UNIVERSIDAD NACIONAL AUTÓNOMA
DE MÉXICO

"XMLVIEW – MECANISMO PARA LA CREACIÓN DE VISTAS SOBRE DOCUMENTOS XML ALMACENADOS EN LA BASE DE DATOS NATIVA EXIST"

T E S I S

QUE PARA OBTENER EL GRADO DE :

MAESTRA EN INGENIERÍA (COMPUTACIÓN)

P R E S E N T A:

ROSA ISELA LÓPEZ AGUILAR

DIRECTORA DE TESIS:

DRA. AMPARO LÓPEZ GAONA

México, D.F. 2010

A mi familia con Amor.

Agradecimientos

Agradezco a mi Familia por haber inculcado en mí los deseos de superación y por su apoyo incondicional.

Agradezco a Luis particularmente por haberme acompañado y apoyado todos estos días.

Gracias Ronny por haber estado ahí siempre, por los ánimos que me diste para continuar éste proceso, por tu apoyo y por haberme ayudado con tus sugerencias a mejorar éste trabajo.

Gracias Amparo por haber confiado en mí y por todo el apoyo brindado en la culminación de éste trabajo. Me trataste no como a una alumna, si no como a una amiga, te respeto y te admiro.

Gracias a mi jurado: Dra. Pilar Ángeles, Dr. Renato Barrera, M.C Gustavo Márquez y Dra. Hanna Oktaba, por sus valiosas sugerencias.

*A*gradezco a Dios por haberme dado la sabiduría para concluir ésta meta.

Índice general

Índice de figuras

1

Introducción

Hoy en día se ha incrementado el uso del estándar XML, lo cual permite a las organizaciones estructurar sus documentos importantes de forma ordenada y sobretodo estandarizada, lo que facilita el compartir información entre ambientes heterogéneos. Por estas ventajas, el XML es una tecnología ahora usada en conjunto con las bases de datos. Estas últimas se han reconocido por proporcionar el almacenamiento de información de forma confiable, segura, permitiendo que los datos en cualquier momento se encuentren consistentes y puedan ser accedidos de forma rápida, por tal razón, actualmente se almacenan documentos XML en bases de datos, gozando de todos los beneficios que éstas les proporcionan. Existen dos tipos de bases de datos para almacenar documentos XML: las habilitadas y las nativas (*N*ative XML Database, NXD). Las bases de datos habilitadas son aquellas que permiten el almacenamiento

de documentos XML convirtiendo éstos a su correspondiente formato relacional. Las NXD definen un modelo lógico para almacenar y obtener documentos XML en su formato original.

Debido a las características observadas en ambos tipos, éste trabajo utilizará como almacén de documentos una NXD llamada eXist.

Cuando se trabaja con documentos XML, casi siempre se opta por almacenarlos en una base de datos. Ahora bien, una vez almacenada la información presente en esos documentos debería poder consultarse, por tal razón existen lenguajes de consulta para XML, tales como: XPath y XQuery. Ambos son capaces de ejecutarse y mostrar como resultado fragmentos de documentos, que cumplen con las condiciones que se especifican en las consultas.

Uno de los elementos más importantes con los que cuenta una organización es su información, por lo que se debería poder mostrar a los usuarios, dependiendo de su rol, ciertas partes de ella. No solo por cuestiones de seguridad, si no también por rapidez en la ejecución de la consulta; por está razón fueron creadas las vistas en el modelo relacional. Una vista según este modelo, se encarga de extraer cierta parte de información de la base de datos y almacenarla en una relación virtual, de tal manera que la relación no existe físicamente, pero es posible consultarla como si realmente lo estuviera. Con base en esta idea, para el caso de los documentos XML también es nece-

sario el uso de vistas. Al crear una vista se debe almacenar la consulta que la define, lo cual implicaría crear un "nuevo documento XML", que no estará almacenado físicamente, si no que será considerado como un documento–vista virtual, porque no almacena los datos resultantes de la consulta, si no la consulta que los obtiene.

En el caso de las vistas creadas sobre documentos XML, también se permite la consulta sobre estas, lo cuál proporcionaría mínimamente dos ventajas:

- Seguridad: al proporcionar al usuario sólo una parte del documento XML que quiere consultar, en vez del documento completo, logrando de ésta forma el nivel de seguridad deseado, ya que cada usuario tendrá acceso únicamente a la parte de la información que la organización le permite.

- Acceso rápido a la información: porque al consultar únicamente sobre la parte de información que el usuario puede ver, tendría necesariamente que ser más rápida la consulta ya que no se forma el resultado con la búsqueda sobre todos los elementos del documento que cumplan la condición, sino que únicamente con la parte que ya se encuentra resumida en la vista.

Este trabajo de investigación retoma el concepto de vistas tal como lo han manejado las bases de datos relacionales, para

así poder otorgar una herramienta que pueda crear, consultar y borrar vistas sobre documentos XML. Las vistas como se dijo anteriormente son importantes, sobretodo por el auge que ha tomado el formato XML en los últimos años; en los siguientes capítulos se explicará más a detalle en que consiste esta herramienta y como será creada.

1.1 Objetivos

Actualmente existen algunas propuestas de cómo resolver la creación de vistas sobre documentos XML; pero desafortunadamente ninguna de ellas ha sido implementada, lo cuál sigue dejando este problema abierto, porque si no existe una herramienta que las cree, las ventajas que éstas ofrecen no están siendo aprovechadas. Así como han surgido las cláusulas de cada lenguaje de consulta como lo es XQuery, deben existir simples comandos para crear vistas, porque XSLT (*E*xtensible Stylesheet Language Transformations) no puede hacer el trabajo que le corresponde a las vistas, éste fue creado únicamente para presentar información de forma agradable al usuario y no como un lenguaje de consulta.

El objetivo principal de este trabajo es:

- Analizar el concepto y la implementación de vistas en el modelo relacional, como base para el desarrollo de un mecanismo de creación de vistas sobre documentos XML

almacenados en bases de datos nativas que proporcione un documento–vista XML bien formado.

1.2 Relevancia y Contribución

La presente investigación es relevante porque trata de un problema hasta hoy en día no resuelto, existen propuestas muy ingeniosas al respecto, pero aún no se desarrolla software donde se diga que se aplicó alguna de ellas, por lo que no se sabe si éstas resolverían el problema de forma eficiente. Es importante mencionar que este tema no es sencillo ya que se trata de documentos XML, los cuales pueden llegar a tener una estructura bastante compleja, haciendo más interesante el problema.

Este trabajo contribuirá de forma benéfica proporcionando una herramienta que permitirá manejar documentos XML, crear vistas sobre ellos y consultarlas sin pérdida de información, además de proporcionar una vista resultante bien formada.

Cabe aclarar que la herramienta propuesta será una primera versión, debido a que el tema es bastante amplio, por lo que estará aún muy limitada con respecto a los avances que se tienen en el modelo relacional.

1.3 Trabajos relacionados

La creación de vistas sobre documentos XML ha sido abordada anteriormente por varios artículos, en [1] el tema de vistas es asociado específicamente al control de acceso en los documentos XML. De tal forma que los usuarios pueden tener acceso a los datos mediante un control que involucra listas de capacidades, listas de control de acceso y matrices, además se aprovecha la característica que tiene todo documento XML de poder ser dividido en conjuntos de elementos para categorizar el acceso a diferentes partes del documento. En éste trabajo se consideran las siguientes restricciones de vista:

- A nivel documento. Se restringe el número de documentos que el usuario o grupo de usuarios puede ver.
- A nivel nodo. Debido a que un documento XML está formado por nodos, se imponen restricciones a algunos de éstos nodos en particular, para que únicamente los puedan ver aquellos usuarios o grupos de usuarios que tengan permisos de acceso sobre ellos.
- A nivel vista combinada. Esta restricción es una combinación de las anteriores, de tal forma que los usuarios o grupos de usuarios podrán ver cierto número de documentos y se permitirá el acceso a sólo algunas partes de ellos.

Otro de los trabajos antecedentes de esta investigación es [12] donde se propone que una vista XML debe de crearse como un documento XML imaginario, debido a que este documento no existe por sí mismo, sino que se obtiene a partir de otros documentos que están físicamente almacenados. Así como también se menciona el darle al usuario la capacidad de modelar XML desde un alto nivel de abstracción usando metodologías ya existentes, tales como modelos conceptuales orientados a objetos, usando notaciones del Lenguaje de Modelado Unificado (UML) y transformando éstos modelados directamente en un modelo XML. El punto más importante en esta propuesta es que la vista no es creada a partir de una consulta, sino que a partir de un mapeo desde el modelado conceptual.

En 2002 [2], IBM hizo una publicación acerca de un middleware llamado XPERANTO que resolvería el problema de la creación de vistas XML, éste no ha salido aún al mercado. Una de las cosas que se propone en este artículo es que esta herramienta tiene la habilidad de crear vistas XML sobre datos que se encuentran almacenados en bases de datos relacionales y tiene la facultad de crearlas de forma automática mapeando los datos desde un sistema de base de datos relacional a una vista XML por default. Por lo tanto, el usuario ingresa una consulta específica y XPERANTO internamente construye esa vista XML sobre la que fue creada inicialmente por default, usando el lenguaje de consultas XQuery. Otra de las características importantes que IBM resalta de esta herramienta es que permitirá

hacer consultas sobre dichas vistas.

Un trabajo más de IBM sobre vistas XML es XViews [6] el cual es un mecanismo que espera crear, borrar y consultar vistas XML. Esta solución pretende crear con el lenguaje XQuery vistas sobre documentos XML almacenados en repositorios nativos XML, agregando ciertas instrucciones. Como característica importante de esta herramienta se dice que aunque la vista no existe físicamente, una vez creada ésta el usuario podrá consultarla como si fuera un documento almacenado. Al diseñar XViews se pensó que al consultar una vista, se ejecutará primeramente la consulta de la definición de la vista y se almacenarán sus resultados en un archivo XML temporal, de tal forma que las consultas que hiciera el usuario serían ejecutadas sobre éste, esta técnica no es útil a menos que la vista a materializar sea pequeña, por lo que se propusieron otras optimizaciones a esta solución, las cuales tampoco han sido implementadas debido a que ésta herramienta aún no se ha puesto en producción.

Es importante mencionar que ninguno de estos trabajos de investigación que proponen dar solución al problema de creación de vistas sobre documentos XML han sido implementados hasta hoy en día, además de que en términos generales no se le ha prestado el suficiente interés a éste tema, por lo que las ventajas que proporciona la utilización de vistas esta siendo desaprovechada.

1.4 Organización de la tesis

La organización del presente trabajo es:

En el capítulo 1, se presenta de forma breve el problema a resolver, los objetivos, la relevancia y contribución, así como también los trabajos relacionados.

El capítulo 2, introduce los antecedentes del XML, los componentes de un documento XML explicados de forma breve e ilustrada mediante ejemplos, además de los lenguajes de consulta XQuery y XPath.

En el capítulo 3, se describe la relación del XML con las bases de datos, aquí también se define en que consisten las bases de datos habilitadas y las nativas, además de las propuestas que han sido hechas por diversos investigadores.

El capítulo 4, consiste en el análisis y diseño de la herramienta propuesta en este trabajo, por lo que se describen los casos de uso, la arquitectura del sistema, los diagramas de clases y de secuencia que le darán la funcionalidad a la misma.

En el capítulo 5, se describe como se implementó el mecanismo de creación de vistas, las características de la computadora en donde se realizaron las pruebas con el software, el algoritmo que da funcionalidad a la herramienta propuesta y los resultados obtenidos.

El capítulo 6, presenta las conclusiones y los trabajos futuros de esta investigación.

2

Fundamentos de XML

2.1 ¿Qué es XML?

El XML tuvo sus inicios en un lenguaje creado por IBM en los años 70's [3], al que inicialmente se llamo GML (Lenguaje de Marcas Generalizado), el cual fue normalizado por ISO en 1986, creando así el SGML (Lenguaje de Marcas Generalizado Estándar), que es un lenguaje descriptivo; éste ya ofrecía nombres para categorizar e identificar partes de un documento, siendo capaz de solucionar una gran cantidad de problemas, debido a que introdujo:

- El uso de etiquetas de apertura y cierre para que los navegadores pudieran interpretar y dar formato al texto incluido en ellas.
- La posibilidad de usar enlaces internos y externos a otros

documentos incrementando las posibilidades de navegación.

- El uso de las DTD's (Definición del tipo de documento).

El SGML era principalmente usado en el intercambio, gestión y publicación de documentos. Éste era flexible, sin embargo, su nivel de complejidad impidió su aprobación en diversos ambientes de aplicación incluyendo el World Wide Web. Fue así que surgió el estándar XML, que tenía como propósito recuperar el poder y la flexibilidad de SGML sin la mayor parte de su complejidad.

Se define al XML[8] como el Lenguaje de Marcas Extensible, desarrollado por el World Wide Web Consortium (W3C), se considera extensible porque una vez que los documentos XML son diseñados y puestos en producción, se pueden añadir nuevas etiquetas sin que por esta razón dejen de funcionar.

Hoy en día el XML es un estándar que permite estructurar la información en partes bien definidas y a su vez dichas partes se pueden integrar por otras partes, dando como resultado una estructura de árbol o grafo. La naturaleza del XML da la posibilidad de que pueda ser usado no sólo para aplicaciones en Internet si no que también en bases de datos relacionales u orientadas a objetos, hojas de cálculo, editores de texto, llamadas a procedimientos remotos, sistemas de correo de voz y muchas otras aplicaciones más.

Los documentos XML, están formados por etiquetas las

cuales indican estructura, por ejemplo: la cadena de texto contenida entre las etiquetas <persona> ... < /persona>, contiene información que describe a una persona. Esto significa que el XML, es un lenguaje auto–descriptivo, que contiene marcas que describen la forma en que el documento esta estructurado y que permiten conocer la relación de los elementos entre sí, considerándose el elemento la unidad básica de datos de todo documento XML.

Algunas de las ventajas con las que se cuenta al usar XML son [11]:

- XML es un lenguaje de meta-marcas, lo cual significa que no tiene un conjunto de etiquetas fijas, si no que deja a decisión de los usuarios de XML, inventarse los nombres de éstas de acuerdo a los elementos que necesiten.
- Da flexibilidad en los datos, porque cualquier información puede ser almacenada dentro de un documento XML y dichos datos siempre serán almacenados como cadenas de texto.
- Proporciona comunicación entre las páginas Web, debido a que la construcción de las páginas es genérica, además, se puede confiar en la definición de las etiquetas y la ubicación de las mismas para presentar los datos.
- Otorga estándares abiertos, porque como se mencionó anteriormente el XML es completamente flexible, ningu-

na compañía de software puede controlar y establecer qué etiquetas deben ser creadas, cuál es el significado y dónde deben aparecer. Sin embargo, a pesar de que XML es flexible también es muy estricto en cuanto a la gramática que un documento XML debe contener.

- Las marcas que contiene un documento XML, ayudan a conocer la semántica y organización dentro del mismo, por lo que en [11] se considera como un lenguaje de marcas estructural y semántico.

- El orden en el cual aparecen los datos en una página XML es irrelevante porque son únicamente datos, estos se pueden representar de diversas formas, del lado del cliente hasta se podría usar XSLT.

La utilización del estándar XML está en aumento, por lo que hoy se cuenta con herramientas que realizan diferentes funciones sobre los documentos XML, a continuación se mencionan algunas de ellas:

- Editores XML: XMLspy, oxygenXML, XmEdiL, Open XML Editor, XPontus XML Editor, GWD Text Editor English 3.2, etc.,

- API's para extraer la información de los documentos XML: SAX y DOM.

- Lenguajes de transformación de documentos: Hojas de estilo CSS y XSLT.

- Lenguajes de consulta en Documentos XML: XQuery, XUpdate y XPath.

2.2 Componentes de un documento XML

Los componentes de un documento XML pueden verse como una colección de nodos relacionados entre sí. Cada nodo puede ser una hoja o bien un nodo interior, el contenido de una hoja es un dato de tipo texto, a diferencia de los nodos interiores que pueden contener hojas o bien otros nodos interiores.

Considere el documento XML presentado en la figura 2.1.

```
<Ordenes>
        <orden id="r1">
                <producto>
                        <descrip>Cajeta envinada, 350 ml</descrip>
                        <precio>26.60</precio>
                </producto>
                <producto>
                        <descrip>Acondicionador Pantene, 790ml</descrip>
                        <precio>76.50</precio>
                </producto>
        </orden>
</Ordenes>
```

Figura 2.1: Documento XML que almacena ordenes de productos.

Si se representará la figura 2.1 mediante nodos, se observaría como en la figura 2.2.

La representación de árbol observada en la figura 2.2, indica que el nodo Ordenes está compuesto por un solo nodo hijo Orden, aunque cabe aclarar que el número de nodos hijos no tiene un límite. El nodo Orden, así mismo tiene un solo nodo hijo Producto, el cual a su vez tiene como nodos hijos a Descrip y Precio. Por lo tanto, cualquier nodo padre puede tener varios nodos hijos, pero un nodo hijo no puede tener más de un nodo padre. Todos los documentos XML tienen un nodo sin un padre, porque éste es el nodo que contiene al resto de los nodos, a éste se le conoce como nodo o elemento raíz del documento. Debido a que todo documento XML esta bien estructurado, únicamente contiene un nodo raíz y el resto de los nodos sólo tiene un padre, estos forman una estructura de datos llamada árbol, aunque también pueden formar un grafo [9].

2.2.1. Elementos

La unidad básica en un documento XML es el elemento, éste consiste en una cadena de texto encerrada entre los símbolos “<” y “>”, las etiquetas correspondientes a un elemento vienen por pares (apertura y cierre), como por ejemplo: <dataset> y </dataset>. A todo lo que está incluido entre la etiqueta de inicio y cierre del elemento se le llama contenido del elemento [8].

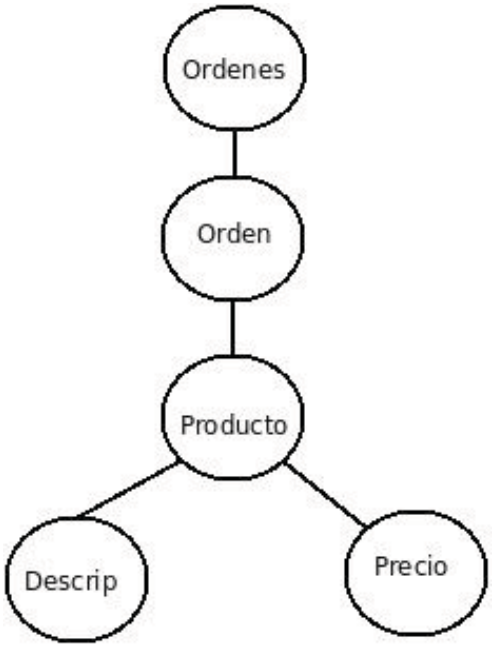

Figura 2.2: Nodos del documento XML presente en la figura 2.1

Se dice que un documento XML también puede ser un grafo (2.3), porque desde un nodo se puede hacer referencia a otro nodo presente en alguna otra rama del árbol que indicaría lo que en bases de datos se conoce como ligaduras de integridad.

En la figura 2.1, se identifican las etiquetas o marcas siguientes: Ordenes, Orden y producto, para que éstas sean correctas deben abrirse y cerrarse, en el orden inverso a como fueron abiertas. Como se mencionó anteriormente las etiquetas no es-

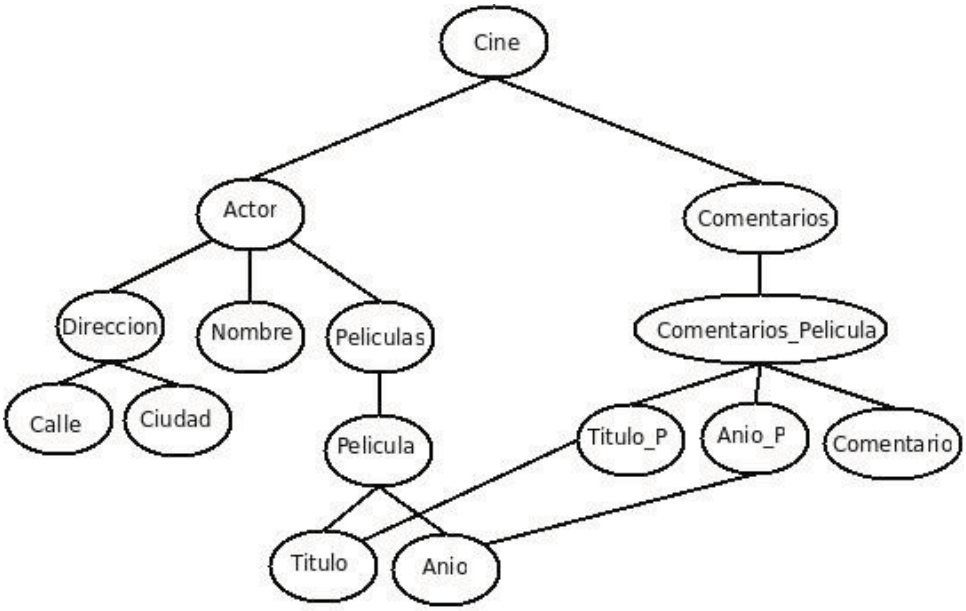

Figura 2.3: Documento XML con ligaduras de integridad en los nodos internos Titulo, Anio, Titulo_P y Anio_P.

tán predefinidas en el XML, sino que cada usuario debe definirlas como crea conveniente para su aplicación. En cada documento XML existe una relación entre los elementos y los elementos que estos contienen, de tal forma que: Orden es un subelemento de Ordenes, producto es un subelemento de Orden, Descrip y precio son subelementos de producto. Debido a que al tratar con XML, se habla de datos semiestructurados, la etiqueta de un elemento puede aparecer varias veces para representar una colección en un documento XML.

Todos los datos contenidos en los elementos del documento XML, son tratados como texto, es por esto que no es necesario encerrar dicho contenido entre comillas dobles.

2.2.2. Atributos

Los elementos pueden tener desde cero o más atributos [8], éstos se añaden a la etiqueta de apertura del elemento, el nombre del atributo se separa de su valor mediante un signo igual y dicho valor se escribe entre comillas dobles o simples. Siguiendo con el ejemplo en la figura 2.1, el elemento orden tiene un atributo llamado id con valor r1.

Considerando el siguiente ejemplo de la figura 2.4, se pueden notar que el elemento PREMIO contiene 2 atributos CATEGORIA y ANIO_PREMIO.

Los atributos se utilizan para describir a los elementos. Existe controversia al respecto de qué debe ser incluido en el documento XML como atributo y qué como elemento. Muchos autores opinan que se deben almacenar como elementos los datos del documento y los metadatos (datos acerca de los datos) como atributos, aunque no siempre es posible distinguir un dato de un metadato. Siguiendo con el ejemplo de la figura 2.1, se le llamaría dato a todo aquello que se guarda de las órdenes, como la descripción del producto, el precio del mismo y metadato serían aquellas propiedades o características que pertenece a un elemento en particular.

```
<PREMIOS>
        <PREMIO CATEGORIA="PREMIOS OSCAR" ANIO_PREMIO="2005">
                <NOMINACIONES>
                        <NOMINACION>Mejor diseño de vestuario</NOMINACION>
                        <RESULTADO>Nominada</RESULTADO>
                </NOMINACIONES>
        </PREMIO>
        <PREMIO CATEGORIA="GLOBOS DE ORO" ANIO_PREMIO="2005">
                <NOMINACIONES>
                        <NOMINACION>Mejor actor - Comedia o Musical</NOMINACION>
                        <RESULTADO>Nominada</RESULTADO>
                </NOMINACIONES>
        </PREMIO>
        <PREMIO CATEGORIA="PREMIOS BAFTA" ANIO_PREMIO="2005">
                <NOMINACIONES>
                        <NOMINACION>Mejor diseño de producción</NOMINACION>
                        <RESULTADO>Nominada</RESULTADO>
                </NOMINACIONES>
                <NOMINACIONES>
                        <NOMINACION>Mejor diseño de vestuario</NOMINACION>
                        <RESULTADO>Nominada</RESULTADO>
                </NOMINACIONES>
        </PREMIO>
</PREMIOS>
```

Figura 2.4: Documento XML que guarda los premios de la película “Charlie y la Fábrica de Chocolate”.

