;)* >
<!ELEMENT termino (#PCDATA | %chardatos;)* >
<!ELEMENT listitem (#PCDATA | %listtype;)* >
<!ELEMENT comando (#PCDATA) >
<!ELEMENT enfasis (#PCDATA) >
<!ELEMENT cita (#PCDATA | %chardatos;)* >
```

```
<!ATTLIST articulo lang CDATA #IMPLIED>
<!ATTLIST ulink url CDATA #REQUIRED>
<!ATTLIST figura  label   CDATA #IMPLIED >
<!ATTLIST img src CDATA #REQUIRED align (center | left | right) #IMPLIED
>
<!ATTLIST sect1 id ID #IMPLIED >
<!ATTLIST sect2 id ID #IMPLIED >
<!ATTLIST sect3 id ID #IMPLIED >
<!ATTLIST sect4 id ID #IMPLIED >
<!ATTLIST sect5 id ID #IMPLIED >
```

C.3 XSL del documento

```
 <?xml version="1.0" encoding="iso-8859-1" ?>
 <xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:output method="html" encoding="iso-8859-1" doctype-
public="//w3c//dtd html 4.0 transitional//en"/>

<xsl:template match="articulo">
 <html>
 <head>
 <meta http-equiv="Content-Type" content="text/html; charset=iso-
8859-1"/>
 <title>
 <xsl:value-of select="artencabezado/titulo"/>
 </title>
 </head>
 <body bgcolor="#ffffff">
 <xsl:apply-templates/>
 </body>
 </html>
</xsl:template>

<xsl:template match="artencabezado">
 <div align="center">
 <xsl:apply-templates select="titulo"/>
 </div>
 <div align="right">
 <xsl:apply-templates select="autor"/>
 <xsl:apply-templates select="fecha"/>
 </div>
 <xsl:apply-templates select="resumen"/>
</xsl:template>

<xsl:template match="sect1">
```

```
  <xsl:apply-templates/>
</xsl:template>

<xsl:template match="sect1/titulo">
  <h2><xsl:apply-templates/></h2>
</xsl:template>

<xsl:template match="sect1/titulo" mode="crossref">
  <i><xsl:apply-templates/></i>
</xsl:template>
<xsl:template match="sect1">
  <xsl:apply-templates/>
</xsl:template>

  <xsl:template match="sect1/titulo">
    <xsl:element name="h2">
      <xsl:choose>
                    <xsl:when test="ancestor::*[@id]">
                      <xsl:element name="a">
                        <xsl:attribute name="name">
                          <xsl:value-of select="ancestor::*/@id"/>
                        </xsl:attribute>
                        <xsl:apply-templates/>
                      </xsl:element>
                    </xsl:when>
                    <xsl:otherwise>
                      <xsl:apply-templates/>
                    </xsl:otherwise>
      </xsl:choose>
    </xsl:element>
  </xsl:template>

<xsl:template match="sect2">
  <xsl:apply-templates/>
</xsl:template>
```

```
 <xsl:template match="sect2/titulo">
  <xsl:element name="h3">
   <xsl:choose>
                  <xsl:when test="ancestor::*[@id]">
                   <xsl:element name="a">
                    <xsl:attribute name="name">
                     <xsl:value-of select="ancestor::*/@id"/>
                    </xsl:attribute>
                    <xsl:apply-templates/>
                   </xsl:element>
                  </xsl:when>
                  <xsl:otherwise>
                   <xsl:apply-templates/>
                  </xsl:otherwise>
   </xsl:choose>
  </xsl:element>
 </xsl:template>
<xsl:template match="sect2/titulo" mode="crossref">
 <i><xsl:apply-templates/></i>
</xsl:template>

<xsl:template match="sect3">
 <xsl:apply-templates/>
</xsl:template>

 <xsl:template match="sect3/titulo">
  <xsl:element name="h4">
   <xsl:choose>
                  <xsl:when test="ancestor::*[@id]">
                   <xsl:element name="a">
                    <xsl:attribute name="name">
                     <xsl:value-of select="ancestor::*/@id"/>
                    </xsl:attribute>
                    <xsl:apply-templates/>
```

```
                    </xsl:element>
                  </xsl:when>
                  <xsl:otherwise>
                    <xsl:apply-templates/>
                  </xsl:otherwise>
    </xsl:choose>
   </xsl:element>
  </xsl:template>