2.2.3. Comentarios

En ocasiones es necesario insertar comentarios en documentos XML, lo que significa que éstos son únicamente para fines informativos de la persona que está diseñando el documento y que son ignorados por el procesador de la información.

Los comentarios pueden ser introducidos en cualquier parte del documento, excepto dentro de las declaraciones de otros comentarios o de etiquetas. El formato que poseen los comentarios XML, inicia con la cadena "<! − −" y termina con "−− >". Por ejemplo:

<! − −− Inicia documento XML − − − >
<root>
...
...
</root>

2.2.4. Espacio de nombres

Diferentes documentos XML, pueden tener elementos con el mismo nombre pero diferente significado y esto es correcto, mientras ellos no sean combinados en un solo documento. Por tal razón surgieron los espacios de nombres, para identificar de forma única los elementos dentro de la estructura de un documento XML y así sus nombres tengan un ámbito más allá del

```
<COMENTARIOS_PELICULA>
        <TITULO_P>Charlie y la Fabrica de Chocolate</TITULO_P>
        <ANIO_P>2005</ANIO_P>
        <DIRECCION>Tim Burton</DIRECCION>
        <COMENTARIO>Orgullo, gula, pereza, avaricia y bondad,
                una moraleja escrita por el talentoso autor
                Roald Dahl, de cuyas obras adaptadas al séptimo arte son:
                'Las brujas', 'Matilda', ' James y el melocotón
                gigante' y 'Charlie y la fábrica de Chocolate',
                siendo esta última la      más conocida en el
                medio literario pero no precisamente en el medio
                                cinematográfico.
        </COMENTARIO>
</COMENTARIOS_PELICULA>
```

Figura 2.5: Documento XML que almacena comentarios de películas.

documento que los contiene. Para ejemplificar este tema véase los siguientes documentos.

La figura 2.5, presenta un documento XML que almacena comentarios de películas. El elemento raíz del documento es COMENTARIOS_PELICULA y contiene a los elementos: TITULO_P que almacena el nombre de la película de la cual se está haciendo el comentario, ANIO_P que guarda el año en el que fue producida la película, DIRECCION que guarda el director de la película y COMENTARIO que almacena el comentario de la película otorgado por críticos de cine.

La siguiente figura (2.6), muestra un documento XML que

incluye los actores y las películas en las que ellos han actuado desde el año 2001. En la estructura del documento se pueden observar elementos tales como: NOMBRE que almacena el nombre del actor, DIRECCION que guarda la o las direcciones del actor y éste a su vez ésta compuesto por los elementos CALLE y CIUDAD. El elemento PELICULAS está compuesto por el elemento PELICULA, que incluye los elementos: TITULO que almacena el título de la película y ANIO que guarda el año en el que la película fue producida.

Supongamos que una compañía maneja los documentos de las figuras 2.5 y 2.6, desea combinar ambos documentos en uno solo, entonces las etiquetas DIRECCION estarían duplicadas para este nuevo documento; además para la figura 2.5, el elemento DIRECCION se refiere a la dirección del actor y para la figura 2.6 se refiere al director de la película. Este tipo de conflictos son resueltos por los espacios de nombre, los cuales consisten en asignar un prefijo diferente a cada documento XML, cada prefijo será asignado a un URI (Uniform Resource Identifier). Los URI se encargan de identificar de forma única un recurso, el cual puede ser un servicio, página, documento, dirección de correo electrónico, etc. En la figura 2.7, se presenta el documento combinado utilizando los beneficios de los espacios de nombres.

Si se observa la figura 2.7, es posible notar que un editor XML, reconocerá a <c:DIRECCION> como un elemento distinto de <p:DIRECCION>, esto se logra por los prefijos c y

```
<ACTOR>
        <NOMBRE>JOHNNY DEPP</NOMBRE>
        <DIRECCION>
                <CALLE>500 South Sepulveda Blvd #500 </CALLE>
                <CIUDAD>Los Angeles, USA</CIUDAD>
        </DIRECCION>
        <DIRECCION>
                <CALLE>United Talent Agency 9560 Wilshire Blvd</CALLE>
                <CIUDAD>Beverly Hills, USA</CIUDAD>
        </DIRECCION>
        <PELICULAS>
                <PELICULA>
                        <TITULO>Soplar</TITULO>
                        <ANIO>2001</ANIO>
                </PELICULA>
                <PELICULA>
                        <TITULO>Los Piratas del Caribe: El curso del Perla Negra</TITULO>
                        <ANIO>2003</ANIO>
                </PELICULA>
                <PELICULA>
                        <TITULO>La Ventana Secreta</TITULO>
                        <ANIO>2004</ANIO>
                </PELICULA>
                <PELICULA>
                        <TITULO>Charlie y la Fabrica de Chocolate</TITULO>
                        <ANIO>2005</ANIO>
                </PELICULA>
        </PELICULAS>
</ACTOR>
```

Figura 2.6: Documento XML, que almacena actores y las películas en las que estos han participado desde el año 2001.

p, los cuales harán que elementos con el mismo nombre sean tratados diferente en el mismo documento.

2.2.5. Instrucciones de procesamiento

XML proporciona instrucciones de procesamiento, mediante las cuales las aplicaciones que leen el documento saben qué hacer con el mismo. Toda instrucción de procesamiento en XML inicia con <? y termina con ?>.

Entonces para iniciar la apertura de esta instrucción tenemos <? seguido de un destino que puede ser el nombre de la aplicación que usará esta instrucción o el nombre para esta instrucción de procesamiento, por ejemplo: <?xml–stylesheet href=“empresa.css” type=“text/css”?>

En este caso *xml–stylesheet* se usa para indicar al navegador que deben aplicar la hoja de estilo *empresa.css* antes de presentar este documento al usuario. Este tipo de instrucciones pueden ser incluidas en cualquier parte del documento XML, incluso antes o después del nodo raíz.

2.3 Estructura de un documento XML

Según [11], para que un documento XML se encuentre bien estructurado debe cumplir con las siguientes reglas:

```
<?xml version= "1.0"?>
<c:Comentarios_p xmlns:c ="http://www.w3c.org/Comentarios"
                 xmlns:p= "http://www.w3c.org/Peliculas">
<c:COMENTARIOS_PELICULA>
        <c:TITULO_P>Charlie y la Fabrica de Chocolate</c:TITULO_P>
        <c:ANIO_P>2005</c:ANIO_P>
        <c:DIRECCION>Tim Burton</c:DIRECCION>
        <c:COMENTARIO>Orgullo, gula, pereza, avaricia y bondad, una moraleja escrita por el
        talentoso autor Roald Dahl, de cuyas obras adaptadas al séptimo arte son: &quot; Las
        brujas&quot;, &quot;Matilda&quot;, &quot;James y el melocotón gigante&quot; y
        &quot;Charlie y la fábrica de chocolate&quot;, siendo esta última la más conocida en el medio
        literario pero no precisamente en el medio cinematográfico.</c:COMENTARIO>
        </c:COMENTARIOS_PELICULA>
<p:ACTOR>
        <p:NOMBRE>JOHNNY DEPP</p:NOMBRE>
        <p:DIRECCION>
                <p:CALLE>500 South Sepulveda Blvd #500 </p:CALLE>
                <p:CIUDAD>Los Angeles, USA</p:CIUDAD>
        </p:DIRECCION>
        <p:DIRECCION>
                <p:CALLE>United Talent Agency 9560 Wilshire Blvd</p:CALLE>
                <p:CIUDAD>Beverly Hills, USA</p:CIUDAD>
        </p:DIRECCION>
        <p:PELICULAS>
                <p:PELICULA>
                        .....
                </p:PELICULA>
        </p:PELICULAS>
</p:ACTOR>
```

Figura 2.7: Documento XML que fusiona los documentos XML en la Figura 2.5 y 2.6, utilizando espacios de nombres.

1. Cada etiqueta de apertura debe tener su etiqueta de cierre correspondiente.

2. Los elementos en el documento, pueden contener subelementos pero no superponerse, por lo tanto el siguiente documento es incorrecto porque rompe esta regla:
   <persona>
   <nombre>
   < /nombre>
   <apellido>
   < /persona>
   < /apellido>

3. En todo documento XML, únicamente debe existir un elemento raíz.

4. Los valores de los atributos deben ser escritos entre comillas.

5. Los atributos para cada elemento deben ser únicos, es decir, un elemento no puede contener dos atributos con el mismo nombre.

6. Si en el documento existen comentarios o instrucciones de procesamiento, éstas no pueden formar parte de los elementos.

7. Los caracteres < o $ no pueden ser parte de los datos en un atributo o elemento.

Existen muchas formas de analizar si la estructura de un documento XML es correcta, una de ellas es escribir dichos documentos en editores de texto, éstos se adaptan al estándar e informarán de los errores presentes en el documento. Otra forma segura de verificar la estructura de un documento XML, es escribir una DTD (Document Type Definition) o un esquema y confrontar el documento elaborado contra éstos, para verificar su validez; debido a que estos documentos contienen las restricciones a las cuales se debe acatar cada documento XML creado. La DTD o el esquema indican qué elementos pueden aparecer en el documento, qué atributos y donde deben aparecer éstos, además de definir para el caso del esquema qué tipo de dato va almacenar cierto elemento o atributo.

2.4 Documentos XML bien–formados

Para que un documento XML sea correcto tiene que ser bien formado y válido. El que un documento XML este bien formado se refiere a la estructura sintáctica, es decir, que los elementos, atributos y comentarios estén especificados como el estándar XML lo determina. En general para que un documento este bien formado se deben cumplir las reglas establecidas en el apartado 2.3.

2.5 Validez de un documento XML

La validez se refiere a que cada aplicación XML, debe verificar cuál es la relación que debe existir entre cada uno de los elementos que aparezcan en el documento, dicha relación debe especificarse en un documento externo que bien puede ser una DTD o un Esquema XML, porque éstos son los que van a indicar la estructura que tendrá internamente cada documento XML creado. Cabe aclarar que para el caso donde se incluyen ligaduras de integridad la DTD o el esquema aún son medios débiles para verificar que estas se cumplan.

2.6 Lenguajes de consulta en documentos XML

Un lenguaje de consulta es utilizado para consultar bases de datos o sistemas de información. Hoy en día existen herramientas para la extracción de datos desde documentos XML, como lo son SAX y DOM[10]. A continuación se explica brevemente cual es la funcionalidad de cada uno de estos:

- SAX: es un analizador orientado a eventos, lo cual significa que al analizar un documento XML, cada vez que se encuentra con un elemento, ya sea una etiqueta de apertura o cierre produce un evento, permitiendo al programador redefinir el evento que desee controlar. La im-

portancia de la extracción de información con SAX radica en que el programador puede comprobar el valor que tiene cada etiqueta y actuar en consecuencia. Construir un analizador SAX es simple, además utiliza muy poca memoria debido a que solo carga la sección de datos XML con la que está trabajando. Tiene la desventaja de que depende directamente de la estructura del documento, de tal forma de que si ésta cambia, se tendría que modificar el código, por lo tanto el código tampoco es portable para utilizarse de un documento XML a otro con diferente estructura.

- DOM: Para extraer información desde un documento XML éste debe almacenar el conjunto de datos en memoria en forma de árbol, de tal forma que cualquier operación sobre el conjunto se realiza en la memoria. Es más rápido que SAX precisamente porque almacena los datos en memoria y debido a que se dispone de toda la estructura de los datos se pueden hacer búsquedas hacia atrás o hacia adelante. Tiene la desventaja de que extraer los datos del documento es más complejo que SAX ya que se tiene que escribir el recorrido del árbol.

Como se puede notar, éstas herramientas son útiles cuando la cantidad de información es pequeña, sin embargo hoy en día la utilización de XML está en aumento, por lo que SAX y DOM ya no se consideran eficientes para manejar colecciones grandes

y complejas de información. Por tal razón surgieron los lenguajes de consulta sobre documentos XML, como son XPath y XQuery.

2.6.1. XPath

Para procesar un documento XML [4], se necesita acceder a cada una de las partes que lo componen y de esta forma poder extraer la información de interés, para este fin se utiliza XPath.

XPath es un lenguaje de consulta desarrollado por el equipo de trabajo del W3C, el cual construye expresiones que recorren y procesan un documento XML utilizando la estructura jerárquica del XML. Sobre éste se han creado otras herramientas también utilizadas en el tratamiento de documentos XML, como son: xLink, xPointer y XQuery.

Para procesar un documento XML, XPath toma en cuenta que dicho documento tiene estructura jerárquica por lo que inicialmente un analizador lo recorre y forma un árbol de nodos. Éste árbol contiene un nodo raíz y desde él cuelgan otra serie de nodos que se extienden hacia abajo hasta llegar a los nodos hoja, los cuales pueden contener solo texto, comentarios, instrucciones de proceso o incluso estar vacíos y sólo contener atributos. Un caso especial de nodo, son los que pertenecen a los atributos debido a que un elemento puede contener el número de atributos que desee, así que por cada uno de éstos se le creará

un nodo atributo, aunque estos no son considerados como hijos sino como etiquetas del nodo elemento.

Debido a que XPath [4] puede contener la cantidad de elementos y atributos que desee, para poder ubicarlos dentro de un documento XML son necesarios los caminos de localización, que son aquellas expresiones con las cuales se le indicará al procesador de XPath las rutas donde se encuentran los elementos deseados, éstos pueden ser expresados usando la sintaxis abreviada o extendida. XPath siembre evalúa una expresión con respecto a su contexto, por tal razón se dice que un camino de localización extrae nodos a partir del nodo de contexto o contextual. A continuación se presentan algunos ejemplos de caminos de localización que utilizan la sintaxis no abreviada:

- child::bib selecciona los elementos bib hijos del nodo contextual.
- child::* selecciona todos los elementos hijos del nodo contextual.
- child::text() selecciona todos los nodos texto hijos del nodo contextual.
- child::node() selecciona todos los hijos del nodo contextual, cualquiera que sea su tipo de nodo.

- child::bib/child::libro[position()=1] selecciona el primer hijo del nodo libro del elemento raíz bib.
- child::bib[position()=last()] selecciona el último hijo bib del nodo contextual.

Existen dos tipos de caminos de localización: caminos de localización relativos y caminos de localización absolutos. Los caminos relativos se definen por una secuencia de uno o más pasos de localización separados por "/". Un paso de localización contiene tres partes:

- Un eje, que determina la relación jerárquica entre los nodos seleccionados por el paso de localización y el nodo contextual.
- Una prueba de nodo, que define el tipo de nodo y el nombre de los nodos seleccionados por el paso de localización.
- Predicados, estos pueden ser cero o más y usan expresiones arbitrarias para refinar aún más el conjunto de nodos seleccionado por el paso de localización.

La sintaxis del paso de localización es el nombre del eje y prueba de nodo separados por dos puntos "::", seguido de cero o más expresiones, cada una entre corchetes. Por ejemplo, en child::bib[position()=1] , child es el nombre del eje, bib es la

prueba de nodo y [position()=1] es un predicado. Dichos pasos se procesan de izquierda a derecha y cada paso va seleccionando un conjunto de nodos relativos a un nodo contextual; de tal forma que cada nodo de ese conjunto se va a ir usando como nodo contextual para el siguiente paso. La unión del conjunto de nodos extraídos por cada paso forman el resultado. Por ejemplo: child::bib, es un camino relativo porque está compuesto por pasos. Un camino de localización absoluto consiste en "/" seguido opcionalmente por un camino de localización relativo. Una "/" por si misma selecciona el nodo raíz del documento que contiene al nodo contextual, si ésta es seguida por un camino de localización relativo, éste selecciona el conjunto de nodos correspondientes al nodo contextual desde el nodo raíz del documento. Un ejemplo de éste tipo de camino es: /libro/titulo, ésta expresión no cuenta con caminos de localización relativo y obtiene los nodos titulo hijos del nodo libro ubicados en cualquier posición del documento. La siguiente expresión si contiene un camino de localización relativo: /bib/child::libro.

XPath es muy fácil de entender, en los siguientes ejemplos se muestran algunas de las expresiones más comunes, que el lector debe conocer para comprender la funcionalidad del lenguaje.

- /libro/titulo
  El XPath anterior selecciona todos los nodos titulo que son hijos del nodo libro, por lo tanto, el resultado mostrará

todos los nodos titulo que cumplan con esta condición.

- //bib
  Ésta expresión selecciona todos los nodos bib que aparezcan en cualquier posición en el árbol XML, la doble barra "//" indica a cualquier profundidad.

- count(//libro)
  Al igual que SQL, XPath cuenta con algunas funciones, como por ejemplo: count(), que para este caso devuelve el número de nodos libro que aparecen en el documento en cualquier posición.

- //libro/@paginas
  Este XPath selecciona todos los atributos paginas de los nodos libro ubicados en cualquier posición del documento XML.

- /child::bib/child::libro/child::titulo
  Este camino selecciona todos los nodos título, hijos del nodo libro que a su vez es hijo del nodo raíz bib.

XPath además soporta el uso de predicados, es decir, en la consulta se pueden utilizar condiciones booleanas para seleccionar un subconjunto de los nodos extraídos. De igual forma que son usadas en cualquier lenguaje de programación, la expresión booleana se evalúa y si esta es cierta, el nodo se selecciona y se devuelve, en caso contrario el nodo se descarta.

A continuación se presenta ejemplos donde se manejan predicados:

- /bib/libro/autor[5]
  Ésta consulta obtiene sólo el quinto nodo autor que aparezca en cada nodo libro, que a su vez es hijo de un nodo bib.

- //libro[autor = “Adams”]
  Este camino de localización devuelve todos los nodos libro que aparezcan en el documento XML en cualquier posición y que tengan un nodo hijo autor con el valor “Adams” .Los corchetes son un filtro para seleccionar únicamente los resultados que cumplan con la condición dada.

- //libro[@año > 1990]
  Ésta consulta obtiene todos los nodos libro que se encuentran en cualquier posición del documento, que tengan un atributo “año” con un valor mayor a 1990. El @ indica que el nodo no es un hijo, sino un atributo.

- //libro[@paginas]
  La expresión obtiene todos los nodos libro ubicados en cualquier posición del documento, que contengan un atributo paginas.

- /child::bib/child:libro[attribute::año=2000]

Ésta consulta selecciona todos los nodos libro cuyo atributo año es igual a 2000, hijos del nodo raíz bib.

- /child::bib/child::libro[attribute::paginas=500]/child::titulo
  La expresión selecciona todos los nodos títulos, hijos del nodo libro cuyo atributo paginas es igual a 500 y éste a su vez es hijo del nodo raíz bib.

- //child::bib/child::libro[@paginas=500 or @paginas=600]
  Este XPath selecciona todos los nodos libro que tienen 500 o 600 páginas, hijos del nodo raíz bib.

- /bib/libro[number(@paginas) = 1000]
  Ésta expresión selecciona todos los nodos libro cuyo atributo paginas (que primero es convertido a numero con la función number) es igual a 100, hijos del nodo raíz bib.

XPath al igual que el SQL cuenta con funciones ya predefinidas las cuales incrementan la funcionalidad del lenguaje.

2.6.2. XQuery

De acuerdo al incremento en la cantidad de información que es almacenada, intercambiada y presentada usando XML, la habilidad para consultar fuentes de datos XML llega a ser muy importante. El XML tiene la flexibilidad de representar diferentes tipos de información desde diversas fuentes. Para explotar esta

flexibilidad, un lenguaje de consulta XML debe proveer componentes para obtener e interpretar la información desde diversas fuentes. Como resultado de esta necesidad surge XQuery.

XQuery es un lenguaje de consulta que extrae información desde colecciones XML. éste lenguaje fue desarrollado por el consorcio internacional W3C y forma parte de sus recomendaciones para la World Wide Web desde el 23 de enero del 2007 [5].

XQuery extrae y manipula toda aquella información que se encuentra en formato XML, dicha información puede estar almacenada en cualquier fuente de datos como puede ser: bases de datos habilitadas o bases de datos nativas (ambas tratadas en el siguiente capítulo). XQuery es un lenguaje de programación funcional que esta compuesto en su totalidad de expresiones. Para ejecutar una consulta, la expresión dentro del cuerpo de ésta se evalúa y regresa el resultado obtenido.

Este lenguaje utiliza XPath para acceder a las partes del documento donde se encuentra la información que desea extraer. Además utiliza las expresiones conocidas como FLWOR, denominadas así por las iniciales de las cláusulas que cualquier consulta XQuery puede tener, las cuales son las siguientes [10]:

- *R*eturn: Construye el resultado de una consulta después de haber sido filtrada por la cláusulas where y ordenada por la cláusula order by.

- *F*or: Con esta cláusula, al aplicar una expresión escrita en

XPath sobre un documento XML, las tuplas resultantes son ligadas a una o más variables.

Ejemplos:

- for $b in doc (“libros.xml”)//bib/libro
  La consulta anterior vinculará a la variable $b las tuplas extraídas por la expresión XPath //bib/libro del documento libros.xml.
- for $b in doc (“libros.xml”)//bib/libro/titulo, $c in doc (“comentarios.xml”)//Comentario/titulo
  Ésta consulta además de vincular a la variable $b las tuplas extraídas de //bib/libro, también extrae en la variable $c el XPath //Comentario/titulo del documento XML comentarios.xml. Nótese que en la cláusula for, cada una de sus sentencias va separada por una “,”.
- for $titulos in doc (“libros.xml”)//bib/libro/titulo
  return
  <titulos> { titulos } <titulos>
  Ésta expresión obtiene como resultado los títulos de libros encontrados en el documento libros.xml. La cláusula for tiene la particularidad de que cada tupla resultante es vinculada a su correspondiente etique-

ta en el return, de tal forma que el resultado se verá así:
<titulos>
<titulo>“XML Databases”</titulo>
</titulos>
<titulos>
<titulo>“Databases Systems”</titulo>
</titulos>
...Como se observa a cada tupla resultante le corresponde una etiqueta <titulos></titulos>.

- *L*et: Vincula a una sola etiqueta el resultado completo de una expresión. Dicha expresión puede contener tuplas que fueron previamente generadas por la cláusula for.
  Ejemplos:

  - let $titulos := doc (“libros.xml”)//bib/libro/titulo
    return
    <titulos> { titulos } <titulos>
    Ésta consulta selecciona los títulos de libros encontrados en el documento libros.xml. La cláusula let hace que cada tupla resultante sea vinculada a una sola etiqueta <titulos>, por lo que el resultado se verá así:
    <titulos>

<titulo>“XML Databases”</titulo>
<titulo>Databases Systems</titulo>
. . .
</titulos>

- *W*here: Filtra las tuplas vinculadas a una variable eliminando todos los valores que no cumplan con la condición dada.
  Ejemplos:

  - for $titulos in doc (“libros.xml”)//bib/libro/titulo
    where $titulos/titulo = “XML Databases”
    return
    <libro> { titulos } <libro>
    El filtro aplicado en la cláusula where especifica que únicamente se desea extraer el libro con título “XML Databases”, por lo tanto, el resultado sería:
    <libro>
    <titulo>“XML Databases”</titulo>
    </libro>

- *O*rder By: Esta cláusula ordena las tuplas de acuerdo a un criterio dado.
  Ejemplos:

- for $libro1 in doc(“libros.xml”)//libro
  let *c* :=libro1//autor
  where count($c) > 5
  order by $libro1/titulo
  return $libro1/titulo
  En ésta consulta se observa la utilización de las 5 cláusulas. Lo que hace esta consulta es extraer la información desde el XPath //libro del documento libros.xml, obteniendo en el let los autores y almacenándolos en la variable $c, el where filtra aquellos libros con más de 5 autores y order by, ordena las tuplas resultantes por título, devolviendo como resultado únicamente los títulos. De ésta consulta no se obtienen resultados, ya que los libros almacenados no cumplen con esta condición.

XQuery es flexible por lo que una consulta no tiene que iniciar forzosamente con una cláusula for o let, si no que también puede formularse de la siguiente forma:

```
<nodo_raiz>
    {
        for $titulo1 in doc(“libros.xml”)/bib/libro
        where $titulo1/editorial = “Wrox” and $titulo1/@año >
2004
        return
```

```
        <libro año="{ $titulo1/@año }">
            { $titulo1/titulo }
                    </libro>
    }
</nodo_raiz>
```

Ésta consulta obtiene el nombre y el año de publicación de todos los libros cuya editorial sea Wrox y hayan sido publicados desde el año 2005.

Además de las cláusulas anteriores XQuery ofrece al igual que el SQL funciones adicionales, tales como: Para ejemplificar

Tipo de Operadores	**Operadores**
Matemáticos	+, -, *, div, idiv, mod.
Comparación	!=, =, <, >, <=, >=, not()
Secuencia	union (|), intersect, except
Redondeo	floor(), ceiling(), round().
De Agrupación	count, min(), max(), avg(), sum()
Funciones de cadena	concat(), string-lenght(), starts–with(), ends–with(), substring()
Uso general	upper-case(), lower-case(), string(), distinct–values(), empty(), exists()

Cuadro 2.1: Funciones de XQuery

algunas de éstas funciones se presentan los siguientes ejemplos:

- for $titulo in doc(“libro.xml”)//libro
  where some $titulo in $titulo//parraf
  satisfies (contains($titulo,“XML”) AND contains($titulo,-
  “Databases”))
  return <TITULO>$titulo/title </TITULO>
  Ésta expresión obtiene los títulos de los libros que contengan las palabras “XML” y “Databases” en el mismo párrafo. En caso de que el libro contenga más de un párrafo, se tomará como necesario que aparezca al menos uno de ellos, para esto se usa la función “some”.

- for $titulo1 in doc(“libros.xml”)//libro
  where $titulo1/titulo = “XML Databases”
  return
  <libro>
  { $titulo1/@* }
  { $titulo1/* except $titulo1/reseña }
  </libro>
  Ésta consulta utiliza el operador except, éste recibe conjuntos de nodos como operandos y regresa un conjunto que contiene todos aquellos nodos del primer operando que no aparezcan en el segundo, por lo que la consulta obtiene un nodo libro cuyo título es “XML Databases” con todos sus nodos hijos excepto el nodo reseña.

- <libros>
  {
  for $libro1 in doc("libros.xml")//libro,
  $comen in doc("comentarios.xml")
  where $libro1/titulo = $comen/titulo
  return
  <PRECIOS>
  { $libro1/titulo }
  <precio_lib>{ $comen/precio/text() }
  </precio_lib>
  <precio_com>{ $libro1/precio/text() }
  </precio_com>
  </PRECIOS>
  }
  </libros>
  La sentencia XQuery anterior obtiene el precio de cada libro guardado en el documento "libros.xml" y el precio que tiene ese mismo libro en el documento "comentarios.xml".

- <bib>
  {
  for $lib1 in doc("libros.xml")//libro,
  $lib2 in doc("libros.xml")//libro
  let $aut1 := for $a in $lib1/autor
  order by $a/apellidos, $a/nombre

```
return $a
let $aut2 := for $a in $lib2/autor
order by $a/apellidos, $a/nombre
return $a
where $lib1 << $lib2
and not($lib1/titulo = $lib2/titulo)
and deep-equal($aut1, $aut2)
return
<libros>
{ $lib1/titulo }
{ $lib2/titulo }
</libros>
}
</bib>
```

Esta consulta muestra los pares de títulos que sean distintos pero tengan el mismo autor o grupo de autores. Es importante aclarar que el orden de aparición de los autores puede variar de un libro a otro.

3

XML y Bases de datos

3.1 Las Bases de datos y su importancia

Una base de datos es una colección de datos interrelacionados entre sí, no redundantes que contiene la información que una organización desea administrar. Se le conoce como Sistema Gestor de Bases de Datos(SGBD) a la colección de datos interrelacionados y de programas que se utilizan para administrar dichos datos.

En un mundo tan competitivo, como el de nuestros días la administración de la información es algo que hace la diferencia y marca ventajas entre las empresas. Las bases de datos por sí mismas son importantes, si éstas no existieran y los archivos cumplieran esa función de almacenamiento no habría forma de mantener la información organizada, protegida contra accesos no autorizados, ni mucho menos podría accederse de forma ráp-

ida y eficiente.

Entre las aplicaciones más relevantes en las cuáles se utilizan bases de datos se pueden mencionar:

- Banca: para almacenar datos de clientes, préstamos, cuentas, etc..
- Líneas aéreas: para guardar datos de los viajes, reservas de vuelos, de hecho ésta es una de las primeras aplicaciones en utilizar bases de datos.
- Universidades: para almacenar información de las materias, profesores, cursos, alumnos, etc.
- Telecomunicaciones: se almacena información de las redes de comunicación, de las llamadas realizadas, datos para generar facturas, etc...
- Ventas: para guardar información de clientes, productos y compras.
- Finanzas: para almacenar datos acerca de ventas y compras de documentos financieros, operaciones de la bolsa, bonos, etc.

3.2 Funcionamiento de las bases de datos

Cuando se habla de almacenamiento de información surge el tema de las bases de datos. Las relacionales son las más utilizadas hoy en día, por lo que éstas pueden ser encontradas en la mayoría de las organizaciones, debido a que proporcionan el acceso a una gran cantidad de información y a un gran número de usuarios al mismo tiempo. Entre las funcionalidades que éstas otorgan se encuentran: la rapidez de acceso, consistencia e integridad en la información y mecanismos de seguridad implementados por ellas mismas.

Todos los aspectos que se han mencionado en el párrafo anterior con relación a las bases de datos y aunados a todas las ventajas ya comentadas que otorga el estándar XML, permite que al usar ambas tecnologías juntas se obtenga una poderosa herramienta para el almacenamiento y administración de la información.

3.3 ¿Puede considerarse un documento XML una base de datos?

El XML posee varias características que podemos encontrar en una base de datos: almacenamiento de documentos, esquemas, lenguajes de consulta, interfaces de programación, pero

también carece de otras cosas como: almacenamiento eficiente, seguridad, transacciones e integridad de datos, índices, acceso multiusuario, triggers, vistas, entre otros.

Según [14], un documento XML es una base de datos sólo en el sentido estricto del concepto, debido a que éste es una colección de datos. Por tal razón un documento XML tiene ventajas al ser una "base de datos", por ejemplo: es portable porque es universalmente entendido, se describe a sí mismo ya que contiene datos y metadatos que describen la estructura de los datos en el documento XML, además posee una estructura de árbol o de grafo que describe los datos que lo conforman. Sin embargo, también tiene desventajas como el acceso lento a los datos debido a que se tiene que hacer un escaneo y análisis completo a lo que podría ser un archivo de texto enorme.

3.4 XML y las Bases de Datos

En 3.2, se planteó la idea de que las bases de datos y el XML pueden ser tecnologías complementarías, porque al trabajar juntas ofrecen logros mayores que los méritos obtenidos individualmente, porque las bases de datos aportan administración y características de seguridad, además de que aseguran que si múltiples usuarios se conectan al mismo recurso, la integridad puede ser conservada gracias al mecanismo de bloqueos. Por otra parte, el XML como texto plano puede enviarse a otras

redes y no importa si éstas manejan diferentes plataformas, de tal manera que el XML otorga a las bases de datos flexibilidad en el intercambio de información.

Cuando se decide almacenar documentos XML en bases de datos, por las razones expuestas anteriormente, se tienen 2 opciones que serán exploradas a continuación.

3.5 Bases de datos habilitadas XML

Como se mencionó anteriormente, debido al incremento de la información almacenada en formato XML, surge la necesidad de guardar estos documentos en bases de datos, pero ¿Qué tipo de bases de datos? Actualmente las bases de datos relacionales cuentan con soporte para almacenar documentos XML, entre las principales se puede mencionar: Microsoft SQL Server, Oracle Oracle8i y DB2, Sybase ASE 12.5, el único inconveniente al respecto es que los siguen almacenando en formato relacional, esto es, guardan los documentos en forma tabular o en el mejor de los casos guardan el documento completo en formato Binary Large Object (BLOB), una de las principales características que proporcionan es que los resultados de las consultas los presentan en formato XML, por tal razón se les llama “Bases de Datos Habilitadas XML”. Esto ha dado pie al surgimiento de una nueva generación de bases de datos, que son muy diferentes a las relacionales, porque los documentos

son almacenados en fragmentos, dentro de tablas y para recuperarse en algunos casos se usan las funciones SQL o una utilidad llamada XML SQL (XSU)[15]. Concluyendo, éstas bases de datos aceptan documentos XML como entrada y desglosan la información de esos documentos en su correspondiente esquema relacional o de objetos.

En la figura 3.1, se presenta un documento XML el cual será almacenado en una base de datos relacional habilitada, por lo que será mapeado a su correspondiente forma tabular:

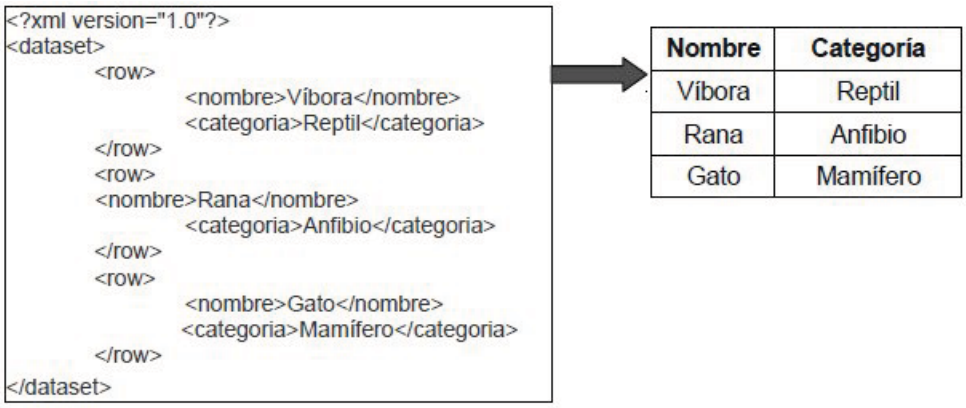

Figura 3.1: Mapeo de un documento XML en una tabla del modelo relacional.

3.6 Bases de datos nativas XML.

Si un documento XML puede ser considerado una base de datos, entonces, puede contener una cantidad enorme de información toda almacenada en un solo lugar, esto puede ocasionar muchos problemas cuando muchos usuarios necesiten acceder a esta gran cantidad de datos. Por estas razones surgieron las bases de datos nativas XML (NXD), que son aquellas en las cuales los documentos XML son almacenados, indexados y recuperados en su formato original, esto significa que los documentos para almacenarse no tienen que ser divididos en fragmentos, en consecuencia el formato de todos aquellos nodos que sean parte del documento XML, será preservado [15].

Una base de datos nativa:

- Define un modelo lógico (DTD, Esquema XML, . . .) para los documentos XML, almacena y obtiene dichos documentos de acuerdo a este. El modelo deberá incluir elementos, atributos, PCDATA (Parsed Character Data) y el orden jerárquico del documento.

- La unidad básica de almacenamiento es un documento XML.

- No necesita un modelo de almacenamiento físico en particular.

La mayoría de las bases de datos XML, usan un modelo lógico para agrupar documentos llamados colecciones. Actualmente las NXD soportan al menos un lenguaje de consulta, como mínimo éstas soportan XPath para consultar documentos o colecciones de documentos, pero no todas soportan XQuery que incluye XPath para la selección de los nodos, además de una serie de cláusulas llamadas "FLWOR" para ejecutar consultas más complejas.

Existen diversas NXD comerciales, como por ejemplo: eXcelon Xis, Tamino, dbXML[5] o GoXML; y de código abierto se puede mencionar a eXist y Apache Xindice, entre las principales. Todas éstas brindan diferentes características, sin embargo, la mayoría de ellas ofrece:

- Procesamiento de datos.
- Almacenamiento.
- Búsquedas.

La figura 3.1, presenta un documento XML que guarda nombres de animales y su correspondiente categoría. Si este documento se almacenará en una base de datos nativa, el formato XML se conservaría.

En la sección 3.6.1 se explicará brevemente acerca de una base de datos nativa llamada eXist, que es una de las herramientas utilizadas para la creación del mecanismo de vistas del cuál se trata este trabajo de investigación.

```
<?xml version="1.0"?>
<dataset>
        <row>
                <nombre>Víbora</nombre>
                <categoria>Réptil</categoria>
        </row>
        <row>
                <nombre>Rana</nombre>
                <categoria>Anfíbio</categoria>
        </row>
        <row>
                <nombre>Gato</nombre>
                <categoria>Mamífero</categoria>
        </row>
</dataset>
```

Figura 3.2: Formato en el que se guarda en una base de datos NXD un documento XML que almacena animales y su categoría.

3.6.1. eXist

eXist es sistema de bases de datos nativa XML [17], de código abierto construida en lenguaje java usando tecnología XML. fue creada en el año 2000 por Wolfgang Meier, en Septiembre de 2006 salieron al mercado las versiones 1.0 y 1.1, actualmente se encuentra ya liberada la versión 1.2.6.

Debido a que eXist es una base de datos nativa XML, permite el almacenamiento de documentos XML en su formato original. Además ofrece ciertas ventajas en la organización de la información, porque todos los documentos almacenados de un mismo tipo pueden ser agrupados en colecciones organizadas

jerárquicamente[18]. Dichas colecciones también permiten guardar esquemas o DTD's, para que al momento de que el usuario quiera almacenar un documento XML dentro de ellas, pueda ser confrontado contra éstos y si el documento no es válido se muestra el error al usuario, en caso contrario se almacena el documento exitosamente.

Esta NXD cuenta con una sección de administrador desde la cual se pueden crear, borrar, consultar y modificar colecciones, así como también permite agregar a éstas los documentos XML que se deseen. Para ingresar a esta sección el usuario debe ser administrador y basta con ingresar su contraseña para que pueda navegar entre las colecciones, administrar usuarios y verificar el estado del sistema, esto es, analizar el espacio de memoria usada en eXist hasta el momento, el espacio libre, la versión con la que se esta trabajando, etc. Si no se es administrador pero ya se tiene una cuenta como usuario de eXist, se puede usar una de las herramientas contenidas en esta base de datos llamada Webstart Client, cuando se elige esta opción se abre una pequeña ventana donde pregunta el nombre de usuario y la contraseña, una vez ingresados se puede acceder a las funcionalidades permitidas de acuerdo a los permisos otorgados por el administrador.

Como toda base de datos eXist soporta lenguajes de consulta, los cuales son usados para extraer la información almacenada en las colecciones contenidas en ésta. Con éstos lenguajes se pueden procesar de manera eficiente las consultas de texto completo, incluyendo búsquedas de palabras clave, consultas

de acuerdo a la proximidad de los términos de búsqueda o expresiones regulares [18].

En términos generales, las tecnologías que soporta eXist son las siguientes:

- XPath: Es un lenguaje de consulta desarrollado por el equipo de trabajo del W3C, el cual construye expresiones que recorren y procesan un documento XML utilizando la estructura jerárquica del XML.

- XQuery: Es un lenguaje de consulta que extrae información desde colecciones XML.

- WebDav: Protocolo que permite crear, cambiar y mover documentos en un servidor remoto (servidor web).

- REST: Considerado como una arquitectura de software para distribuir los sistemas hipermedia como la World Wide Web.

- SOAP: Es un protocolo que establece dos objetos en procesos diferentes para que puedan comunicarse por medio de intercambio de datos XML.

- XACML: Es un lenguaje de control de acceso implementado en XML y un modelo de procesamiento, que describe como interpretar las políticas.

- Xinclude: Es una herramienta utilizada para la fusión de documentos XML, que agrega etiquetas a un documento XML principal y para poder incluir automáticamente los demás documentos o partes de ellos en éstas.

eXist también permite el desarrollo de aplicaciones web basadas en XQuery, las cuales pueden ser escritas en éste lenguaje usando XSLT, XHTML, CSS y javascript.

Es importante mencionar que eXist cuenta con una serie de documentos XML de ejemplo y consultas en XQuery y XPath listas para ejecutarse. Para la ejecución de las consultas ésta base de datos ofrece una herramienta llamada XQuery Sandbox, que es una pantalla donde se pueden ingresar consultas en los lenguajes descritos anteriormente.

Por las características vistas en ésta sección, la facilidad de instalación y por la ventaja de ser de código abierto, se consideró a eXist una base de datos de gran ayuda para el desarrollo de la herramienta que se está proponiendo en la presente investigación. ésta NXD permitirá el almacenamiento de la información en el formato XML, además aceptará consultas sobre los documentos utilizando los lenguajes XQuery y XPath, que serán indispensables en el desarrollo de la misma.

3.7 Vistas en XML

Una vez analizada la relación de las bases de datos y el XML, se abordará un tema no menos importante como es la creación de vistas sobre documentos XML. De la misma forma en que se pueden ejecutar consultas SQL sobre una base de datos relacional, se puede consultar con XQuery una NXD.

Si se habla de una base de datos relacional, es posible crear vistas, es decir, extraer cierta información de la base de datos y almacenarla en una relación virtual, de tal manera que la relación no existe físicamente pero es posible consultarla como si estuviera materializada. En las NXD aún no es posible crear vistas, por lo que en este trabajo se creará un mecanismo de vistas sobre documentos XML almacenados en este tipo de base de datos.

Para crear una vista sobre un documento XML, se debe tener claro que se espera obtener si un documento XML o un bosque (colección de árboles no ordenados), debido a que una vista XML debe incluir uno o más fragmentos XML. Mediante el siguiente ejemplo se comprenderá mejor esta diferencia:

La figura 3.3, representa los datos en forma de árbol, mientras que la figura 3.4 implica un bosque, no existe un elemento raíz por lo tanto no esta bien formado.

Existe una diferencia clara en lo que será tratado como una vista en el presente trabajo y lo que es resultado de una consulta, por ejemplo: En un documento XML se tienen almacenados

```
<Cliente ClienteID= '1'>
        <Orden ClienteID= '1' OrdenId = '10308'>
        <Orden ClienteID= '1' OrdenId = '10625'>
</Cliente>
```

Figura 3.3: Documento XML bien formado, porque contiene un elemento raíz.

```
<Orden ClienteID= '1' OrdenId = '10308'>
<Orden ClienteID= '1' OrdenId = '10625'>
```

Figura 3.4: La representación de un fragmento es un bosque.

datos de los actores más famosos del momento, datos como son: nombre, dirección (calle y ciudad) y películas (título y año) en las que han actuado desde 1995. Una consulta XQuery a ese documento podría ser la mostrada en la figura 3.5.

La figura 3.5, muestra una consulta que obtiene los nombres de los actores almacenados en el documento Actores1.XML mostrado en el Apéndice A, el resultado de esta consulta es un

```
for $p in //Actores
return
    <Actores>
        {$p/ACTOR/NOMBRE}
    </Actores>
```

Figura 3.5: Consulta que obtiene los nombres de los actores almacenados en el documento Actores1.XML

```
<Actores>
  <NOMBRE> .... </NOMBRE>
  <NOMBRE> .... </NOMBRE>
  ...          ....
  ...          ....
</Actores>
```

Figura 3.6: Resultado de la consulta propuesta en la Figura 3.5

```
create view Nombres_Actores
        for $p in //Actores
        return
                <Actores>
                        {$p/ACTOR/NOMBRE}
                </Actores>
```

Figura 3.7: Creación de una vista utilizando una consulta XQuery.

documento bien formado:

La consulta que se muestra en la figura 3.5, no es considerada una vista únicamente es una consulta ejecutada por una NXD que acepta XQuery como lenguaje de consulta, llamaremos vistas a las consultas a las cuales se les asocia un nombre. Retomando el mismo ejemplo, se creará ahora una vista:

La propuesta de este trabajo es que si hoy en día se pudieran crear vistas desde XQuery seguramente se verían de la siguiente forma: Como se puede observar en la figura 3.7, esta ya no es una consulta simple, si no que es una consulta con nombre, la cual al igual que sus semejantes en las bases de datos

relacionales pueden ser borradas y consultadas desde otras consultas XQuery.

Una vez establecido lo que se denominará vistas en este trabajo, se establece que los resultados que se esperan obtener de la consulta de una vista serán únicamente documentos XML bien formados.

Esta temática de vistas extraídas de documentos XML, es lo que veremos más a detalle en las próximas secciones.

3.8 Antecedentes de Vistas

Como se ha dicho anteriormente al hablar de vistas en el modelo relacional, el tema esta bastante investigado y resuelto, sin embargo, al referirse a vistas extraídas desde documentos XML, la temática se vuelve interesante, tanto que es un tema de investigación vigente, por tal razón en la literatura es posible encontrar algunas propuestas de como resolver el problema, pero aun no se han implementado.

En las siguientes secciones se presentan diferentes propuestas hechas hasta ahora con respecto a vistas sobre documentos XML.

3.8.1. Control de acceso basado en bitmaps.

En [1] el tema de vistas es asociado específicamente al control de acceso en los documentos XML. Esta propuesta para la

creación de vistas y restricción del acceso a los datos involucra listas de capacidades, listas de control de acceso y matrices, las cuales ya han sido usadas en otros sistemas convencionales que implementan control de acceso.

Debido a que cualquier documento XML se puede ver como un conjunto de elementos y nodos (árbol), se pueden distinguir varias categorías de acceso a diferentes partes del documento:

- Restricción de vista a nivel documento: Se refiere a restringir la cantidad de documentos que puede ver el usuario.
- Restricción de vista a nivel nodo: En un documento, el usuario puede tener permisos para accesar a algunos de los nodos existentes, por lo que se define la restricción para ver únicamente esos nodos.
- Restricción de vista combinada: Este tipo de restricción combina las anteriores, por lo tanto algunos usuarios tienen restringida la cantidad de documentos que pueden ver, además de los nodos que pueden visualizar de esos documentos.

Esta propuesta [1] introduce el término de ePaths, los cuales son definidos como la secuencia de elementos anidados, donde el elemento más anidado es un elemento de contenido simple. Por tal razón, un documento XML es entonces una secuencia de ePaths con elementos asociados contenidos. Si observamos

la tabla 3.1, podemos observar que 'e0.e2.e4' es un ePath en el documento d1, porque el elemento e4 tiene un contenido simple.

Así mismo en la tabla 3.1, también se pueden obtener los siguientes ePaths: p0=e0.e1, p1=e0.e2.e3, p2=e0.e2.e4, p3=e0.e5, p4=e0.e2.e4.e6, p5=e0.e2.e4.e7, p6=e0.e8, p7=e0.e9.