<xsl:template match="sect3/titulo" mode="crossref">
  <i><xsl:apply-templates/></i>
</xsl:template>

<xsl:template match="sect4">
  <xsl:apply-templates/>
</xsl:template>

  <xsl:template match="sect4/titulo">
   <xsl:element name="h5">
    <xsl:choose>
                  <xsl:when test="ancestor::*[@id]">
                    <xsl:element name="a">
                      <xsl:attribute name="name">
                        <xsl:value-of select="ancestor::*/@id"/>
                      </xsl:attribute>
                      <xsl:apply-templates/>
                    </xsl:element>
                  </xsl:when>
                  <xsl:otherwise>
                    <xsl:apply-templates/>
                  </xsl:otherwise>
    </xsl:choose>
   </xsl:element>
  </xsl:template>
```

```
<xsl:template match="sect4/titulo" mode="crossref">
  <i><xsl:apply-templates/></i>
</xsl:template>

<xsl:template match="sect5">
  <xsl:apply-templates/>
</xsl:template>

  <xsl:template match="sect5/titulo">
    <xsl:element name="h5">
      <xsl:choose>
                    <xsl:when test="ancestor::*[@id]">
                      <xsl:element name="a">
                        <xsl:attribute name="name">
                          <xsl:value-of select="ancestor::*/@id"/>
                        </xsl:attribute>
                        <xsl:apply-templates/>
                      </xsl:element>
                    </xsl:when>
                    <xsl:otherwise>
                      <xsl:apply-templates/>
                    </xsl:otherwise>
      </xsl:choose>
    </xsl:element>
  </xsl:template>

<xsl:template match="sect5/titulo">
  <h5><xsl:apply-templates/></h5>
</xsl:template>

<xsl:template match="sect5/titulo" mode="crossref">
  <i><xsl:apply-templates/></i>
</xsl:template>

<xsl:template match="artencabezado/titulo">
```

```
<h1><xsl:apply-templates/></h1>
</xsl:template>

<xsl:template match="autor">
  <br/><xsl:apply-templates select="nombre"/>
  <xsl:text> </xsl:text>
  <xsl:apply-templates select="apellido"/>
  <xsl:apply-templates select="email"/>
</xsl:template>

<xsl:template match="email">
    <xsl:apply-templates/>
</xsl:template>

<xsl:template match="artencabezado/fecha">
<br/><br/><xsl:apply-templates/>
</xsl:template>

<xsl:template match="resumen">
<br/><br/><div align="center"><b>Resumen</b></div><br/>
<xsl:apply-templates/>
</xsl:template>

<xsl:template match="itemizedlist">
  <ul>
    <xsl:apply-templates/>
  </ul>
</xsl:template>

<xsl:template match="orderedlist">
  <ol>
    <xsl:apply-templates/>
  </ol>
</xsl:template>
```

```
<xsl:template match="variablelist">
  <dl>
    <xsl:apply-templates/>
  </dl>
</xsl:template>

<xsl:template match="programlisting">
  <div align="center">
  <table width="100%" border="1" bgcolor="#ccccff">
<tr><td><br/><br/><pre><xsl:apply-templates/></pre></td></tr>
  </table>
  </div>
</xsl:template>

<xsl:template match="varlistentry/termino">
  <dt>
    <b><xsl:apply-templates/></b>
  </dt>
</xsl:template>

<xsl:template match="varlistentry/listitem">
  <dd>
    <xsl:apply-templates/>
  </dd>
</xsl:template>

<xsl:template match="itemizedlist/listitem">
  <li>
    <xsl:apply-templates/>
  </li>
</xsl:template>

<xsl:template match="orderedlist/listitem">
  <li>
    <xsl:apply-templates/>
```

```
  </li>
</xsl:template>

<xsl:template match="parrafo">
  <p>
    <xsl:apply-templates/>
  </p>
</xsl:template>

<xsl:template match="cita">
&amp;quot;<xsl:apply-templates/>&amp;quot;
</xsl:template>

<xsl:template match="comando">
<b><xsl:apply-templates/></b>
</xsl:template>
<xsl:template match="enfasis">
<i><xsl:apply-templates/></i>
</xsl:template>
<xsl:template match="ulink">
  <xsl:element name="a">
    <xsl:attribute name="href">
      <xsl:value-of select="./@url"/>
    </xsl:attribute>
    <xsl:apply-templates/>
  </xsl:element>
</xsl:template>