Otro término clave de esta propuesta es el bitmap (tabla 3.2), el cual es definido como una representación bidimensional de la relación entre dos entidades diferentes, por ejemplo: ePaths y documentos. En documentos XML, la presencia de un ePath, se puede expresar utilizando dicho bitmap, de tal forma que si el documento tiene un ePath (P0, P1, P2, . . .), el bit correspondiente es de 1, en otro caso el bit se almacena a 0.

Observando la tabla 3.1, obtenemos que el documento d1, tiene un ePath P0, denotado como p0=e0.e1, sin embargo, no tiene un ePath en P4, el cual se denota por p4=e0.e2.e4.e6. Debido a que el control de acceso es un mecanismo que los sistemas de información utilizan para permitir o rechazar el acceso a los datos, entonces en los documentos XML se puede tener control de acceso en diferentes niveles:

a)A nivel de DTD.

Este nivel se representa en la siguiente figura:

La tabla 3.3, representa el bitmap para cuatro usuarios o grupos y cuatro ePaths; la presencia de un 1 significa que ese usuario o grupo tiene la capacidad de acceder al ePath P1 en todos los documentos.

d1:	d2:	d3:
<e0>	<e0>	<e0>
<e1>V1</e1>	<e1>V1</e1>	<e1>V11</e1>
<e2>	<e2>	<e2>
<e3>V2V3V5</e3>	<e3>V3V7</e3>	<e3>V2V7</e3>
<e4>V3V8</e4>	<e4>V9	<e4>V3V9</e4>
<e5/>	<e6>V4</e6>	<e5/>
</e2>	<e7>V6</e7>	</e2>
</e0>	</e4>	<e9>V5</e9>
	</e2>	</e0>
	<e8>V6V12</e8>	
	</e0>	

Cuadro 3.1: Ejemplos de documentos XML

b) A nivel de Documento.

La tabla 3.4, es un bitmap que representa el nivel de documento con cuatro usuarios o grupos y cuatro documentos, la

Documentos	ePaths							
	P0	**P1**	**P2**	**P3**	**P4**	**P5**	**P6**	**P7**
d1	1	1	1	1	0	0	0	0
d2	1	1	1	0	1	1	1	0
d3	1	1	1	1	0	0	0	1

Cuadro 3.2: Bitmap para los documentos XML, mostrados en la tabla 3.1

Usuarios	ePaths			
	P1	**P2**	**P3**	**P4**
Usuario1	1	0	0	1
Usuario2	1	0	1	0
Usuario3	0	0	0	0
Usuario4	1	1	1	1

Cuadro 3.3: A nivel DTD.

presencia de un 1 indica que el usuario tiene la capacidad de visualizar ese documento en particular, de tal forma que el Usuario4 puede ver los cuatro documentos.

c) A nivel de ePath. En la tabla 3.5, se muestra el bitmap que representa el acceso a la información para un solo usuario o grupo. A dicho usuario o grupo se le restringe el acceso a un

Usuarios	Documentos			
	D1	D2	D3	D4
Usuario1	0	1	0	1
Usuario2	0	0	1	0
Usuario3	1	0	0	0
Usuario4	1	1	1	1

Cuadro 3.4: A nivel documento.

Documentos	ePaths			
	P1	P2	P3	P4
Documento1	0	0	0	0
Documento2	1	0	0	0
Documento3	0	0	0	0
Documento4	1	0	0	1

Cuadro 3.5: A nivel de ePath.

ePath en específico de un documento, por ejemplo: este usuario o grupo tiene permitido ver el documento2 únicamente en sus ePaths P1 y P4.

d) A nivel de Contenido

Este bitmap (tabla 3.6), representa pares (ePath, contenido) para cada documento, al igual que el anterior representa el ac-

Documentos	ePaths			
	(P1,W1)	(P2,w2) ... (P2,w1)	...	(P4,W4)
Documento1	0	1 ... 0	...	0
Documento2	1	0 ... 0	...	0
Documento3	0	0 ... 0	...	0
Documento4	1	1 ... 1	...	1

Cuadro 3.6: A nivel de contenido.

ceso a la información para un solo usuario/grupo. El bitmap indica que ese usuario/grupo puede acceder a un ePath y a su contenido de un documento en específico, de tal forma que si el documento puede verse por completo, tiene 1's en todos los pares(ePaths, contenido).

3.8.2. Vistas conceptuales

En [12] se propone que la vista XML a crearse se defina como un documento imaginario XML, debido a que este documento no existe por si mismo, sino que se obtiene a partir de otros documentos que están físicamente almacenados. Además se pretende darle al usuario la capacidad de modelar XML desde un alto nivel de abstracción usando metodologías ya existentes, tales como modelos conceptuales orientados a objetos, usando notaciones del Lenguaje de Modelado Unificado

(UML) y transformando estos modelados directamente en un modelo XML. El punto más importante en esta propuesta es que dicha vista difiere de otros trabajos similares, porque aquí la vista no es creada a partir de una consulta, sino que a partir de un mapeo de "Vista Conceptual". Se define a una vista conceptual como una vista XML, la cual se determina en el nivel conceptual con el nivel más alto de abstracción.

Las vistas conceptuales XML en [13] se crean siguiendo el siguiente proceso:

- Se obtiene la percepción del mundo real.
- Se crea el modelo conceptual utilizando UML o redes semánticas, el cual será utilizado para mapear la información de interés al modelo de vista conceptual.
- En este paso se obtiene el modelado del esquema (esquema XML) que será mapeado al esquema de vista XML.
- Finalmente se obtiene el modelado de como se almacenarán los documentos XML, el cual será mapeado al modelado de almacenamiento del los documentos de vista XML.

Como se puede observar en los pasos anteriores se modela la vista conceptual en etapas de análisis y diseño independientemente del almacenamiento físico del esquema. Dicha vista con-

ceptual debe contener:

- Un nombre para la vista.
- La definición de un esquema conceptual válido, el cual restringe y válida el documento.
- Una colección de etiquetas y sus espacios de nombres (o dominio).
- Nuevas etiquetas y sus espacios de nombres los cuales son derivados desde otros y,
- Un constructor (que es especificado en el nivel conceptual) que define como el documento será materializado.

Los 5 puntos anteriores implican que un documento de vistas XML, debe contar con las siguientes características:

1. La vista conceptual debe incluir: un nombre, un esquema conceptual válido, una colección de etiquetas con sus espacios de nombres, nuevas etiquetas (con sus espacios de nombres) y un constructor opcional.
2. Las consultas utilizadas en la definición de la vista conceptual son opcionales. Debido a que la vista conceptual es definida en un alto nivel, éstas pueden especificarse usando sintaxis abstracta, tal como un SDX u omitirla.

3. La definición de etiquetas para la vista conceptual es semejante a la definición de etiquetas del documento XML almacenado.

4. Cuando se genera una nueva etiqueta XML ya sea definiéndola o derivándola para la vista conceptual, esta etiqueta quizá sea derivada desde una o más etiquetas en el esquema/espacio de nombres XML almacenado.

5. Las limitaciones adicionales, tales como: ámbito, valores por default, restricciones, pueden ser definidas para etiquetas en la vista conceptual.

6. Para una vista conceptual, la especificación solo es definida usando una notación declarativa dada, como lo son las redes semánticas o el modelado UML.

Una vez que ya se tiene la vista conceptual definida, se hace el mapeo entre ésta a la vista XML, el mapeo es 1:1 entre ellas. Si la vista conceptual fue bien creada, la única diferencia que debería prevalecer entre ellas es el nivel de abstracción. Las siguientes tablas muestran cómo la vista conceptual es mapeada a una vista XML, usando UML(tabla 3.7) y redes semánticas (tabla 3.8)

En [13] se establece que la definición de vistas inicia en el lenguaje de manipulación de datos (LMD). En esta propuesta se defiende que las vistas deben ser definidas en el nivel más

abstracto (nivel conceptual) porque esto mejorará su resultado, es una idea similar al modelo conceptual que tiene todo sistema de software.

3.8.3. Vistas en XPERANTO

Las secciones anteriores describieron propuestas para la creación de vistas sobre documentos almacenados en bases de datos, las cuales no han sido implementadas en software hasta hoy en día.

En 2002 [2], IBM hizo una publicación acerca de un middleware llamado XPERANTO que resolvería el problema de la creación de vistas XML, el cual no ha salido aún al mercado. Una de las cosas que proponen en este articulo es que XPERANTO tiene la habilidad de crear vistas XML sobre datos que se encuentran almacenados en bases de datos relacionales. XPERANTO tendrá la facultad de crear vistas XML de forma automática mapeando los datos desde un sistema de base de datos relacional a una vista XML por default. Por lo tanto, el usuario ingresa una consulta en específico de lo que quiere visualizar, el middleware construye esa vista XML sobre la vista XML creada inicialmente por default, usando el lenguaje de consultas XQuery.

Otra de las características que IBM resalta de XPERANTO es que permite hacer consultas sobre las vistas XML creadas desde datos relacionales. Esta parte es importante porque muchas

veces los usuarios quieren consultar un subconjunto de datos proyectados en la vista o necesitan extraer datos desde múltiples vistas, todas las consultas se realizan usando XQuery.

IBM define a XPERANTO en [2] como diferente de XSL-T en tres aspectos:

- XSL-T funciona para crear y consultar datos desde una vista XML creada sobre datos relacionales, pero para realizar esta tarea necesita que los datos sean materializados antes de ser procesados por el mismo. De tal forma que las consultas no pueden ser entonces procesadas por el motor relacional.

- Se reduce la eficiencia por la cantidad innecesaria de datos que tienen que ser materializados, además algunos fragmentos XML que no forman parte de la consulta final quizá también tengan que ser materializados.

- El procesamiento de consultas es más eficiente en XPERANTO, porque es un middleware que deja la mayor parte del cómputo al motor de la base de datos relacional.

3.8.4. XViews – Mecanismo de vistas para datos XML Nativos

XViews es un mecanismo propuesto por IBM que pretende crear, borrar y consultar vistas XML. En [6] se reconoce que

el poder hacer uso de las vistas es indispensable para cualquier base de datos, además considera a las vistas importantes porque con ellas se pueden tener ventajas, tales como:

- Permiten a los desarrolladores de aplicaciones definir porciones de la base de datos que ellos desean visualizar en su aplicación, lo cual simplifica el desarrollo de la misma, porque ya no tienen que programar la vista que desean presentar al usuario, sino que la misma base de datos le proporciona esta utilidad.

- Se simplifican las consultas hechas a la base de datos, ya que al usar vistas, la consulta se hace sobre una porción y no sobre el documento completo.

- Permiten mantener confidencialidad, porque se restringe el acceso del usuario a solo las partes seleccionadas de la base de datos que conforman la vista.

Esta propuesta creará utilizando el lenguaje XQuery vistas sobre documentos XML almacenados en repositorios nativos XML. Una de las características importantes que aquí se menciona es que aunque la vista no existe físicamente, una vez creada ésta el usuario podrá consultarla como si fuera un documento almacenado.

Cuando XViews recibe una petición para crear una vista,

toma esta consulta y crea un esquema XML de la misma, agregando un elemento constructor con el nombre que se le dio a la vista. Dicho esquema será tratado como virtual y sobre este se harán todas las demás consultas. Las vista serán almacenadas y usadas posteriormente en las consultas, ya que XViews permite la creación, borrado y consulta sobre vistas.

Cuando se diseñó XViews se contempló que al consultar una vista, se ejecutará primeramente la consulta de la definición de la vista y almacenar sus resultados en un archivo XML temporal, de tal forma que las consultas que hiciera el usuario serían ejecutadas sobre éste, es decir, la vista es materializada. Sin embargo, esta solución no es tan útil, quizá sería eficiente en casos que la vista a materializar fuera pequeña. Dicha solución fue pasada a otro nivel de optimización en la que se planeó analizar la definición de la vista de tal forma que únicamente se materializarán partes relevantes en ella y éstas mezclarlas con la consulta que hace el usuario sobre la vista, de tal forma que el mapeo final que se tenga de la consulta obtenga los resultados correctos de forma eficiente. Hubo otra optimización propuesta, la cuál basaba su solución en el álgebra XML. Cabe aclarar que ninguna de estas optimizaciones han sido implementadas.

Una vez presentadas las propuestas realizadas hasta hoy en día con respecto a las vistas, en el siguiente capítulo se presentará el aporte que tiene ésta investigación.

Propiedad de la Vista	**Vista Conceptual (Usando UML)**	**Vista XML**
Representación	Clase	Documento XML
Nombre de la vista	Nombre de clase UML válida	Nombre de Documento (sintáxis específica XML)
Esquema	Lista de especificación de atributos (nombre, tipo, restricciones, etc..)	Esquema XML
Dominio	Dominio de clases	Espacio nombres XML
Datos	Valor de atributo	Valor de las etiquetas XML encapsuladas
Constructor	Especificación de clases Método/Funciones (Opcional)	Lenguaje específico (Por ejemplo: XPath, XQuery, SQL)

Cuadro 3.7: Mapeo entre Vista conceptual (UML) y Vista XML.

Propiedad de la Vista	**Vista Conceptual (Usando redes semánticas)**	**Vista XML**
Representación	Colección de nodos y aristas conectados a una raíz.	Documento XML
Nombre de la vista	Etiqueta del nodo raíz	Nombre de Documento (sintáxis específica XML)
Esquema	Colección conectada de nodos y aristas etiquetadas	Esquema XML
Dominio	Dominio de redes semánticas	Espacio nombres XML
Datos	Nodos hoja	Valor de las etiquetas XML encapsuladas
Constructor	Operadores de red semánticos (operadores unarios/conjunto)(Opcional)	Lenguaje específico (Por ejemplo: XPath, XQuery, SQL)

Cuadro 3.8: Mapeo entre Vista conceptual (Usando redes semánticas) y Vista XML.

4

Análisis y diseño del mecanismo de creación de vistas XMLView

4.1 Importancia de la creación de vistas sobre documentos XML

La creación de vistas sobre documentos XML, es un problema no resuelto, como se veía en el capítulo anterior, muchas han sido las ideas propuestas sobre este tema, pero todavía hay mucho por hacer tomando en cuenta como funcionan las vistas basadas en el modelo relacional y todo lo que se puede hacer sobre ellas en él.

Por lo anterior se puede aseverar que las vistas creadas sobre documentos XML son importantes por varios aspectos, desde el control de acceso, es decir, por motivos de seguridad, debido a que en muchas ocasiones se desea que los usuarios, vean

solo una pequeña parte de la información almacenada en la base de datos. Así como también porque al crear la vista sobre un documento, la información en ella se resume, por lo tanto, al hacer consultas sobre ésta solo se procesan sobre la parte de información que está en su definición y no contra todo el documento, lo cuál debería permitir mayor eficiencia en dichas consultas.

En las siguiente sección se describe brevemente como funcionará la herramienta XMLView, que es la propuesta de la presente investigación.

4.2 Funcionalidad General de XMLView

Como se planteó en la sección 3.7, se le llamará vista a toda aquella consulta con nombre. El funcionamiento de XMLView sobre documentos XML, retomará el concepto de vista manejado en el modelo relacional, por lo tanto se espera que todo documento–vista XML no sea materializado al crearse, sino que sea considerado como un documento virtual, debido a que sólo se almacenará su definición.

XMLView permitirá que el usuario ejecute cualquier consulta XQuery o XPath, con la diferencia que será añadida a la sintaxis de estos dos lenguajes un “;” al final de la consulta que será utilizado como carácter que indique el fin de la misma.

Si este carácter no es ingresado el software marcará un error y no devolverá ningún resultado, aunque la consulta de acuerdo a XQuery o XPath sea correcta. Si se ingresa una consulta y ésta no es correcta sintácticamente el software debe marcar un error indicando en que línea se encuentra éste y una vez corregido se podrá volver a consultar y obtener el resultado deseado.

Al ingresar la consulta sobre XMLView, éste sin excepción deberá extraer los documentos con extensión ".xml" en la consulta. Una vez obtenidos deberá verificar si éstos documentos existen para eXist y si pertenecen a un documento físicamente almacenado o a una vista. Si existe más de un documento en la consulta y al menos uno de ellos pertenece a una vista, se enviará un mensaje al usuario ya que ésta será la primera versión y no permite multiplicación entre documentos físicos y documentos–vista. Si en la consulta existe un solo documento XML y éste corresponde a una vista se reestructurará la consulta para que eXist la pueda entender y ejecutar, el como se hace será visto en las próximas secciones de este capítulo. Si la consulta es únicamente sobre documentos almacenados físicamente en eXist, ésta será enviada directamente a eXist para que la ejecute y regrese los resultados al software quién mostrará en pantalla al usuario el documento XML resultante de la misma y para el caso de que el documento mencionado en ésta no exista también se enviará un mensaje al usuario para informarle tal situación.

Cuando la consulta a ejecutar sea iniciada con un coman-

do Create o Drop en cualquiera de sus variantes (mayúsculas, minúsculas, etc. . .) significará que el usuario desea crear o borrar vistas por lo que XMLView tendrá que evaluar la consulta para que ésta contenga todos los elementos necesarios para ejecutarse, se espera que si es una creación de vistas cuente con lo siguiente: Create View Nombre_vista {definición_de_la_vista}; y si se refiere a un borrado, deberá contener: Drop View Nombre_vista; por lo tanto, esta herramienta también deberá contener dos analizadores:

- Léxico: el cual detectará que los tokens create, view, drop estén formados con los caracteres correspondientes. Además que el nombre de la vista sea válido, es decir, que empiece por una letra, seguido de un número o una letra y que no contengan caracteres especiales. La parte de la definición de la vista, consiste en una sentencia XQuery o XPath, de tal forma que éste también debe identificar que esta porción corresponde a una consulta y no a algo incorrecto dentro de la consulta en general.

- Sintáctico: éste se encargará de verificar que cada uno de los componentes de la consulta se encuentre en el orden adecuado.

Esto es a grandes rasgos como funcionará XMLView, a continuación se presentará la arquitectura que tendrá la aplicación.

4.3 Arquitectura de XMLView

{arquitectura

En el transcurso de la investigación se definieron todos aquellos aspectos que el mecanismo de creación de vistas sobre documentos XML planteado en este trabajo debería contener y en base a las necesidades de su desarrollo se decidió utilizar un patrón de diseño el cuál es Modelo Vista Controlador (MVC) [16].

Un patrón de diseño se enfoca en un problema y encuentra la forma de cómo darle solución a éste, de tal manera que se puede usar la solución miles de veces cuando se presente dicho problema, sin que necesariamente se haga de la misma forma dos veces. Por lo tanto, un patrón otorga las estrategias o buenas prácticas para desarrollar una aplicación orientada a objetos, la ventaja de utilizarlos de acuerdo a su definición es que le dan la oportunidad al diseñador de la solución de utilizar un enfoque que ya ha sido probado anteriormente por otros diseñadores, son fáciles de adoptar, facilitan el trabajo haciendo el código más legible y permiten la reutilización de componentes [7].

Cada patrón describe un problema que ocurre una y otra vez en nuestro medio ambiente y, a continuación se describe el núcleo de la solución a ese problema, de tal manera que puede utilizar esta solución un millón de veces, sin llegar a hacerlo de la misma manera dos veces.

. El patrón de diseño MVC[16] separa la funcionalidad de una aplicación de software en capas las cuales pueden ser: Interfaz del usuario, lógica del negocio y los datos que maneja dicha aplicación. Para la creación de XMLView las capas se han dividido en:

- La capa de Vista la cual será tratada en el diseño de ésta herramienta como de Presentación, se compone de los elementos de la interfaz que interactuarán directamente con el usuario, permitiéndole acceder a los datos de la interfaz y manipularlos. Esta capa presenta los datos producidos por la capa de negocio.

- La capa de Modelo será llamada de Negocio y en ésta se ubican los componentes encargados de darle la funcionalidad al sistema. Dichos componentes procesarán las peticiones que son hechas por el usuario a través de la capa de presentación y si es necesario se comunicará con la capa de control para hacer el requerimiento de datos necesario para devolver la respuesta a la petición del usuario.

- La capa de Control contiene a la base de datos donde se almacenará, consultará o extraerá la información requerida por la capa de presentación la cuál hizo su petición a través de la capa de negocio.

Una vez diseñada la arquitectura del software, se procede a la especificación del ambiente de desarrollo:

- Ambiente de Desarrollo Integrado (IDE): NetBeans IDE 6.5
- Compilador: JDK 6 update 16.
- Herramientas de Diagramación UML: plug-in UML NetBeans.
- Sistema de Bases de Datos Nativa: eXist.
- Servidor de aplicación que soporta la Base de Datos Nativa: Jetty Web Server.

Es importante mencionar que se decidió desarrollar sobre NetBeans porque anteriormente ya había sido utilizado por el autor de éste trabajo de tesis, por lo tanto, fue el dominio de ésta herramienta el factor predominante para su elección. eXist y Jetty fueron elegidos por ser software libre, por su facilidad de uso e instalación y porque existe soporte confiable en el Internet para su trabajo con Java.

4.3.1. Casos de uso

En el análisis de toda aplicación de software es necesario identificar los casos de uso que describirán la secuencia de acciones que realizará ésta cuando el actor interactúe con ella y así poder dar una respuesta correcta a este. En este caso el único

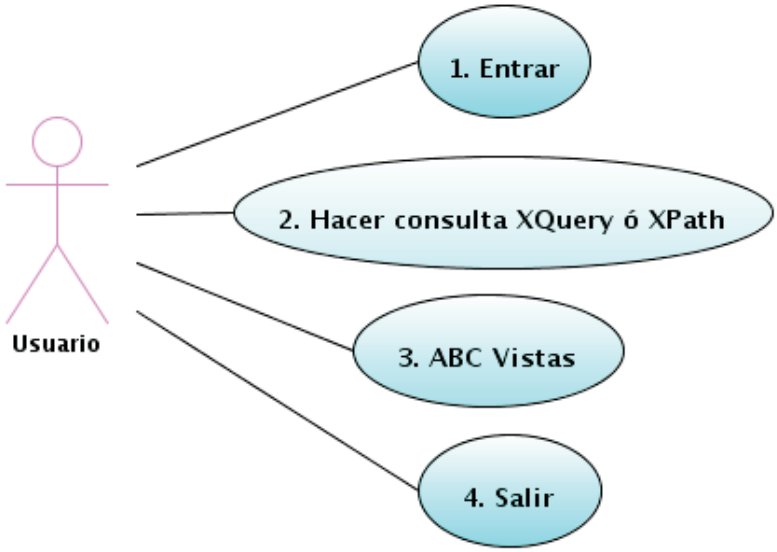

Figura 4.1: Caso de uso general de XMLView.

actor para esta aplicación será el usuario, que es quien va a interactuar con XMLView para obtener el resultado de la consulta deseada, en base a esto el caso de uso general se presenta a continuación: En la figura 4.1, se observa que el usuario interactúa con XMLView, por lo que primero debe ingresar al sistema (Entrar), posteriormente puede elegir cualquiera de los dos casos de uso: Hacer consulta XQuery o XPath o ABC Vistas (Altas, Bajas y Consulta de Vistas) y al concluir las acciones a realizar deberá salir de la aplicación (Salir). El caso de uso ABC Vistas involucra todas las acciones para administrar una vista en

específico lo cual consiste en crear, borrar y consultar.