  <xsl:template match="figura">
    <br/><br/>
     <xsl:element name="div">
      <xsl:attribute name="align">
                    <xsl:choose>
                      <xsl:when test='grafico[@align]'>
                        <xsl:value-of select="grafico/@align"/>
```

```
                    </xsl:when>
                    <xsl:otherwise>
                      <xsl:text>center</xsl:text>
                    </xsl:otherwise>
                  </xsl:choose>
    </xsl:attribute>
    <xsl:element name="img">
                  <xsl:attribute name="src">
                    <xsl:value-of select="grafico/@fileref"/>.<xsl:value-of
select="grafico/@format"/>
                  </xsl:attribute>
                  <xsl:attribute name="alt">
                    <xsl:value-of select="grafico/@fileref"/>.<xsl:value-of
select="grafico/@format"/>
                  </xsl:attribute>
    </xsl:element>
    <br/><br/>
    <b>Figura <xsl:value-of select="./@label"/></b><xsl:apply-
templates select="titulo"/>
   </xsl:element>
   <br/><br/>
  </xsl:template>

<xsl:template match="figura/titulo">
<xsl:text>: </xsl:text><xsl:apply-templates/>
</xsl:template>
<xsl:template match="anchor">
  <xsl:element name="a">
    <xsl:attribute name="name">
      <xsl:value-of select="./@id"/>
    </xsl:attribute>
  </xsl:element>
</xsl:template>

</xsl:stylesheet>
```

C.4 CSS del documento

```
titulo
{
   COLOR: blue;
   DISPLAY: block;
   FONT-FAMILY: 'Arial Black', Arial;
   FONT-SIZE: 14pt;
   FONT-STYLE: italic;
   TEXT-ALIGN: center;
   PADDING: 5pt;
}

nombre
{
   COLOR: blue;
   FONT-FAMILY: Arial;
   DISPLAY: inline;
   FONT-SIZE: 12pt;
}

apellido
{
   COLOR: blue;
   DISPLAY: inline;
   FONT-FAMILY: Arial;
   FONT-SIZE: 12pt;
}
email
{
   COLOR: blue;
   DISPLAY: inline;
   FONT-FAMILY: Arial;
   FONT-SIZE: 12pt;
}
```

```
fecha
{
   COLOR: black;
   DISPLAY: block;
   FONT-FAMILY: Arial;
   FONT-SIZE: 12pt;
   TEXT-ALIGN: right;
}

sect1 titulo
{
   COLOR: green;
   DISPLAY: block;
   FONT-FAMILY: Arial;
   FONT-SIZE: 12pt;
}

sect1 sect2 titulo
{
   COLOR: violet;
   DISPLAY: block;
   FONT-FAMILY: Arial;
   FONT-SIZE: 12pt;
}

sect1 sect2 sec3 titulo
{
   COLOR: orange;
   DISPLAY: block;
   FONT-FAMILY: Arial;
   FONT-SIZE: 12pt;
}
```

```
sect1 sect2 sec3 sect 4 titulo
{
   COLOR: grey;
   DISPLAY: block;
   FONT-FAMILY: Arial;
   FONT-SIZE: 12pt;
}

sect1 sect2 sec3 sect 4 sect5 titulo
{
   COLOR: pink;
   DISPLAY: block;
   FONT-FAMILY: Arial;
   FONT-SIZE: 12pt;
}

parrafo
{
   COLOR: black;
   DISPLAY: block;
   FONT-FAMILY: Arial;
   FONT-SIZE: 12pt;
   TEXT-ALIGN: justify;
   TEXT-INDENT: 20pt;
   MARGIN: 2pt;
   PADDING: 10pt;
}

itemizedlist
{
   COLOR: black;
   DISPLAY: block;
   LIST-STYLE: unordered;
   list-style-type: disc;
   MARGIN-LEFT: 40px;
```