4.3.1.1. Detalles de casos de uso

Cada uno de los casos de uso mostrados en la Figura 4.1, se muestran a detalle a continuación:

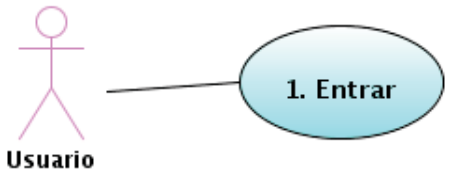

Figura 4.2: Detalle de caso de uso Entrar.

Caso de Uso 1: Entrar.
Actor: Usuario.
Descripción: El usuario ingresa al sistema.
Precondición: El usuario desea ingresar al sistema.
FLUJO:

ACTOR		SISTEMA		
PASO	**ACCIÓN**	**PASO**	**ACCIÓN**	**EXCEPCIÓN**
1	Da click en el acceso directo a XMLView	2	Muestra la pantalla con todas las funcionalidades con las que cuenta.	-

Postcondición: El usuario ingresa al sistema y se le otorgaron todas las funcionalidades con las que cuenta éste.

En la figura 4.2, se presenta la forma en la que el usuario ingresará al sistema, éste únicamente deberá dar un click en el acceso directo de XMLView y el sistema mostrará la pantalla donde se puede ejecutar cualquiera de las funcionalidades con las que cuenta el sistema.

Figura 4.3: Detalle de caso de uso Hacer consulta XQuery ó XPath.

Caso de Uso 2: Hacer Consulta XQuery o XPath.
Actor: Usuario.
Descripción: El usuario puede hacer una consulta en cualquiera de los dos lenguajes permitidos: XQuery y XPath.
Precondición: El usuario desea hacer una consulta en cualquiera de los lenguajes disponibles.
FLUJO:

ACTOR		SISTEMA		
PASO	**ACCIÓN**	**PASO**	**ACCIÓN**	**EXCEPCIÓN**
1	Ingresa la consulta en lenguaje XQuery o XPath y oprime el botón “Send”.	2	Extrae el nombre de los documentos XML de la consulta y verifica que no sean documentos–vista.	E1, E2
		3	Si los documentos no son documentos–vista, envía la consulta a eXist quién analiza la consulta y la ejecuta.	E2, E3
		4	Recibe desde eXist los resultados de la consulta y los muestra en pantalla.	E2
5	El usuario puede decidir entre presionar el botón “New Query” o “Exit”.	-	-	-

EXCEPCIONES

ID	DESCRIPCION	ACCIÓN
E1	Documento No Encontrado	El sistema envía un mensaje de error al usuario indicando que el documento no existe.
E2	Falló en la conexión con la Base de datos	El sistema muestra un mensaje de error indicando que falló la conexión con la base de datos, solicitando se intente nuevamente más tarde.
E3	Consulta Incorrecta	El sistema envía un mensaje de error indicando que la consulta es sintácticamente incorrecta, mostrando al usuario en que linea se cometió el error.

Postcondición: El usuario recibe los resultados de la consulta que ingresó, en un documento XML bien formado.
La figura 4.3, muestra el detalle de caso de uso Hacer consulta XQuery o XPath, donde el usuario deberá ingresar a la pantalla

su consulta a procesar, una vez hecho esto deberá dar click en el botón “Send”, XMLView extraerá los nombres de los documentos XML en la consulta y verificará que ninguno de ellos corresponda a un documento-vista, de cumplirse esta condición envía la consulta a eXist quién verificará que la consulta sea sintácticamente correcta y de ser así enviará los resultados al sistema y XMLView los presentará en pantalla al usuario. La figura 4.4, muestra la administración de las vistas, lo cual involucra 3 casos de uso: Crear vista, borrar vista y consultar vista; las cuáles se verán a detalle en las siguientes figuras.

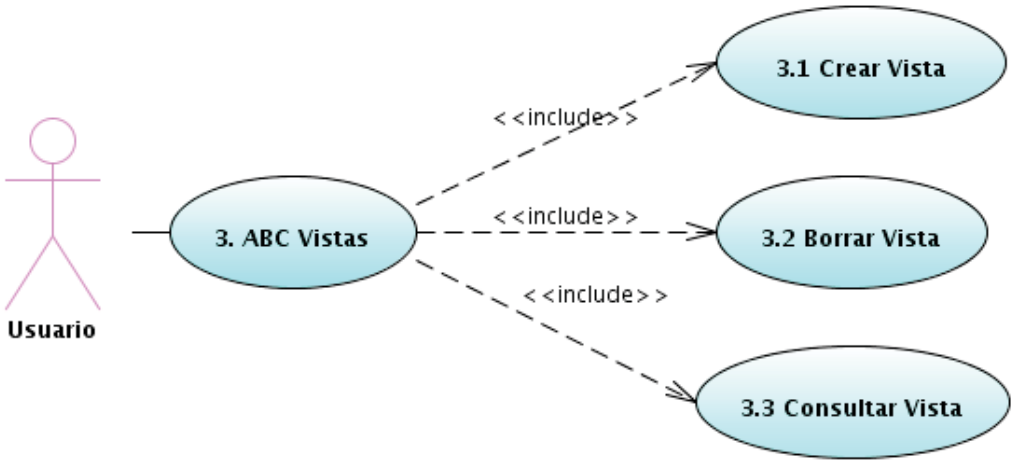

Figura 4.4: Detalle de caso de uso ABC Vistas.

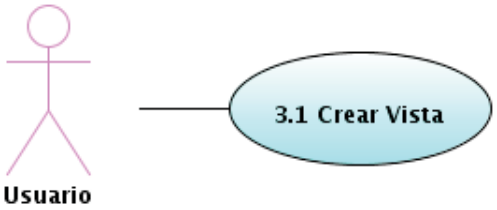

Figura 4.5: Detalle de caso de uso de Crear vista.

Caso de Uso 3.1: Crear Vista
Actor: Usuario.
Descripción: El usuario ingresa la consulta para crear una vista.
Precondición: El usuario desea crear una vista, por lo cuál ingresa el nombre de ésta y su definición.
FLUJO:

ACTOR		SISTEMA		
PASO	**ACCIÓN**	**PASO**	**ACCIÓN**	**EXCEPCIÓN**
1	Ingresa la consulta para crear la vista y presiona el botón “Send”	2	Verifica que la consulta sea correcta.	E1, E2
		3	Si la consulta es correcta, extrae el nombre de la vista y verifica que no exista en eXist.	E2, E3
		4	Crea la vista y almacena la definición de ésta en eXist.	E2
		5	Muestra mensaje al usuario de que la vista fue creada satisfactoriamente.	
6	El usuario puede decidir entre presionar el botón “New Query” o “Exit”.	-	-	-

Postcondición: El usuario crea una vista sobre un documento XML, cuya definición se almacena en eXist.
La figura 4.5, presenta el caso de uso que se aplicará cuando el usuario desee crear una vista, para esto se debe ingresar una consulta de creación de vista en el formato Create View Nombre_vista definición_de_la_vista; y presionar el botón "Send", una vez hecho esto el sistema verificará que la consulta sea sintácticamente correcta, luego se extraerá el nombre de la vista y se buscará en eXist, en el caso de que ya sea una vista creada no podrá modificarse por lo que se informará al usuario mediante un mensaje de error, en caso contrario la definición de la vista será almacenada en eXist y el sistema informará que la vista fue creada.

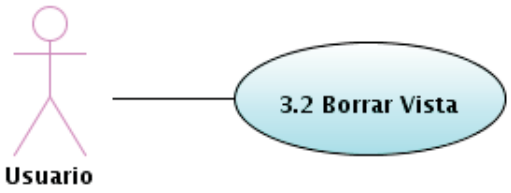

Figura 4.6: Detalle de caso de uso de Borrar vista.

Caso de Uso 3.2: Borrar Vista
Actor: Usuario.
Descripción: El usuario ingresa la consulta para borrar una

vista.
Precondición: El usuario desea borrar la definición de una vista almacenada en eXist.
FLUJO:

ACTOR		SISTEMA		
PASO	**ACCIÓN**	**PASO**	**ACCIÓN**	**EXCEPCIÓN**
1	Ingresa la consulta para borrar una vista.	2	Verifica que la consulta sea correcta sintácticamente.	E1
		3	Extrae el nombre de la vista que se desea borrar y se busca el nombre en eXist.	E2, E4
		4	Una vez encontrado el documento lo elimina desde eXist.	E2
		5	Envía mensaje al usuario de vista borrada satisfactoriamente.	-
6	El usuario puede decidir entre presionar el botón “New Query” o “Exit”.	-	-	-

Postcondición: El usuario borra una vista previamente creada.
La figura 4.6, presenta el caso de uso para Borrar vista, donde el usuario debe ingresar una consulta de borrado de vista en el

Figura 4.7: Detalle de caso de uso de Consultar vista.

formato: Drop View Nombre_vista y presionar el botón “Send”, una vez hecho esto el sistema verificará que la consulta sea sintácticamente correcta, luego se extraerá el nombre de la vista y se buscará en eXist, en el caso de que la vista no exista se mandará un mensaje de error al usuario informando de la situación, en caso contrario se borrará la vista en eXist y el sistema informará que la vista fue borrada satisfactoriamente.

Caso de Uso 3.3: Consultar Vista
Actor: Usuario.
Descripción: El usuario ingresa la consulta sobre un documento-vista.
Precondición: El usuario desea consultar sobre un documento-vista.
FLUJO:

ACTOR		SISTEMA		
PASO	**ACCIÓN**	**PASO**	**ACCIÓN**	**EXCEPCIÓN**
1	Ingresa la consulta sobre un documento-vista.	2	Extrae el nombre del documento y verifica que éste sea un documento–vista.	E2, E4
		3	Si el documento–vista a consultar existe analiza que la consulta sea correcta.	E2, E1
		4	Lee el documento–vista desde eXist y obtiene la definición de éste.	E2, E5
		5	Reestructura la consulta y la envía a eXist para que la ejecute.	E2
		6	Recibe desde eXist los resultados de la consulta y los muestra en pantalla.	
7	El usuario puede decidir entre presionar el botón “New Query” o “Exit”.	-	-	-

EXCEPCIONES

ID	DESCRIPCIÓN	ACCIÓN
E1	Consulta Incorrecta	El sistema envía un mensaje de error indicando que la consulta es sintácticamente incorrecta, mostrando al usuario en que linea se cometió el error.
E2	Falló en la conexión con la Base de datos	El sistema muestra un mensaje de error indicando que falló la conexión con la base de datos, solicitando se intente nuevamente más tarde.
E3	La vista ya existe	El sistema muestra un mensaje de error indicando que la vista ya existe.
E4	Documento No Encontrado	El sistema envía un mensaje de error al usuario indicando que el documento no existe.
E5	No se pudo leer el documento–vista	El sistema envía un mensaje de error al usuario indicando que no se pudo leer el documento–vista.

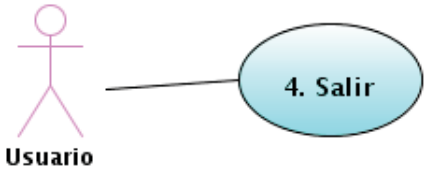

Figura 4.8: Detalle de caso de uso Salir.

Postcondición: El usuario recibe los resultados de la consulta sobre un documento–vista, observando en pantalla un documento XML bien formado.
La figura 4.7, muestra el detalle de caso de uso Consultar vista, donde el usuario ingresará en la pantalla la consulta que desea ejecutar, una vez hecho esto deberá dar click en el botón "Send", XMLView extraerá el nombre del documento XML en la consulta y verificará que corresponda a un documento–vista, de cumplirse esta condición verifica que la consulta sea sintácticamente correcta y extrae la definición del documento-vista desde eXist, reestructura la consulta ingresada por el usuario usando la definición de vista obtenida y la envía a eXist para que la ejecute, el sistema recibe los resultados y los muestra en pantalla al usuario. **Caso de Uso 4:** Salir.
Actor: Usuario.
Descripción: El usuario cierra el sistema.
Precondición: El usuario desea desea salir del sistema.
FLUJO:

ACTOR		SISTEMA		
PASO	**ACCIÓN**	**PASO**	**ACCIÓN**	**EXCEPCIÓN**
1	Da click en el botón “Exit” o presiona el botón cerrar desde la barra de título de la aplicación “X”.	2	Cierra la pantalla del sistema.	-

Postcondición: El usuario cierra el sistema.
La figura 4.8, muestra el caso de uso salir, para este fin el usuario tiene dos opciones ya sea presionar el botón “Exit” en el área de la pantalla ó el botón “X” que se encuentra ubicado en la barra de título.

4.3.1.2. Diagrama detallado de clases

Los diagramas de paquetes como se dijo en la sección **??** serán estructurados de acuerdo al patrón MVC, en 3 niveles de aplicación: A continuación se presenta el diseño detallado de las clases.

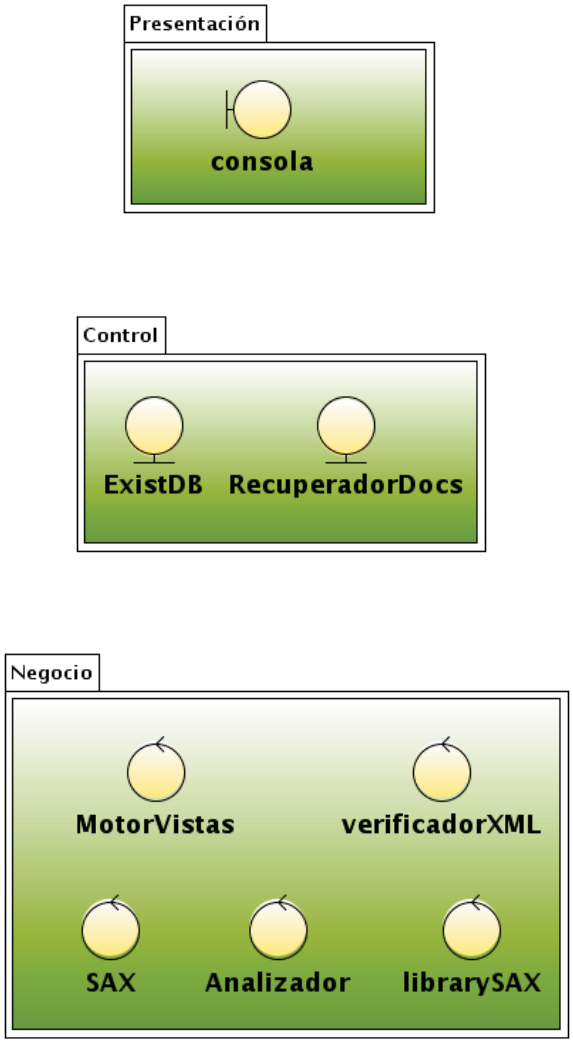

Figura 4.9: Arquitectura de 3 capas: Presentación, Control y Negocio, de acuerdo al patrón de diseño MVC.

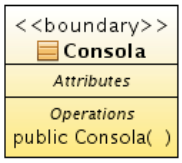

Figura 4.10: Diagrama detallado de clases de la capa de Presentación.

4.3.1.3. Diagramas de secuencia

A continuación se presentan los diagramas de secuencia que muestran el escenario que se ejecuta cuando el objeto Usuario desea realizar las diversas actividades dentro de XMLView, se podrá observar los mensajes de intercambió entre las clases y la funcionalidad que tiene cada una de ellas, dichas clases se presentaron en el diagrama de paquetes mostrado en la figura 4.9 y en forma más detallada en las figuras 4.10 – 4.12. El proceso de entrar al sistema XMLView, se muestra en la figura 4.13, donde el usuario envía el mensaje Entrar a Consola que es la interfaz con la que él se comunicará directamente, esta le puede responder enviando la pantalla en la que podrá gozar de todas las funcionalidades con las que cuenta el sistema o con un mensaje de error que indicará que el sistema no esta disponible. El diagrama de secuencia mostrado en la figura 4.14, presenta la forma en la que el usuario interactúa con XMLView para poder hacer una consulta XQuery ó XPath, se puede observar tam-

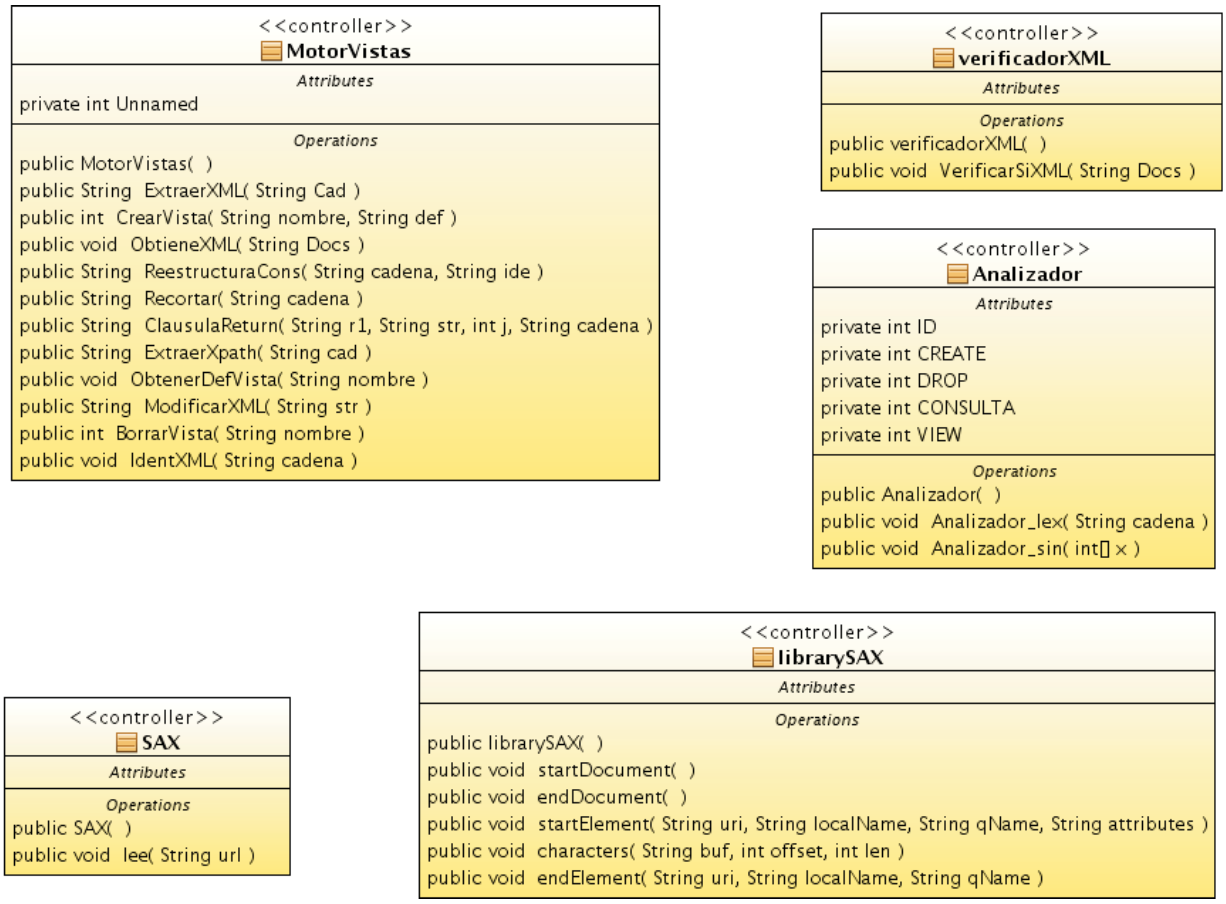

Figura 4.11: Diagrama detallado de clases de la capa de Negocio.

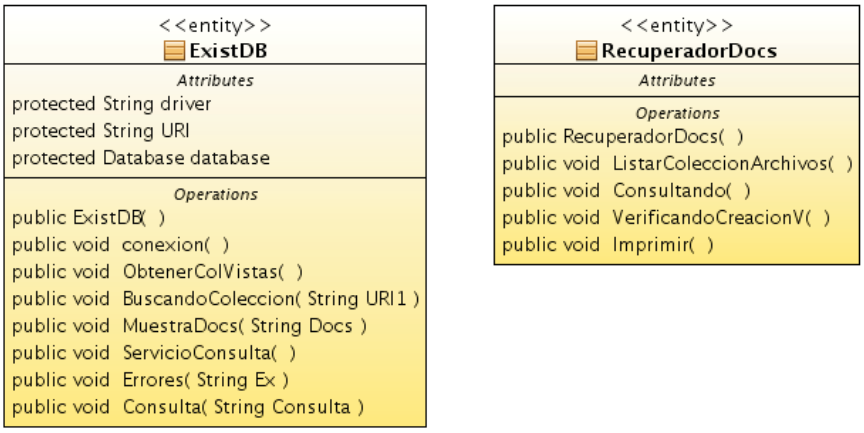

Figura 4.12: Diagrama detallado de clases de la capa de Control.

bién el intercambió de mensajes entre las clases y el orden en que se ejecutan las acciones. En la figura 4.15, se presentan las acciones que deben ser ejecutadas para poder crear una vista, los pasos a seguir ya fueron presentados en el caso de uso de la figura 4.5 de forma detallada y en este diagrama se muestra en forma más clara cual es el orden en que se dan estas acciones y en que momento se envían los mensajes de error y que clase es la encargada de hacerlo. El diagrama de secuencias mostrado en la figura 4.16, corresponde al caso de uso consultar vista por lo que se presenta la forma en la que interactúan cada una de las clases encargadas de hacer esta labor, los mensajes que intercambian para ello y los mensajes de error que se deben mostrar

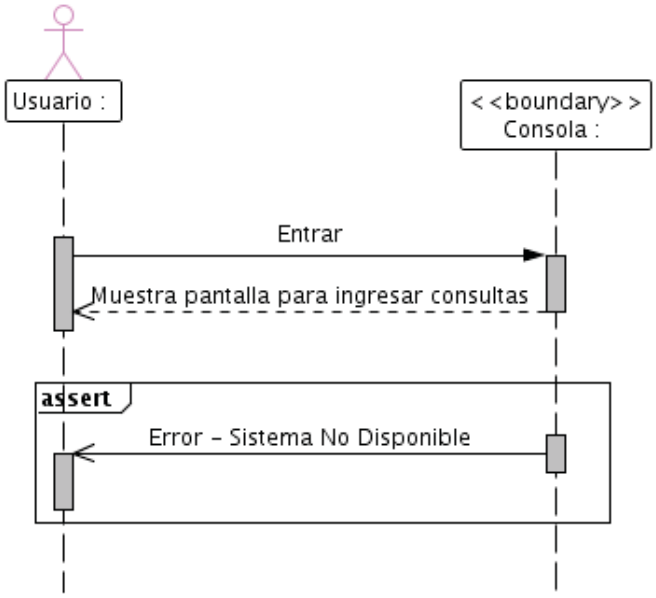

Figura 4.13: Diagrama de secuencia Entrar.

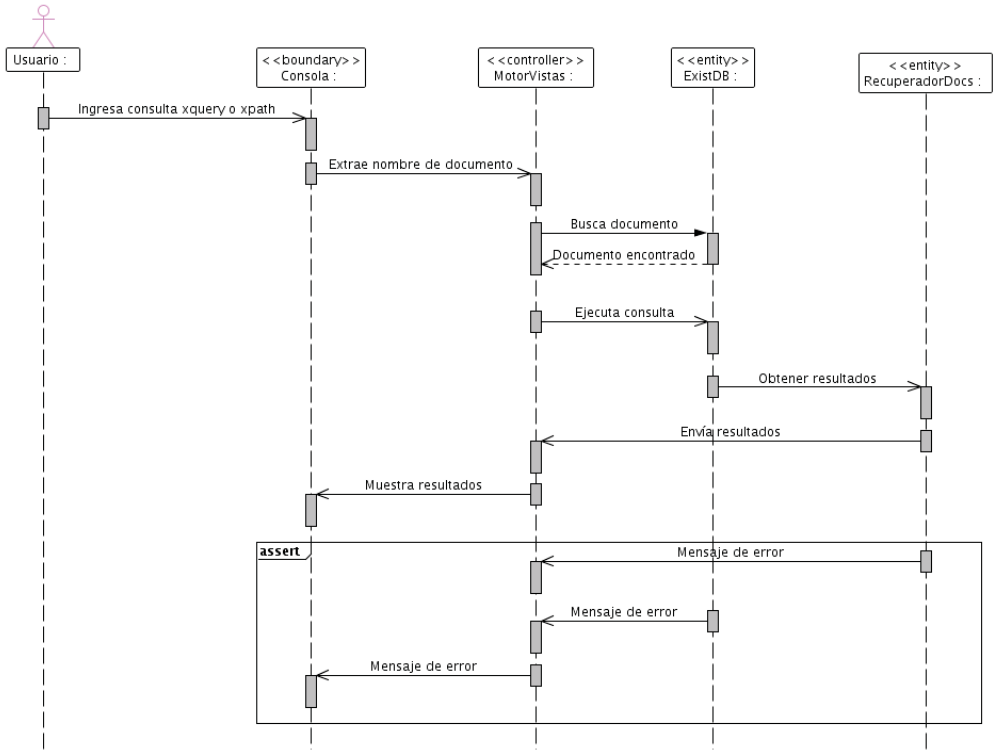

Figura 4.14: Diagrama de secuencia Hacer consulta XQuery ó Xpath.

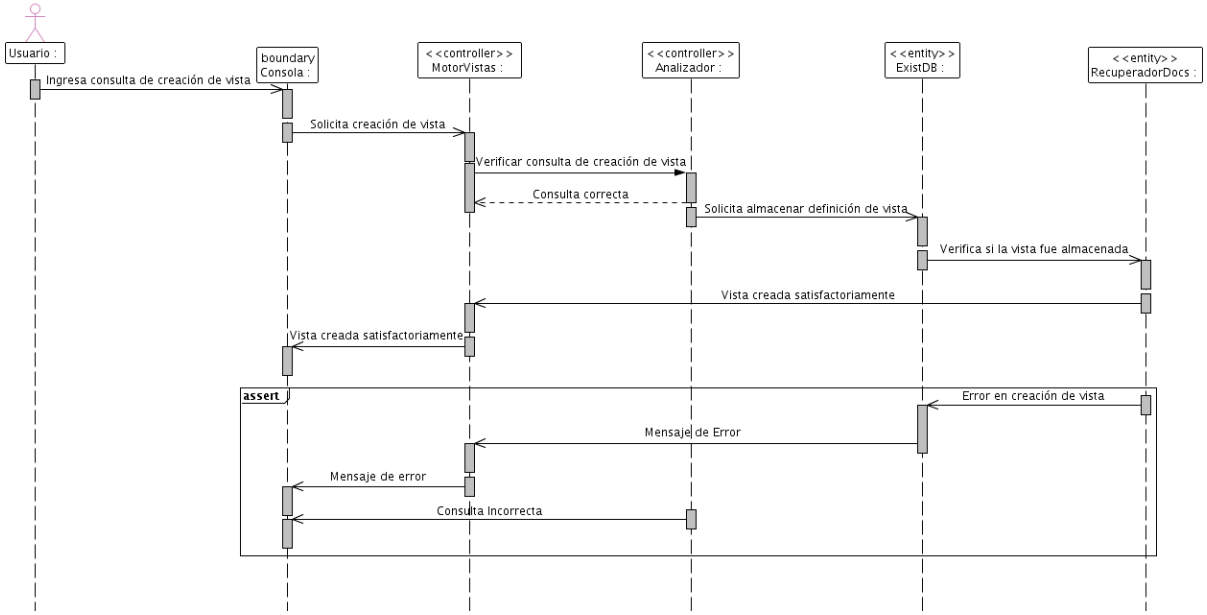

Figura 4.15: Diagrama de secuencia Crear vista.

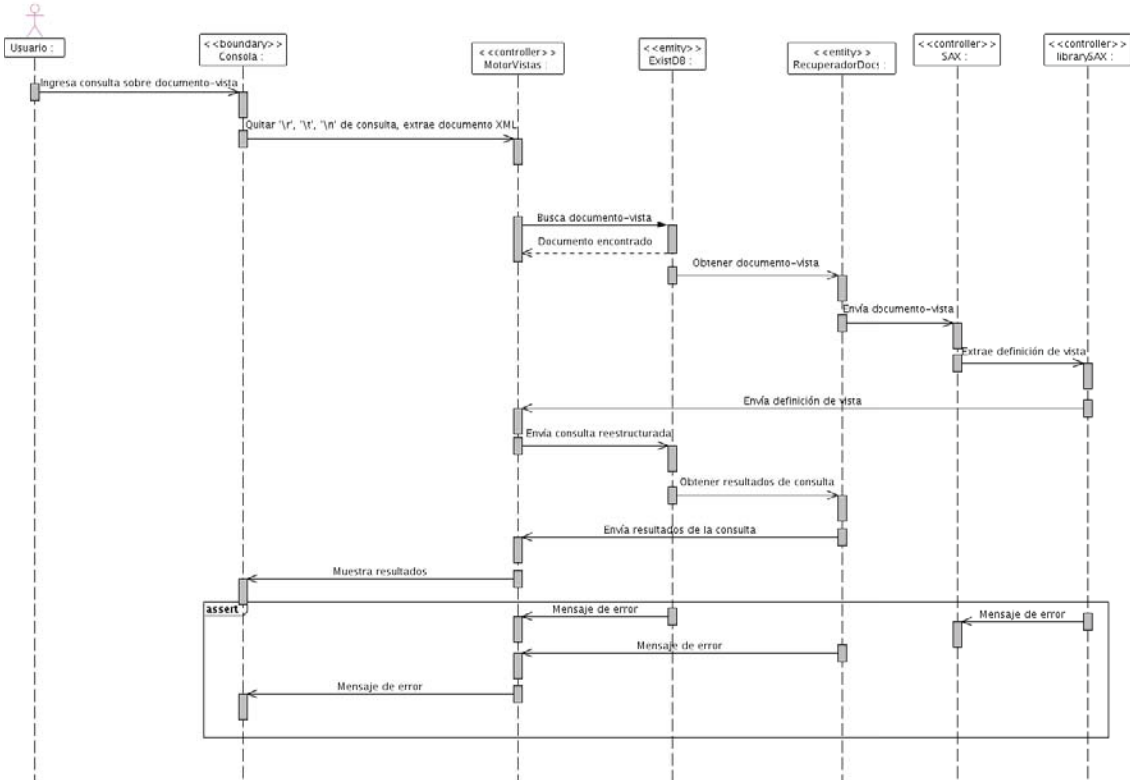

Figura 4.16: Diagrama de secuencia Consultar vista.

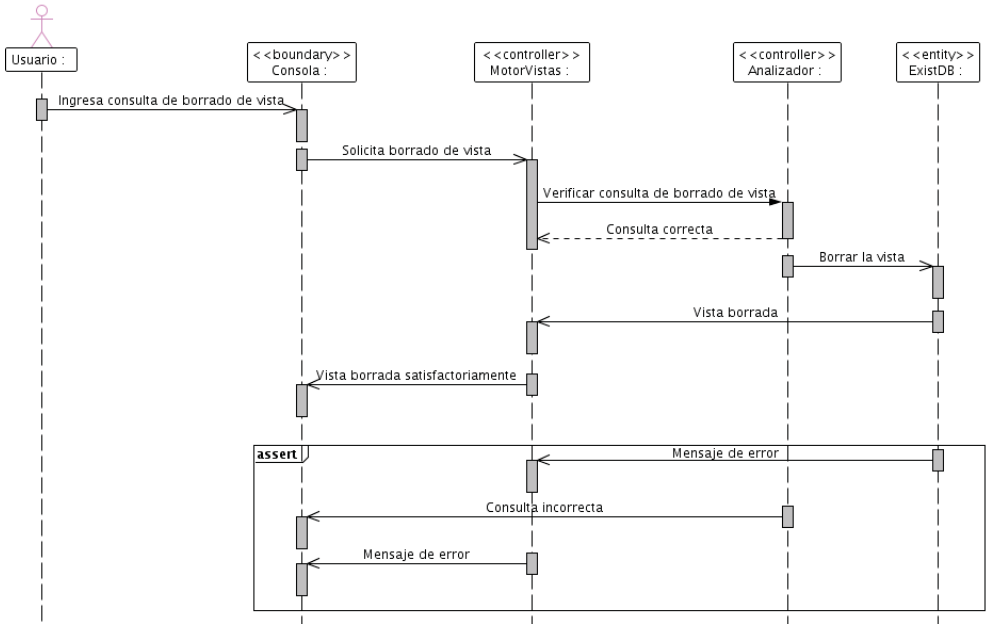

Figura 4.17: Diagrama de secuencia Borrar vista.

en el momento que la acción indicada no se pueda efectuar. La figura 4.17, muestra la secuencia en la que se ejecutan los pasos para poder borrar una vista en XMLView, la forma en la que interactúan las clases de todas las capas en la arquitectura y los mensajes de error que deben ser mostrados. El proceso de salir del sistema XMLView, se muestra en la figura 4.18, donde el usuario envía su petición de salir del sistema a la clase Consola y ésta se encarga de cerrar la aplicación o en caso de ocurrir un error informarlo al usuario.

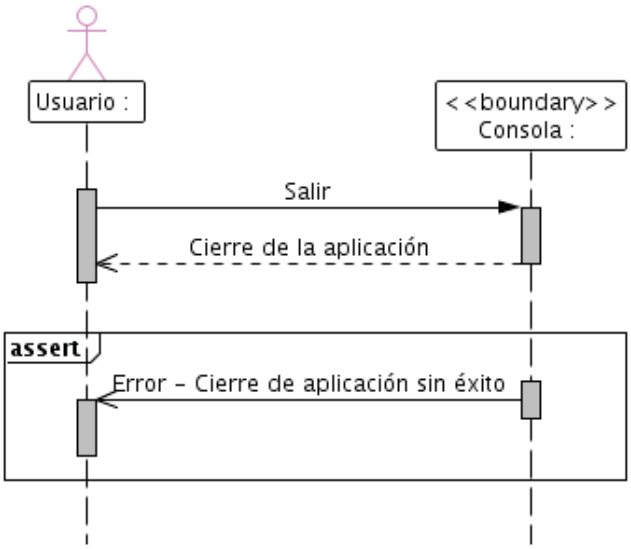

Figura 4.18: Diagrama de secuencia Salir.

5

Pruebas y Resultados

5.1 Plataforma de Implementación

En el capítulo anterior se mencionaron las características indispensables del diseño de XMLView, en este apartado se abordará un XMLView ya implementado, el cuál cuenta ya con la funcionalidad descrita brevemente en secciones anteriores.

El equipo de cómputo en el que se implementó XMLView tiene las siguientes características.

- Computadora con procesador Intel Core 2 Duo a 1.66 Ghz, disco duro de 100 GB y una memoria RAM de 1 GB.

Cabe aclarar que XMLView es un software muy ligero que pesa 300 MB, por lo cual es el espacio mínimo en disco duro con el que se debe contar para su almacenamiento. Además de los

aspectos de hardware es importante mencionar que en cuanto a software es necesario contar con la base de datos nativa eXist para que éste funcione adecuadamente, así como también la máquina virtual de java.

En la siguiente sección se presentarán los documentos XML que han sido utilizados para hacer las pruebas en XMLView y demostrar que éste funciona de la forma esperada.

5.2 Documentos XML almacenados en eXist

Debido a que XMLView es un software que trabaja sobre documentos XML almacenados en eXist, se crearon tres documentos XML diferentes para realizar las pruebas necesarias y verificar que el funcionamiento de éste sea el esperado. Cada documento XML usado en las pruebas se válida con su esquema correspondiente, en la figuras 5.1 a la 5.3 se presentan dichos esquemas y los documentos son mostrados en el Apéndice A.

En la siguiente sección se describirá como funciona internamente XMLView, se ejecutarán pruebas y se explicará el algoritmo que le da las funcionalidades para trabajar con las vistas. Para mejor comprensión de los resultados obtenidos en las pruebas vea los documentos Actores1.xml, Comentarios1.xml y Productos1.xml presentadas en el apéndice A, debido a que sobre éstos se ejecutan todas las consultas prueba.

5.3 Analizadores léxico y sintáctico

A lo largo de todo el análisis de la consulta ingresada, se hace uso de 2 analizadores, los cuáles permiten desde la verificación de tokens hasta la extracción de los mismos para posteriormente ser procesados por la clase MotorVistas.

XMLView utiliza el analizador incluido en eXist para verificar que una consulta XPath o XQuery sea correcta, así como también cuenta con dos analizadores propios: léxico y sintáctico. El léxico es necesario para identificar los tokens dentro de la consulta ingresada y el sintáctico para determinar si dicha consulta cuenta con los tokens necesarios para considerarse como válida. De tal forma que cuando se ingresa una consulta, lo primero que se hace es verificar si ésta inicia con alguno de los comandos agregados por XMLView, de ser así se manda a llamar a la clase Analizador en su método Analizador_lex() que recibe como parámetro la consulta, éste la analiza y va guardando el orden en el que aparecen los tokens dentro de la consulta, por ejemplo si la consulta es la siguiente:

Create View Prueba //Actores;

La clase Analizador cuenta con los atributos:

final int ID = 1;
final int CREATE = 2;
final int DROP = 5;
final int CONSULTA = 4;
final int VIEW = 3;

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="NOMBRE" type="xs:string"/>
 <xs:element name="CALLE" type="xs:string"/>
 <xs:element name="CIUDAD" type="xs:string"/>
 <xs:element name="TITULO" type="xs:string"/>
 <xs:element name="ANIO" type="xs:integer"/>
        <xs:element name="Actores"> <!--raiz -->
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="ACTOR" maxOccurs="unbounded"/>
                </xs:sequence>
        </xs:complexType>
        </xs:element>
        <xs:element name="DIRECCION">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="CALLE"  minOccurs="1" maxOccurs="1"/>
                        <xs:element ref="CIUDAD" minOccurs="1" maxOccurs="1"/>
                </xs:sequence>
        </xs:complexType>
        </xs:element>
        <xs:element name="PELICULA">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="TITULO"  minOccurs="1" maxOccurs="1"/>
                        <xs:element ref="ANIO" minOccurs="1" maxOccurs="1"/>
                </xs:sequence>
        </xs:complexType>
        </xs:element>
      <xs:element name="PELICULAS">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="PELICULA"  maxOccurs="unbounded"/>
                </xs:sequence>
        </xs:complexType>
        </xs:element>
        <xs:element name="ACTOR">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="NOMBRE" maxOccurs="unbounded"/>
                        <xs:element ref="DIRECCION" maxOccurs="unbounded"/>
                        <xs:element ref="PELICULAS" maxOccurs="unbounded"/>
                </xs:sequence>
        </xs:complexType>
        </xs:element>
</xs:schema>
```

Figura 5.1: Esquema XML que válida documentos que almacenan actores y las películas en las que estos han participado.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="TITULO_P" type="xs:string"/>
 <xs:element name="ANIO_P" type="xs:integer"/>
 <xs:element name="COMENTARIO" type="xs:string"/>
        <xs:element name="COMENTARIOS"> <!--raiz -->
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="COMENTARIO_PELICULA" maxOccurs="unbounded"/>
                </xs:sequence>
        </xs:complexType>
        </xs:element>
        <xs:element name="NOMINACIONES">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="NOMINACION" minOccurs="1" maxOccurs="unbounded"/>
                        <xs:element ref="RESULTADO" minOccurs="1" maxOccurs="1"/>
                </xs:sequence>
        </xs:complexType>
        </xs:element>
        <xs:element name="PREMIO">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="NOMINACIONES" minOccurs="1" maxOccurs="1"/>
                </xs:sequence>
                <xs:attribute ref="CATEGORIA"/>
                        <xs:simpleType>
                            <xs:restriction base="xs:string">
                                <xs:enumeration minOccurs="1"/>
                                <xs:enumeration maxOccurs="1"/>
                        </xs:restriction>
                    </xs:simpleType>
                <xs:attribute ref="ANIO_PREMIO"/>
                        <xs:simpleType>
                            <xs:restriction base="xs:integer">
                                <xs:enumeration minOccurs="1"/>
                                <xs:enumeration maxOccurs="1"/>
                        </xs:restriction>
                    </xs:simpleType>
        </xs:complexType>
        </xs:element>
        <xs:element name="PREMIOS">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="PREMIO" maxOccurs="unbounded"/>
                </xs:sequence>
        </xs:complexType>
        </xs:element>
        <xs:element name="COMENTARIO_PELICULA">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="TITULO_P"  minOccurs="1" maxOccurs="1"/>
                        <xs:element ref="ANIO_P" minOccurs="1" maxOccurs="1"/>
                        <xs:element ref="COMENTARIO" maxOccurs="unbounded"/>
                        <xs:element ref="PREMIOS" maxOccurs="unbounded"/>
                </xs:sequence>
        </xs:complexType>
        </xs:element>
</xs:schema>
```

Figura 5.2: Esquema XML que válida documentos que almacenan comentarios de las películas almacenadas en los documentos de la figura 5.1

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="DESCRIP" type="xs:string"/>
 <xs:element name="PRECIO" type="xs:double"/>
        <xs:element name="Ordenes"> <!--raiz -->
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="ORDEN" maxOccurs="unbounded"/>
                </xs:sequence>
        </xs:complexType>
        </xs:element>
        <xs:element name="PRODUCTO">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="DESCRIP"  minOccurs="1" maxOccurs="1"/>
                        <xs:element ref="PRECIO" minOccurs="1" maxOccurs="1"/>
                </xs:sequence>
        </xs:complexType>
        </xs:element>
        <xs:element name="ORDEN">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="PRODUCTO"  minOccurs="1" maxOccurs="unbounded"/>
                </xs:sequence>
                <xs:attribute ref="id"/>
                        <xs:simpleType>
                             <xs:restriction base="xs:string">
                                    <xs:enumeration minOccurs="1"/>
                                    <xs:enumeration maxOccurs="1"/>
                           </xs:restriction>
                        </xs:simpleType>
        </xs:complexType>
        </xs:element>
</xs:schema>
```

Figura 5.3: Esquema XML que válida documentos que almacenan ordenes y los productos que éstas contienen.

Estos son utilizados para que cuando el analizador recorra la consulta ingresada identifique uno a uno los tokens presentes en ésta, posteriormente almacena el identificador de cada token encontrado en un arreglo de 4 elementos, por ejemplo, para la consulta anterior, primeramente encontrará un Create por lo que almacenará al arreglo un 2, luego encontrará el token view guardará en el arreglo el 3, posteriormente encuentra un nombre de la vista que para este caso es Prueba, por lo que almacena un 1 y finalmente se encuentra una consulta XPath por lo que se almacena el 4 dentro del arreglo, una vez obtenido éste, se hace uso del método Analizador_Sin() ubicado en esta misma clase el cual analiza si los tokens fueron ingresados en la forma correcta, en caso de ser así se envía a la clase MotorVistas() la consulta y se verifica que ésta sea válida enviándola a eXist para que ésta la analice, porque si no es correcta se debe informar al usuario y se detiene el proceso en este caso de creación de la vista. Estos analizadores funcionan de forma similar para cuando se trata del borrado de una vista.

En caso de que la consulta ingresada sea de tipo XQuery y XPath se hace uso del mismo analizador léxico mencionado anteriormente, que es quién analiza la consulta para extraer el token correspondiente al documento con extensión xml presente en ella y posteriormente éste token es enviado a la clase MotorVistas, que es la encargada de enviar la petición de búsqueda a eXist para verificar si el documento encontrado es un docu-

mento XML o un documento–vista.

Otra de las funciones del analizador léxico que es importante para el funcionamiento de XMLView es que también es usado para extraer rutas XPath de las consultas sobre vistas, si se observa el algoritmo mostrado en la sección 5.4, se puede notar que es importante extraer la ruta XPath para la reestructuración de la consulta ingresada y mostrar al usuario lo que desea.

5.4 Algoritmo de XMLView

XMLView cuenta con un motor que le permite realizar las funcionalidades especificadas en las secciones anteriores: hacer consultas xQuey ó XPath además de crear, consultar y borrar vistas. Cuando se habla de un motor se refiere al algoritmo sobre el cuál se basa XMLView, éste fue creado desde cero para esta investigación y se presentarán ejemplos de su ejecución en las siguientes secciones de este capítulo.

*NOTA: Pasos para realizar la consulta sobre vistas (únicamente funciona para consultas en las que se hace referencia a un solo documento–vista a consultar):

1 Inicio

1.1 Se extrae desde la consulta ingresada el documento XML a consultar.

1.2 Se busca el nombre del documento en las colecciones almacenadas en eXist, excepto en la colección de vistas, si este coincide con alguno de los documentos almacenados significa que no es una vista.

1.2.1 Si el documento es un documento almacenado, ir paso 1.4.

1.3 Si el documento no está almacenado físicamente.

1.3.1 Se busca en la colección de vistas almacenadas en eXist, si la cadena es una vista ir al paso 1.3.2, si no, se envía mensaje de que el documento consultado no existe e ir al paso 3.

1.3.2 Si la cadena corresponde al nombre de un documento–vista, se debe reestructurar la consulta ingresada:

1.3.2.1 Se obtiene la definición de la vista desde eXist, si ésta no tiene ninguna cláusula for (es un XPath) ir al paso 1.3.2.1.3, si no:

1.3.2.1.1 Se empieza a recorrer la definición de la vista, para sustituir cada clausula for existente en ella en una nueva consulta reestructurada. En este paso comienza el proceso de reestructuración.

1.3.2.1.2 Se prosigue recorriendo la definición dela vista y todas las cláusulas antes del return también se añaden a la consulta reestructurada.

1.3.2.1.2.1 Cuando se encuentra la cláusula return en la definición de la vista se añade a

la consulta reestructurada.
1.3.2.1.2.2 Posteriormente se recorre la consulta ingresada por el usuario para buscar la etiqueta de apertura que encierra el resultado (ubicada después del return) en caso que ésta exista y se añade a la consulta reestructurada.
1.3.2.1.2.2.1 Si se encuentra más de una variable se termina la reestructuración e ir a paso 3. éste algoritmo aún no funciona para consultas que tengan más de una variable.
1.3.2.1.2.3 Se agrega la variable correspondiente a la consulta reestructurada.
1.3.2.1.2.4 Se concatena el último nodo de la ruta XPath encontrado en la clausula for de la consulta ingresada por el usuario a la variable agregada en el paso anterior y también se agrega a la consulta reestructurada.
1.3.2.1.2.5 Se añade la etiqueta de cierre, en caso de que ésta exista a la consulta reestructurada.
1.3.2.1.2.6 Ir a paso 1.4
1.3.2.1.3 Se verifica que la consulta ingresada por el usuario únicamente contenga clausulas for, ya que no se contemplan en la

reestructuración las demás clausulas que permite XQuery, de ser así pasar al paso siguiente, si no ir a 1.3.

1.3.2.1.4 Se aãde la cláusula for de la consulta ingresada por el usuario hasta encontrar la cadena "in" en la nueva consulta reestructurada.

1.3.2.1.5 Posteriormente se obtiene la ruta XPath almacenada en la definición de la vista y se agrega a la consulta reestructurada.

1.3.2.1.6 Se extrae toda ruta XPath encontrada en la consulta ingresada por el usuario y se concatena a la ruta agregada en el paso anterior en la consulta reestructurada. En este paso se debe verificar que la ruta sea correcta resultante de acuerdo al lenguaje XPath.

1.3.2.1.7 Se añade a la consulta reestructurada la cláusula return de la consulta ingresada por el usuario con su etiqueta de apertura en caso de existir.

1.3.2.1.8 Se obtiene de la consulta ingresada por el usuario la variable que otorga el resultado y se añade a la consulta reestructurada.

1.3.2.1.9 Se extrae desde la consulta ingresada por el usuario la etiqueta de cierre, en caso de que exista y se añade a la consulta reestructurada.

1.3.3 Fin
1.4 Se envía la consulta, reestructurada o no, a la base de datos eXist, para que la ejecute.
1.5 Presentar el resultado en pantalla al usuario.
2 Iniciar nueva consulta.
3 Fin.

5.4.1. Pruebas

En esta sección se explicará a detalle cómo funciona XMLView y se darán ejemplos de la ejecución del algoritmo presentado en 5.4

Es importante mencionar que en las consultas ingresadas sobre vistas se pueden tener los siguientes casos:

- Que haya un solo documento XML en la consulta ingresada y éste se encuentre almacenado físicamente en eXist.

- Que haya un solo documento XML en la consulta ingresada y éste corresponda a un documento–vista.

- Que haya más de un documento XML en la consulta ingresada y que todos ellos se encuentren almacenados físicamente en eXist.

- Que haya más de un documento XML en la consulta ingresada y que alguno de ellos corresponda a un documento–vista.

El algoritmo sólo contempla los dos primeros casos, porque como se ha dicho anteriormente XMLView cuenta con algunas limitaciones, las cuales son propuestas como trabajo futuro.

La forma en la que XMLView trabaja sobre vistas, es analizando cada consulta que ingresa a la aplicación:

- Si la consulta ingresada es una consulta XPath, se ejecutará enviando ésta a eXist y los resultados que eXist devuelva serán mostrados en la pantalla de la aplicación.

- Si la consulta que ingresa el usuario está escrita en XQuery, se debe verificar que el documento a consultar no corresponda a un documento–vista. En caso de ser documento–vista, se verifica que sea el único documento en la consulta y se reestructura de tal forma que eXist la pueda entender, para esto se utiliza el algoritmo presentado en la sección anterior. Si no es un documento–vista, se procede a enviar la consulta a eXist y los resultados que ésta devuelve son mostrados en pantalla al usuario.

- Si la consulta ingresada inicia con la palabra Create, (en mayúsculas, minúsculas o una combinación de ambas), se verifica que los tokens estén en el lugar adecuado y

que se contenga todos los necesarios para la creación de la vista.

- Si la consulta deseada inicia con la palabra Drop, significa que el usuario desea borrar una vista y de igual forma que el Create ésta sentencia se reconoce en cualquiera de sus variantes. Para ejecutar la consulta se verifica que todos los tokens necesarios estén ubicados correctamente. Para poder borrar una vista se deberá buscar que el documento a borrar realmente sea una vista, de no ser así se informará al usuario que el documento no se encontró.

5.4.1.1. Consultas XPath y XQuery sobre documentos almacenados en eXist

Como todo software que permite hacer consultas a una base de datos, XMLView permite hacer consultas básicas, llamando así aquellas que desean obtener información de documentos XML almacenados físicamente en la base de datos.

Para realizar una consulta en XMLView ésta tendrá que ser agregada en el área de trabajo y después presionar el botón "Send". Si la consulta es correcta sintácticamente se verán los resultados en el panel de la parte inferior ubicado en la pantalla del software, en caso contrario será notificado en éste mismo el error. Por ejemplo, la siguiente consulta es un XPath que extrae las direcciones de los actores almacenados en el documento Ac-

tores1.xml (Para ver la ejecución en XMLView observe la figura A.1, que se encuentra en el apéndice A):

//Actores/ACTOR/DIRECCION;

Cuando se realiza la consulta en XMLView y existen muchos registros resultantes, no se podrá percibir en la pantalla el resultado completo, por lo que para ver todos estos basta con deslizar la barra de desplazamiento ubicada en el área de resultados.

Ahora se ejecutará una consulta que muestra aquellas órdenes con id = "r1" ó id = "r3", en caso de que ambos id existieran se mostrarán los dos resultados, para estos fines se usará el documento en el apéndice A llamado Productos1.xml y la ejecución de esta consulta en XMLView se puede observar en la figura A.2. La consulta se presenta a continuación:

/child::Ordenes/child::orden[@id="r1" or @id="r3"];

Una vez presentados algunos de los tipos de consultas XPath que se pueden hacer en XMLView se abordarán consultas escritas en XQuery, por ejemplo, la siguiente extrae los nombres de los actores almacenados en el documento Actores1.XML:

```
for $p in //Actores
return
    <Actores>
        $p/ACTOR/NOMBRE
    </Actores>
```

El ejemplo de ejecución de esta consulta se muestra en la

figura A.3.
Si se analiza la siguiente consulta se puede observar que las sentencias XQuery pueden ser tan complejas como las del SQL, por ejemplo, ésta obtiene los comentarios hechos para las películas cuyo título en el documento Comentarios1.xml y en Actores1.xml es el mismo, esto es similar a un join en SQL. Para ilustrar que XMLView también cuenta con detección de errores se ingresará un error de forma arbitraria para analizar que hace el software en estos casos:

```
    for $todo in doc("/db/viewXML/Actores1.xml")//PELICU–
    LA/TITULO,
    $peliculas in doc("/db/viewXML/Comentarios1.xml")//CO–
    MENTARIOS_PELICULA
where $todo = $peliculas/TITULO_P
    return
        <comentario>
            $todo, $peliculas/COMENTARIO
        <comentario>
```

El error es que existen dos etiquetas de apertura <comentario> por lo que la que esta remarcada debería estar así </comentario>, veáse la figura 5.4 para observar que hace el software en este caso.

Como se observa en la figura 5.4 XMLView muestra el siguiente error: "expecting XML end tag [at -line 5, column -63] [at

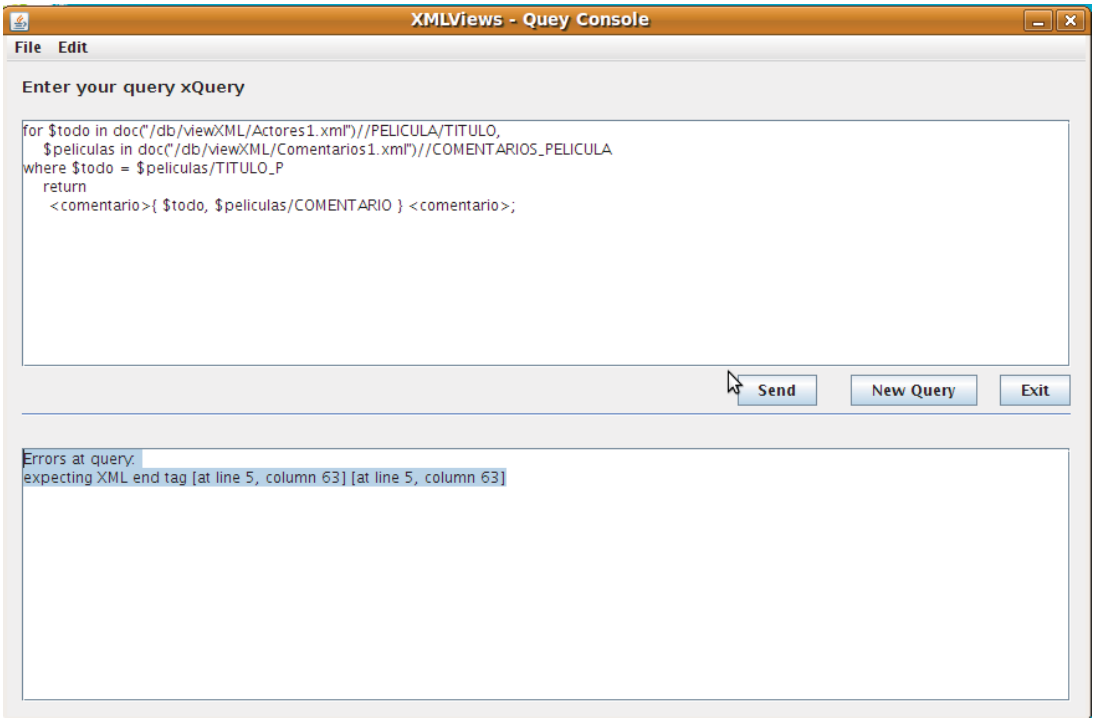

Figura 5.4: Consulta XQuery que desea obtener los comentarios de películas y XMLView detecta errores.

line 5, column 63]", lo cuál significa que hace falta una etiqueta de cierre en la consulta.

Ahora se corregirá el error y se mostrará en la figura A.4 como el software devuelve los resultados encontrados desde los documentos Actores1.xml y Comentarios1.xml.

5.4.1.2. Creación de vistas

Continuando con la descripción de las funcionalidades de XMLView, en esta sección se presenta la forma en la que éste crea vistas a partir de una consulta sobre documentos XML, además de describir lo que hace internamente para poder hacer dicha creación.

Para que XMLView permita la creación de vistas, el usuario debe introducir una sentencia con el siguiente formato:

Create View Nombre_vista definición_de_la_vista;

Cuando esta sentencia es agregada XMLView comienza el proceso de creación, primeramente se tiene que verificar que la sentencia de Create este correcta sintácticamente, para esto se cuenta con los analizadores léxico y sintáctico abordados anteriormente; si es correcta se almacena el nombre de la vista para ser usado posteriormente. Posteriormente se prosigue a verificar que la definición de la vista sea sintácticamente correcta, si ésta es correcta se retoma el nombre de la vista y se verifica que esta no sea una vista existente, para esto se tiene que verificar los documentos–vista en eXist, en caso de existir se deberá infor-

mar al usuario de ésta situación, debido a que XMLView aún no considera la opción de modificación de una vista previamente creada; en caso contrario se creará un documento–vista XML, el cuál contiene únicamente las siguientes etiquetas:

<view>
    <nombre> Nombre_vista </nombre>
    <definicion> Definición_de_la_vista </definicion>
</view>

Se le llama documento–vista porque éste almacenará los datos correspondientes a la vista, su nombre en la etiqueta "nombre" y la definición de la misma en "definicion". Cabe aclarar que en ningún momento se materializa un documento con la definición de la vista, sólo se almacenan los datos correspondientes a su definición para que cuando se consulte se haga sobre los resultados que se obtienen al ejecutar esta consulta y es importante almacenar el nombre de la vista para poder consultar sobre ésta.

El documento–vista se puede considerar como un documento virtual, por que no es un documento XML que cuenta con datos y está físicamente almacenado en eXist, si no que es un documento como se dijo anteriormente, que guarda la consulta que constituye la definición de la vista. Una vez creado éste se tiene que almacenar en la base de datos en una colección llamada Views que es donde se almacenan las vistas, estando ahí se puede consultar en el momento que se desee.

Es importante mencionar que las vistas pueden ser creadas a partir de consultas de tipo XPath y XQuery, como se muestra

en las figuras A.5 y A.6 presentadas en el apéndice A.

La vista creada en la figura A.5, tiene una definición que obtiene todas las películas que fueron creadas después del año 1999. De acuerdo al funcionamiento interno de XMLView, se creará un documento–vista con el valor del elemento "nombre" PELICULAS_1999.xml y un elemento "definición" con contenido "//Actores/ACTOR/PELICULAS/PELICULA[ANIO>1999]", por lo tanto, el documento de definición de la vista se vería así:

```
<view>
    <nombre>PELICULAS_1999.xml</nombre>
    <definicion>//Actores/ACTOR/PELICULAS/PELICULA[A–
    NIO>1999]</definicion>
</view>
```

Al nombre de la vista como se puede ver se le agregó la extensión .xml al almacenarla en el documento–vista, esto se realiza con la intención de que al ser consultada se haga de la forma en la que se consultaría cualquier otro documento almacenado físicamente en eXist.

Cada vez que se crea una vista se debe verificar que la vista no haya sido creada previamente, en este caso se le informa al usuario de que la vista fue creada satisfactoriamente como se ve en la figura A.6, en caso contrario se enviará un mensaje de error que informará al usuario de que la vista ya existe

y que deberá borrarla o bien cambiar el nombre de la misma. De igual forma que la creación de vistas con XPath, se almacena la definición en un documento–vista, pero en este caso la definición comenzará con una clausula for, como se observa a continuación:

```
<view>
    <nombre>TITULOS.xml</nombre>
    <definicion> for $todo in doc("/db/viewXML/Actores1.xml")–
    //PELICULA/TITULO
        return
            <Titulo> $todo </Titulo>
            </definicion>
</view>
```

5.4.1.3. Consulta sobre vistas en XPath

En el algoritmo visto en la sección 5.4, se informa que cuando a vistas se refiera, únicamente se podrá consultar sobre un documento–vista, proponiéndose mejoras en el apartado de trabajos futuros.

Como se determina en el algoritmo, para poder ejecutar la consulta ingresada por el usuario, una vez obtenido el documento XML en ella, se prosigue a verificar en eXist si éste se encuentra físicamente almacenado, corresponde a la definición de una vista o en su defecto no existe. Si no corresponde a una vista, entonces se envía la consulta directamente a eXist para

que la ejecute; en caso de ser una vista la que se esta consultando, se debe reestructurar y luego enviar a eXist para su ejecución, para estos dos casos, eXist devolverá al software los resultados quién se encargará de mostrarlos en pantalla al usuario. Si el documento no existe, se enviará un mensaje al usuario indicando que el documento no existe. Uno de los aspectos más importantes en la consulta sobre vistas es que para poder obtener el resultado correcto, ésta debe ser reestructurada a la entrada debido a que si se envía a eXist tal como ingresa, luego de verificarse que sea correcta sintácticamente, eXist devolverá los resultados de la misma, los cuales no serán los esperados por el usuario, porque recordando cada documento–vista solo contiene el nombre de la vista y la definición de la misma, no los datos correspondientes a ésta.

Al identificar que el documento a consultar es una vista, XMLView se prepara para hacer la reestructuración de la misma. Una vez que ya se tiene determinado que el documento a consultar es una vista, se prosigue a la reestructuración de la consulta. Inicialmente se debe extraer la definición de “definicion”, a esta definición le llamaremos C. Posteriormente XMLView verifica que lenguaje se utilizó en esta, si esta escrita en XPath o en XQuery ya que serán tratadas de forma distinta según el algoritmo. En caso de ser una consulta de tipo XPath, se obtiene Q que fue la consulta sobre la vista, para poder ir agregándola poco a poco como dicta el algoritmo a la nueva consulta reestructurada a la cual llamaremos r1. Tomando en

cuenta que las consultas sobre las vistas solo pueden ser formuladas usando XQuery, se debe extraer desde Q toda sentencia for hasta encontrar la palabra "in" en dicha consulta y añadirla a r1 (aquí comienza la reestructuración). Luego se debe extraer la ruta XPath almacenada en C y agregarla a r1, tomando en cuenta la sintaxis de una ruta XPath para que no haya errores, después también se agrega la ruta XPath en Q y agregarla a lo que se tiene en r1, aquí también se verifica que la sintaxis XPath siga siendo válida. Una vez hecho esto se procede a añadir la clausula return de Q a r1, agregando sus respectivas etiquetas y variables en ella. Cuando r1 ya esta completamente formada, se procede a enviar esta nueva consulta a eXist, la cual verifica que la consulta reestructurada sea válida sintácticamente y devuelve los resultados, mismos que son tomados por XMLView y encerrados entre una etiqueta raíz, la cual tendrá el nombre de la vista a consultar, esto asegura que el documento resultante de consultar una vista, sea bien formado.

A continuación se presentan dos ejemplos de como se hace la reestructuración de una consulta sobre vistas creadas a partir de una definición Xpath.

Ejemplo 1:
Retomando el ejemplo de creación de vistas de la figura A.8, se tiene lo siguiente:
CREATE VIEW PELICULAS_1999 //Actores/ACTOR/PELICULAS/PELICULA[ANIO>1999].

Ahora se desea consultar sobre ésta con la siguiente sentencia (para ver la ejecución de ésta ir a la figura A.7):

```
for $n in doc("PELICULAS_1999.xml")
    return $n/TITULO;
```

Siguiendo las instrucciones del algoritmo:

1. Se extrae desde la consulta ingresada el documento XML a consultar.

2. Se verifica si el documento PELICULAS_1999.xml, corresponde a un documento almacenado físicamente en eXist o a un documento–vista.

3. Se determina que PELICULAS_1999.xml es un documento–vista.

4. Se obtienen desde eXist el documento correspondiente a PELICULAS_1999.xml y se recorre con SAX para poder extraer la información del elemento "definicion", ya que ésta será utilizada en la reestructuración. A la definición de la vista obtenida le llamaremos C.

5. Posteriormente se evalúa que el lenguaje de consulta de la definición sea XPath.

6. Se extrae desde Q la sentencia for existente hasta la palabra "in" y se añade a la nueva sentencia r1 que será la consulta reestructurada. Como se observa a continuación:
   for $n in —->r1

7. Luego se deberá extraer desde C la ruta XPath y agregarla también a r1, quedando de la siguiente forma:
   for $n in //Actores/ACTOR/PELICULAS/PELICULA[ANIO>1999] —->r1

8. Ahora se debe agregar la ruta XPath en la consulta Q, en caso de que ésta existiera, por lo tanto la consulta quedaría igual que el paso anterior, puesto que el XPath TITULO está contenido en una variable $n y no en la clausula for como lo dicta el algoritmo, por lo que XMLView no lo toma en cuenta:
   for $n in //Actores/ACTOR/PELICULAS/PELICULA[ANIO>1999]–>r1

9. Prosiguiendo con la reestructuración, se debe añadir la clausula return en Q con sus respectivas etiquetas y variables en caso de que existieran alguna de ellas:
   for $n in //Actores/ACTOR/PELICULAS/PELICULA[ANIO>1999]/TITULO return $n

10. Una vez que r1 está completamente formada, ya se puede enviar a eXist y los resultados que ésta devuelva serán mostrados dentro de una etiqueta raíz que XMLView agrega con el nombre de la vista.

A continuación se muestran los resultados que son mostrados al usuario una vez que se ejecuta la consulta sobre la vista PELICULAS_1999.

<PELICULAS_1999>
<TITULO>Before night falls</TITULO>
<TITULO>Blow</TITULO>
<TITULO>Pirates of the Caribbean: The curse of the Black Pearl</TITULO>
<TITULO>Secret window</TITULO>
<TITULO>The Libertine</TITULO>
<TITULO>Charlie And The Chocolate Factory</TITULO>
<TITULO>Pirates of the Caribbean: Dead Man's Chest</TITULO>
<TITULO>Pirates of the Caribbean: At world's end</TITULO>
<TITULO>Gonzo: The Life and Work of Dr. Hunter S. Thompson</TITULO>
<TITULO>The Mexican</TITULO>
<TITULO>Ocean's Eleven</TITULO>
<TITULO>Confessions of a Dangerous Mind</TITULO>
<TITULO>Troy</TITULO>
<TITULO>The Curious Case of Benjamin Button</TITULO>
<TITULO>The Adventures of Rockyamp;Bullwinkle</TITULO>
<TITULO>15 minutes</TITULO>
<TITULO>City by the Sea</TITULO>
<TITULO>Meet the fockers</TITULO>

<TITULO>Hide and Seek</TITULO>
<TITULO>Stardust</TITULO>
<TITULO>What Just Happened?</TITULO>
<TITULO>Nurse Betty</TITULO>
<TITULO>Along Came a Spider</TITULO>
<TITULO>High Crimes</TITULO>
<TITULO>Levity</TITULO>
<TITULO>Million Dollar Baby</TITULO>
<TITULO>March of the Penguins</TITULO>
<TITULO>Lucky Number Slevin</TITULO>
<TITULO>Evan Almighty</TITULO>
<TITULO>Wanted</TITULO>
<TITULO>The Others</TITULO>
<TITULO>The Hours</TITULO>
<TITULO>Cold Mountain</TITULO>
<TITULO>Birth</TITULO>
<TITULO>The interpreter</TITULO>
<TITULO>Happy Feet</TITULO>
<TITULO>The Golden Compass</TITULO>
<TITULO>Australia</TITULO>
<TITULO>Erin Brockovich</TITULO>
<TITULO>Ocean's Eleven de Steven Soderbergh</TITULO>
<TITULO>Confessions of a dangerous mind</TITULO>
<TITULO>Mona Lisa Smile</TITULO>
<TITULO>Closer</TITULO>
<TITULO>Charlotte's Web</TITULO>

<TITULO>Charlie Wilson’s war</TITULO>
<TITULO>Fireflies in the Garden</TITULO>
<TITULO>Hanging Up</TITULO>
<TITULO>Kateamp;Leopold</TITULO>
<TITULO>In the Cut</TITULO>
<TITULO>Against the Ropes</TITULO>
<TITULO>In the land of women</TITULO>
<TITULO>My mom’s new boyfriend</TITULO>
<TITULO>The Women</TITULO>
<TITULO>Men of Honor</TITULO>
<TITULO>Sweet November</TITULO>
<TITULO>Trapped</TITULO>
<TITULO>Monster</TITULO>
<TITULO>Head in the Clouds</TITULO>
<TITULO>North Country</TITULO>
<TITULO>In the Valley of Elah</TITULO>
<TITULO>Hancock</TITULO>
<TITULO>Battle in Seattle</TITULO>
</PELICULAS_1999>

Ejemplo 2:

Supongamos que se desea hacer una consulta de todas las descripciones de los productos que son parte de la orden con id = “r2”, almacenadas en la vista ORDEN, cuya creación se presenta a continuación:

CREATE VIEW ORDEN /child::Ordenes/child::orden;

Y la consulta quedaría de la siguiente forma:

for $n in doc (“ORDEN.xml”)/child::orden[attribute::id=“r2”]-
/child::producto/descrip
return
<Orden2>$n</Orden2>;

XMLView ejecuta la consulta (Figura A.8) y muestra el siguiente resultado:

```
<ORDEN>
    <Orden2>
        <descrip>Korn Flakes</descrip>
    </Orden2>
</ORDEN>
```

Estas son algunas de las consultas que XMLView puede ejecutar con XPath, en el siguiente tema se presentarán algunos tipos de consultas que se pueden realizar utilizando como lenguaje de definición de la vista XQuery.

5.4.1.4. Consulta sobre vistas con XQuery

La consulta sobre vistas cuya definición fue escrita en XQuery, realiza en parte los pasos vistos en la sección anterior. De tal forma que al ingresar la consulta sobre la vista también se debe extraer el documento XML en la consulta, una vez identificado que éste corresponde a un documento–vista se prosigue con la reestructuración de la consulta.

Cuando ya se tiene el nombre del documento–vista se debe extraer la definición de la vista almacenada en la colección Views en eXist, utilizando SAX para obtener la información del elemento "definicion" que corresponde a la definición de la vista. Siguiendo la convención adoptada en la sección anterior a ésta definición le llamaremos C y la consulta ingresada por el usuario será denominada como Q. Para hacer la reestructuración cuando C es una consulta de tipo XQuery, se empieza a recorrer C y se sustituye toda clausula for en r1, siendo r1 la consulta reestructurada. Posteriormente se continua el recorrido de C, se agrega a r1 toda clausula y condiciones antes del return. Cuando se encuentra la clausula return se agrega a r1 al mismo tiempo que se agrega la etiqueta de apertura. Luego se deberá recorrer Q para observar si ésta contiene únicamente una variable, en caso de que existan más de una, no podrá hacerse la reestructuración sugiriendo este problema como trabajo futuro. Una vez obtenida la variable se deberá agregar a r1. Siguiendo con la reestructuración se debe extraer el último nodo de

la ruta XPath en Q y concatenar éste a la variable, quedando de la siguiente forma: $variable//Ruta_XPath. Ahora solo falta agregar la etiqueta de cierre de Q en r1. Una vez teniendo r1 se deberá enviar a eXist para que ésta verifique si es correcta sintácticamente y enseguida otorgue los resultados de la misma a XMLView, el cual los mostrará al usuario en el área de resultados, agregando al resultado un nodo raíz correspondiente al nombre de la vista consultada.

Para ejemplificar lo que se ha dicho en el párrafo anterior se tiene la siguiente consulta:

```
for $todo in doc("/db/viewXML/Actores1.xml")//PELICULA
return $todo
```

Esta consulta extrae desde el documento Actores1.xml todas los elementos dentro del elemento "PELICULA". Ahora se desea crear una vista que incluya en su definición esta consulta XQuery (veáse figura A.9). Cuando se desea consultar información sobre la vista a la cual llamaremos PELICULAS, basta con que el usuario introduzca consultas de este tipo:

```
for $n in doc ("PELICULAS.xml")/TITULO
return $n;
```

En esta consulta se obtienen los resultados observados en el

apéndice A (Página 96).