```
    FONT-FAMILY: Arial;
    counter-reset: item;
}

itemizedlist listitem
{
    DISPLAY: list-item;
    PADDING: 5pt;
    FONT-FAMILY: Arial;
    TEXT-ALIGN: justify;
}

enfasis
{
    COLOR: black;
    DISPLAY: inline;
    FONT-FAMILY: Arial;
    FONT-SIZE: 12pt;
    FONT-WEIGHT: bold;
}

orderedlist
{
    display:block;
    margin-left:40px;
    list-style-type:decimal;
    counter-reset: item;
}

orderedlist listitem
{
    display:list-item;
}
```

```
orderedlist
{
    content: counter(item);
    counter-increment: item;
}

figura
{
  display: block;
  align: middle;
}

figura titulo
{
   COLOR: black;
   DISPLAY: block;
   FONT-FAMILY: 'Arial Black', Arial;
   FONT-SIZE: 12pt;
   FONT-STYLE: italic;
}

grafico
{
   display: block;
   align: middle;
}

variablelist
{
  display: inline;
}
varlistentry
{
  display: inline;
}
```

```
programlisting
{
  display: inline;
}

link
{
  display: inline;
}

ulink
{
  display: inline;
}

termino
{
  display: inline;
}

comando
{
  display: inline;
  font-style: italic;
}

cita
{
    COLOR: black;
    DISPLAY: block;
    FONT-FAMILY: 'Arial Black', Arial;
    FONT-SIZE: 10pt;
    FONT-STYLE: italic;
}
```

APÉNDICE D
ANÁLISIS DE HERRAMIENTAS CONSIDERADAS

Para el desarrollo de SICIDEX, se instalaron, evaluaron y desarrollaron varios programas. A continuación, se presentan algunos de los problemas más relevantes que permitieron llegar a la elección del procesador de búsqueda y el analizador que finalmente se utilizó para el desarrollo del Sistema.

1. Se investigó sobre los conceptos básicos y filosofía de XML.

2. Una vez que se estudiaron los conceptos básicos se empezó por realizar un DTD, en este caso se desarrolló un DTD adecuado al material de la Universidad Virtual, sólo como ejemplo y para poder realizar las pruebas pertinentes.

3. Una vez creado un documento estructurado basado en el DTD, se creó una hoja de estilo CSS, que permitiera visualizar el documento en un navegador que reconociera la estructura del mismo, en este caso se realizaron las pruebas con el navegador Mozilla para Linux.

4. Además se realizaron pruebas para utilizar un XSL que permitiera transformar un documento estructurado XML a otros formatos (HTML, PDF, RTF, PostScript, etc.).

5. La aplicación de un XSL a un documento XML puede ocurrir tanto en el origen (por ejemplo, un servlet que convierta de XML a HTML para que sea mostrado a un navegador conectado a un servidor Web), o en el mismo navegador (como en el caso del MS IE5 y en Netscape 5).

6. En este caso se usó Emacs, psgml, sp, sablotron y expat, para la creación del archivo XML basado en un DTD y la conversión de XML a HTML.

7. En Emacs, se abre un documento nuevo con extensión XML indicando en el encabezado del mismo cuál es el DTD que se usará. Esto permite que Emacs ejecute el modo XML de psgml, apareciendo menús adicionales para trabajar con XML. De esta manera se puede empezar a introducir cada una de las etiquetas que se han declarado en el DTD así como su contenido.

8. Una vez terminado el archivo XML y que ha sido validado ya se puede usar.

Se investigaron y evaluaron diversas herramientas que permitieran realizar búsquedas sobre documentos XML, así se encontró primeramente la herramienta XML-QL:

Para que XML-QL funcionara se tuvo que instalar el JDK 1.1.7, ya que es la única versión de Java con la que trabaja. Se realizaron pruebas con XML-QL versión 0.6 y versión 0.9, además de que XML-QL requiere tener instalado strudel, xml4j y OROMatch.

El inconveniente de usar el XML-QL es que se requería realizar todas las actividades en una página HTML mediante un formulario, un CGI y un archivo de comandos propio de XML-QL. Esto se realiza por medio de la línea de comandos:

```
xmlql.cmd archivo-entrada archivo-salida
```

En el “archivo-entrada” se especifica el comando de búsqueda y en el “archivo-salida” se almacenan los resultados encontrados, por lo tanto es necesario tener acceso de escritura en el directorio de trabajo.

Una vez que el usuario introducía en el formulario de la página HTML la consulta a realizar, ésta se tenía que traducir de tal manera que fuera entendible por el XML-QL para poder ejecutarla.

Se requiere además de la ejecución del comando:

```
xsl.sh archivo-xml archivo-xsl archivo-html
```

Este comando lo que hace es la transformación del archivo XML en un archivo HTML usando un archivo XSL. Es decir permite crear un documento HTML que pudiera ser desplegado por el navegador y de ésta manera visualizar los resultados de la búsqueda.

Pero no se pudo usar este comando ya que enviaba un error de Java

```
org/apache/xalan/xslt/XSLTEngineImpl
```

Esto finalmente no sería un problema ya que existen otras herramientas que convierten un archivo de XML en HTML por medio de un archivo XSL, como es el caso de Sablotron el cual es un procesador XSLT implementado en C++ basado en el analizador Expat XML.

El problema que se encontró al utilizar estas herramientas para la búsqueda, fue que todo se tenía que realizar directamente en la línea de comandos pero no se podía realizar por medio del CGI, el cual fue programado, porque enviaba error al tratar de escribir los resultados. Esto finalmente es una protección de Linux al no permitir que se escriba un archivo en el sistema por cualquier usuario.

Otro inconveniente que se encontró fue el uso del CGI, el cual conlleva algunas vulnerabilidades de seguridad, esto es por que se recibían los parámetros por medio del formulario y el CGI se encargaba de llevar a cabo la búsqueda.

El formulario y el CGI desarrollados, funcionan correctamente por separado (directamente en la línea de comandos y como usuario Web). La restricción de escritura en el servidor podría ser solucionada, pero el inconveniente continuaba siendo el realizar todo por medio de un CGI que finalmente debería tener parámetros de ejecución y escritura en el servidor, además

de estar ejecutando el CGI cada que se realizara una búsqueda y esto podría llegar a saturar el servidor, sobre todo al realizar búsquedas en documentos muy grandes por varios usuarios al mismo tiempo (dadas las características del actual Servidor de la Universidad Virtual sobre el cual se llevaron a cabo las pruebas).

XMLQuery Engine

Para poder revisar XMLQuery Engine se usó j2sdk1.3.1.

Uno de los problemas que se encontraron al desarrollar esta aplicación fue la falta de experiencia en la programación en Java, siendo uno de los obstáculos a vencer.

Una vez instalado el software XMLQuery Engine, enviaba errores al tratar de usarlo, no existe información referente a errores de instalación, es decir, según las instrucciones de instalación y ejecución de XMLQuery Engine lo único que se requiere es Java, XQEngine y Jaxp, pero después de probar con diversos programas se llegó a la conclusión de que se requería tener instalado y correctamente direccionado el sistema Xerces.

Después de instalado este software finalmente se pudo ejecutar XMLQuery Engine, revisar sus características y ver si cumplía con los requerimientos para poder desarrollar el software que se tenía planeado. Una de las ventajas de este software es que está basado en Java, además de que usa XQL para realizar las búsquedas, las cuales son mucho más expresivas y poderosas que las búsquedas realizadas en interfaces basados en Web.

XMLQuery Engine es un componente que permite indexar y realizar búsquedas en archivos XML, por elementos, atributos o contenido de texto completo. Puede usar los analizadores SAX1 o SAX2 y DOM.

La ventaja principal que se observó en esta herramienta fue la del uso de la plataforma Java, con lo cual se esperaba tener la independencia de la plataforma en la cual se ejecutara el sistema, es decir poder utilizar el sistema tanto en un ambiente Linux como Windows (principales plataformas de uso de la Universidad Virtual).

Para poder transportar la aplicación de la plataforma Linux a Windows, se creó un archivo JAR que contiene las clases compiladas e imágenes que usa la aplicación. Además de incluir las clases necesarias para su ejecución y que son parte de los programas: XML Query Engine, Xerces.jar y Jaxp.jar.

Una vez que se ejecutó la aplicación en una computadora con ambiente Windows, la cual sólo tiene instalado j2sdk1.2, enviaba errores de ejecución.

Este tipo de errores los envían algunas aplicaciones que no encuentran el paquete parser.jar, el cual se supone forma parte del paquete Jaxp. Por alguna razón el Jaxp1-1 no incluye el archivo parser.jar entre sus archivos una vez que es instalado.

Se tuvo que buscar dicho archivo en Internet y colocar sus archivos de clases junto con los demás archivos necesarios para la ejecución de la aplicación, resultando finalmente una satisfactoria ejecución del “Sistema de Consulta de Información en Documentos Estructurados de XML” en la plataforma Windows.