```
    for $n in doc (“PELICULAS.xml”)/ANIO
    return <AÑO> $n </AÑO>;
for $n in doc (“PELICULAS.xml”)
    return <AÑO>$n/ANIO </AÑO>;
```

Las dos consultas anteriores presentan los mismos resultados, es decir, todos los años encontrados en el documento Actores1.xml, tomando en cuenta los que se encuentran repetidos.

5.4.1.5. Borrado de vistas

XMLView también proporciona la posibilidad de borrar una vista creada previamente. La intrucción de borrado es muy sencilla, basta con ingresar los comandos DROP VIEW y el nombre de la vista. Por ejemplo si desea borrarse la vista PELICULAS_1999, creada en la sección 5.4.1.2, se tendría que ingresar:
DROP VIEW PELICULAS_1999;
Para observar como se ejecuta esta sentencia en XMLView, veáse la figura A.10.
En el caso de que se desee borrar una vista que no ha sido creada, XMLView mostrará un mensaje de error indicando que la vista no existe. Es importante mencionar que para el borrado no se debe escribir la extensión xml del documento, para evi-

tar confusiones al usuario ya que al crear esta vista tampoco es necesario ingresar esta extensión. Sin embargo al consultarse una vista, si es importante ingresar la extensión, porque se recuerda que el usuario al consultar sobre está, lo hace sobre un documento que para él existe físicamente almacenado.

5.4.1.6. Características de las Vistas resultantes

XMLView proporciona como resultado de una consulta sobre vistas, un documento bien formado, situación que puede corroborarse en los resultados de la consultas presentes en el Apéndice A , en la cuál se observa que los elementos obtenidos de la consulta son incrustados dentro de un nodo raíz, el cual corresponde al nombre de la vista consultada. Con esto se esta asegurando que todos los documentos que se obtengan como resultado de una consulta sobre vistas, serán bien–formados.

Otro de los aspectos característicos de las consultas sobre vistas es que al igual que las bases de datos relacionales, al almacenar la definición de la vista, se obtendrá información actualizada al realizar la consulta sobre éstas, ya que cada vez que se hace referencia a ellas en una consulta, se vuelve a ejecutar ésta, asegurando que sí el documento base ha sido modificado, estos cambios serán reflejados en el resultado. Por lo que el usuario puede tener la seguridad de que los resultados obtenidos están siendo íntegros con respecto a lo que se tiene almacenado en la base de datos.

6 Conclusiones

6.1 Conclusiones

En este trabajo se investigó acerca de las propuestas de creación de vistas sobre documentos XML existentes hasta hoy en día, obteniéndose de ello diversos mecanismos, entre éstos se encuentran: XPERANTO, que crea vistas XML sobre información almacenada en bases de datos relacionales y XViews que ya aborda la creación de vistas sobre documentos XML, ambas son ideas sólidas en cuanto a éste tema, sin embargo, estas no han resuelto el problema, porque aún no han sido puestas en producción. XMLView que es la propuesta de ésta tesis, difiere de éstos mecanismos porque no materializa la vista cuando se necesita consultar sobre ella, permite que la definición de una vista sea escrita en XPath o XQuery y crea vistas sobre documentos XML almacenados en bases de datos nativas, entre otros

aspectos importantes analizados a lo largo de este trabajo.

Se partió de la idea de vistas del modelo relacional, en el cuál se dice que una vista es una tabla "virtual", porque no existe físicamente almacenada, sin embargo, cuando una vista es creada posteriormente puede ser consultada y el usuario jamás se percata que está consultado sobre algo virtual. Por lo que retomando esto, se pensó en crear un mecanismo que pudiera crear vistas, es decir, documentos XML virtuales, los cuáles pudieran ser consultados como si estuvieran físicamente almacenados. Ahora bien, como lenguajes de consulta sobre XML, se cuentan con XPath y XQuery, por lo que se hizo uso de estos para la administración de vistas.

Se desarrolló así XMLView, una herramienta para la creación de vistas, que mantiene la idea del modelo relacional, donde se implementó que al crearse una vista, había que almacenar el nombre y su definición para que cuando ésta fuera consultada, se obtuviera esa definición, se reestructurará con la consulta ingresada y los resultados obtenidos correspondieran a la información almacenada en los documentos base en cualquier momento. Con lo cual se cumplió el objetivo principal de éste trabajo de tesis.

El algoritmo presentado en la sección 5.4, es el que permite a XMLView consultar sobre una vista creada, ésta funcionalidad fue en realidad la parte más difícil de desarrollar, debido a que se consideró que la definición de la vista estuviera creada con XPath ó con XQuery. Ambos tipos son consultados con

XQuery, sin embargo, existen diferencias al consultarse, porque XQuery utiliza las clausulas FLOWR en sus sentencias, lo cuál incrementa su dificultad, por lo que se tenía que considerar estos puntos en la reestructuración. En cambio en XPath, solo se deben considerar consultas que inicien con "/" o con "//".

XMLView proporciona el poder de consultar sobre vistas toda aquella consulta que se desee en XPath, además da la oportunidad de agilizar el tiempo de ésta, ya que cuando se crea una vista se extrae parte de un documento XML que es de interés del usuario, utilizando una consulta XQuery o XPath, de tal forma que al consultar sobre ella se disminuye el tiempo de ejecución, porque ésta ya no es ejecutada contra todo un documento completo, el cual puede ser tan grande como se desee, sino sobre el fragmento que se obtuvo como resultado la vista; esto aunado a la eficiencia que se gana al no materializar ninguna de las partes de la misma.

Como se dijo anteriormente, es importante mencionar que una de las características con las que cuenta XMLView, es que al igual que en el modelo relacional, jamás materializa el documento–vista completo, ni partes de él; cuando se hace la consulta sobre la vista, se obtiene la definición de la misma y se reestructura usando XQuery, ya reestructurada eXist que es el manejador de bases de datos utilizado se encarga de obtener los resultados, los cuales serán mostrados al usuario, quién no se percatará de dicha reestructuración.

Uno de los aspectos importantes de XMLView es que al

hacer una consulta sobre vistas agrega a los resultados de la consulta un nodo raíz de tal forma que con esto se esta asegurando que el documento resultante que observa el usuario sea bien formado.

Resumiendo, la contribución de este trabajo es que proporciona una herramienta que permite la creación, consulta y borrado de vistas sobre documentos XML, almacenados en la base de datos nativa XML, eXist. El algoritmo presentado para la consulta sobre vistas (Sección 5.4), fue implementado en XMLView y está funcionando de forma correcta, lo cual puede verificarse tanto en el Capítulo 5 como en las ejecuciones mostradas en el Apéndice A.

6.2 Trabajo Futuro

Considerando que una consulta XQuery puede ser tan compleja como el usuario lo desee, se plantea como trabajo futuro la posibilidad de extender el algoritmo presentado en la sección 5.4, para que se considere la consulta sobre vistas que incluyan todos los tipos de consultas que XQuery permite, por ejemplo: aquellas en las cuáles se utiliza lo que en SQL sería un JOIN, porque hasta ahora XMLView se encuentra limitado al no considerar estos casos, así como también no considera consultas en las cuáles se utilizan algunas de las clausulas FLOWR en las que son usados diferentes nombres de variables a lo largo de to-

da la consulta. Así mismo sugiero que éste algoritmo podría ser optimizado, para reducir los tiempos de ejecución de las operaciones en cuanto a la administración de vistas. A continuación se describen algunos tipos de consultas que XMLView aún no soporta:

- No soporta la creación de vistas sobre definiciones donde se utilicen diferentes tipos de documentos. Porque cuando se consultan el usuario puede desear extraer información con un XPath en cualquiera de los documentos consultados, lo cual puede significar que en el return para la consulta reestructurada se deba obtener información de una variable diferente a la variable guardada en la definición en esta clausula, por lo que se tendría que buscar en cuál de los documentos se encuentra el XPath solicitado, para saber que variable devolverá el resultado en el return.

- Debido a que XQuery puede utilizar en sus consultas cualquiera o todas las clausulas FLOWR, en este trabajo no fueron consideradas las consultas que inician con una clausula let.

- Así mismo no existe aún en XMLView la reestructuración de vistas cuya definición inicia con un nodo raíz.

Otro trabajo futuro que se sugiere es hacer mejoras para que se permita la creación de una vista X basándose en una definición

que contenga un documento-vista Y. De tal forma que cuando se consulte sobre la vista X, XMLView sea capaz de reestructurar X e Y obteniendo los resultados esperados.

Uno de los aspectos con los que cuentan las bases de datos relacionales y sería necesario agregar a XMLView, es que al momento en que se borre un documento y éste forme parte de la definición de una vista creada, inmediatamente la vista sea borrada. Además de la modificación de vistas, sería un punto importante a agregarse.

A

Documentos y Pruebas de XMLView

A continuación se presentan los documentos XML usados en las pruebas de XMLView.

Documento Actores1.xml

```
<?xml version="1.0"?>
<!DOCTYPE Actores SYSTEM "Actores.xsd">
    <Actores>
        <ACTOR>
            <NOMBRE>JOHNNY DEPP</NOMBRE>
            <DIRECCION>
                <CALLE>500 South Sepulveda Blvd #500 </CALLE>
                <CIUDAD>Los Angeles, USA</CIUDAD>
```

```
</DIRECCION>
<DIRECCION>
    <CALLE>United Talent Agency 9560</CALLE>
    <CIUDAD>Beverly Hills, USA</CIUDAD>
</DIRECCION>
<PELICULAS>
    <PELICULA>
        <TITULO>Nick Of Time</TITULO>
        <ANIO>1995</ANIO>
    </PELICULA>
    <PELICULA>
        <TITULO>Dead Man</TITULO>
        <ANIO>1996</ANIO>
    </PELICULA>
    <PELICULA>
        <TITULO>Donnie Brasco</TITULO>
        <ANIO>1997</ANIO>
    </PELICULA>
    <PELICULA>
        <TITULO>Sleepy Hollow</TITULO>
        <ANIO>1998</ANIO>
    </PELICULA>
    <PELICULA>
        <TITULO>The Astronaut&amp;apos;s Wife-</TITULO>
        <ANIO>1999</ANIO>
    </PELICULA>
```

```
                <PELICULA>
                    <TITULO>Before night falls</TITULO>
                    <ANIO>2000</ANIO>
                </PELICULA>
                <PELICULA>
                    <TITULO>Blow</TITULO>
                    <ANIO>2001</ANIO>
                </PELICULA>
                <PELICULA>
                    <TITULO>Pirates of the Caribbean: The cur-
se of the Black Pearl</TITULO>
                    <ANIO>2003</ANIO>
                </PELICULA>
                <PELICULA>
                    <TITULO>Secret window</TITULO>
                    <ANIO>2004</ANIO>
                </PELICULA>
                <PELICULA>
                    <TITULO>The Libertine</TITULO>
                    <ANIO>2005</ANIO>
                </PELICULA>
                <PELICULA>
                    <TITULO>Charlie And The Chocolate Fac-
tory</TITULO>
                    <ANIO>2005</ANIO>
                </PELICULA>
```

```
                <PELICULA>
                    <TITULO>Pirates of the Caribbean: Dead
Man&amp;apos;s Chest</TITULO>
                    <ANIO>2006</ANIO>
                </PELICULA>
                <PELICULA>
                    <TITULO>Pirates of the Caribbean: At world-
&amp;apos;s end</TITULO>
                    <ANIO>2007</ANIO>
                </PELICULA>
                <PELICULA>
                    <TITULO>Gonzo: The Life and Work of
Dr. Hunter S. Thompson</TITULO>
                    <ANIO>2008</ANIO>
                </PELICULA>
                </PELICULA>
        </ACTOR>
        <ACTOR>
            <NOMBRE>Brad Pitt</NOMBRE>
            <DIRECCION>
                <CALLE>CAA9830 Wilshire Blvd </CALLE>
                <CIUDAD>Beverly Hills, USA</CIUDAD>
            </DIRECCION>
            <PELICULAS>
                <PELICULA>
                    <TITULO>Twelve Monkeys</TITULO>
```

```
    <ANIO>1995</ANIO>
</PELICULA>
<PELICULA>
    <TITULO>Sleepers</TITULO>
    <ANIO>1996</ANIO>
</PELICULA>
<PELICULA>
    <TITULO>Seven Years in Tibet </TITULO>
    <ANIO>1996</ANIO>
</PELICULA>
<PELICULA>
    <TITULO>Devil&amp;apos;s Own</TITULO>
    <ANIO>1997</ANIO>
</PELICULA>
<PELICULA>
    <TITULO>Meet Joe Black</TITULO>
    <ANIO>1997</ANIO>
</PELICULA>
<PELICULA>
    <TITULO>Fight Club</TITULO>
    <ANIO>1999</ANIO>
</PELICULA>
<PELICULA>
    <TITULO>The Mexican</TITULO>
    <ANIO>2001</ANIO>
</PELICULA>
```

```
                <PELICULA>
                    <TITULO>Ocean&amp;apos;s Eleven</TITULO>
                    <ANIO>2001</ANIO>
                </PELICULA>
                <PELICULA>
                    <TITULO>Confessions of a Dangerous Mi-
nd</TITULO>
                    <ANIO>2002</ANIO>
                </PELICULA>
                <PELICULA>
                    <TITULO>Troy</TITULO>
                    <ANIO>2004</ANIO>
                </PELICULA>
                <PELICULA>
                    <TITULO>The Curious Case of Benjamin
Button</TITULO>
                    <ANIO>2008</ANIO>
                </PELICULA>
                </PELICULA>
        </ACTOR>
        <ACTOR>
            <NOMBRE>Robert De Niro</NOMBRE>
            <DIRECCION>
                <CALLE>Tribeca Entertainment 375</CALLE>
                <CIUDAD>New York, USA</CIUDAD>
            </DIRECCION>
```

```
<PELICULAS>
    <PELICULA>
        <TITULO>Casino</TITULO>
        <ANIO>1995</ANIO>
    </PELICULA>
    <PELICULA>
        <TITULO>Sleepers</TITULO>
        <ANIO>1996</ANIO>
    </PELICULA>
    <PELICULA>
        <TITULO>Wag the Dog</TITULO>
        <ANIO>1997</ANIO>
    </PELICULA>
    <PELICULA>
        <TITULO>Great Expectations</TITULO>
        <ANIO>1998</ANIO>
    </PELICULA>
    <PELICULA>
        <TITULO>Flawless</TITULO>
        <ANIO>1999</ANIO>
    </PELICULA>
    <PELICULA>
        <TITULO>The Adventures of Rocky&amp;amp;-
Bullwinkle</TITULO>
        <ANIO>2000</ANIO>
    </PELICULA>
```

```
        <PELICULA>
            <TITULO>15 minutes</TITULO>
            <ANIO>2001</ANIO>
        </PELICULA>
        <PELICULA>
            <TITULO>City by the Sea</TITULO>
            <ANIO>2002</ANIO>
        </PELICULA>
        <PELICULA>
            <TITULO>Meet the fockers </TITULO>
            <ANIO>2004</ANIO>
        </PELICULA>
        <PELICULA>
            <TITULO>Hide and Seek</TITULO>
            <ANIO>2005</ANIO>
        </PELICULA>
        <PELICULA>
            <TITULO>Stardust</TITULO>
            <ANIO>2007</ANIO>
        </PELICULA>
        <PELICULA>
            <TITULO>What Just Happened?</TITULO>
            <ANIO>2008</ANIO>
        </PELICULA>
        </PELICULA>
</ACTOR>
```

```
        <ACTOR>
            <NOMBRE>Morgan Freeman</NOMBRE>
            <DIRECCION>
                <CALLE>Revelations Entertainment 1221 2nd
Street 4th Floor</CALLE>
                <CIUDAD>Santa Monica, USA</CIUDAD>
            </DIRECCION>
            <PELICULAS>
                <PELICULA>
                    <TITULO>Seven</TITULO>
                    <ANIO>1995</ANIO>
                </PELICULA>
                <PELICULA>
                    <TITULO>Moll Flanders</TITULO>
                    <ANIO>1996</ANIO>
                </PELICULA>
                <PELICULA>
                    <TITULO>Kiss the Girls</TITULO>
                    <ANIO>1997</ANIO>
                </PELICULA>
                <PELICULA>
                    <TITULO>Deep Impact</TITULO>
                    <ANIO>1998</ANIO>
                </PELICULA>
                <PELICULA>
                    <TITULO>Nurse Betty</TITULO>
```

```
    <ANIO>2000</ANIO>
</PELICULA>
<PELICULA>
    <TITULO>Along Came a Spider</TITULO>
    <ANIO>2001</ANIO>
</PELICULA>
<PELICULA>
    <TITULO>High Crimes</TITULO>
    <ANIO>2002</ANIO>
</PELICULA>
<PELICULA>
    <TITULO>Levity</TITULO>
    <ANIO>2003</ANIO>
</PELICULA>
<PELICULA>
    <TITULO>Million Dollar Baby</TITULO>
    <ANIO>2004</ANIO>
</PELICULA>
<PELICULA>
    <TITULO>March of the Penguins-</TITULO>
    <ANIO>2005</ANIO>
</PELICULA>
<PELICULA>
    <TITULO>Lucky Number Slevin</TITULO>
    <ANIO>2006</ANIO>
</PELICULA>
```

```
                <PELICULA>
                    <TITULO>Evan Almighty</TITULO>
                    <ANIO>2007</ANIO>
                </PELICULA>
                <PELICULA>
                    <TITULO>Wanted</TITULO>
                    <ANIO>2008</ANIO>
                </PELICULA>
                </PELICULA>
        </ACTOR>
        <ACTOR>
            <NOMBRE>Nicole Kidman</NOMBRE>
            <DIRECCION>
                <CALLE>Shanahan Management Pty Ltd. Lev-
el 3 Berman House 91 Campbell Street</CALLE>
                <CIUDAD>Surry Hills, Australia</CIUDAD>
            </DIRECCION>
            <DIRECCION>
                <CALLE>Blossom Films 10201 W. Pico Blvd.
45</CALLE>
                <CIUDAD>Los Angeles, USA</CIUDAD>
            </DIRECCION>
            <PELICULAS>
                <PELICULA>
                    <TITULO>Batman Forever</TITULO>
                    <ANIO>1995</ANIO>
```

```
</PELICULA>
<PELICULA>
    <TITULO>The Leading Man</TITULO>
    <ANIO>1996</ANIO>
</PELICULA>
<PELICULA>
    <TITULO>The Peacemaker</TITULO>
    <ANIO>1997</ANIO>
</PELICULA>
<PELICULA>
    <TITULO>Practical Magic</TITULO>
    <ANIO>1998</ANIO>
</PELICULA>
<PELICULA>
    <TITULO>Eyes Wide Shut</TITULO>
    <ANIO>1999</ANIO>
</PELICULA>
<PELICULA>
    <TITULO>The Others</TITULO>
    <ANIO>2001</ANIO>
</PELICULA>
<PELICULA>
    <TITULO>The Hours</TITULO>
    <ANIO>2002</ANIO>
</PELICULA>
<PELICULA>
```

```
            <TITULO>Cold Mountain</TITULO>
            <ANIO>2003</ANIO>
        </PELICULA>
        <PELICULA>
            <TITULO>Birth</TITULO>
            <ANIO>2004</ANIO>
        </PELICULA>
        <PELICULA>
            <TITULO>The interpreter</TITULO>
            <ANIO>2005</ANIO>
        </PELICULA>
        <PELICULA>
            <TITULO>Happy Feet</TITULO>
            <ANIO>2006</ANIO>
        </PELICULA>
        <PELICULA>
            <TITULO>The Golden Compass</TITULO>
            <ANIO>2007</ANIO>
        </PELICULA>
        <PELICULA>
            <TITULO>Australia</TITULO>
            <ANIO>2008</ANIO>
        </PELICULA>
        </PELICULA>
</ACTOR>
<ACTOR>
```

```
            <NOMBRE>Julia Roberts</NOMBRE>
            <DIRECCION>
                <CALLE>386 Park Avenue South 10</CALLE>
                <CIUDAD>New York, USA</CIUDAD>
            </DIRECCION>
            <DIRECCION>
                <CALLE>8942 Wilshire Boulevard</CALLE>
                <CIUDAD>Beverly Hills, USA</CIUDAD>
            </DIRECCION>
            <PELICULAS>
                <PELICULA>
                    <TITULO>Something to Talk About</TITULO>
                    <ANIO>1995</ANIO>
                </PELICULA>
                <PELICULA>
                    <TITULO>Michael Collins</TITULO>
                    <ANIO>1996</ANIO>
                </PELICULA>
                <PELICULA>
                    <TITULO>My Best Friend&amp;apos;s Wedding-
</TITULO>
                    <ANIO>1997</ANIO>
                </PELICULA>
                <PELICULA>
                    <TITULO>Stepmom</TITULO>
                    <ANIO>1998</ANIO>
```

```
        </PELICULA>
        <PELICULA>
            <TITULO>Notting Hill</TITULO>
            <ANIO>1999</ANIO>
        </PELICULA>
        <PELICULA>
            <TITULO>Erin Brockovich</TITULO>
            <ANIO>2000</ANIO>
        </PELICULA>
        <PELICULA>
            <TITULO>Ocean's Eleven de Steven Soder-
bergh</TITULO>
            <ANIO>2001</ANIO>
        </PELICULA>
        <PELICULA>
            <TITULO>Confessions of a dangerous mi-
nd</TITULO>
            <ANIO>2002</ANIO>
        </PELICULA>
        <PELICULA>
            <TITULO>Mona Lisa Smile</TITULO>
            <ANIO>2003</ANIO>
        </PELICULA>
        <PELICULA>
            <TITULO>Closer</TITULO>
            <ANIO>2004</ANIO>
```

```
            </PELICULA>
            <PELICULA>
                <TITULO>Charlotte&amp;apos;s Web</TITULO>
                <ANIO>2006</ANIO>
            </PELICULA>
            <PELICULA>
                <TITULO>Charlie Wilson&amp;apos;s war-</TITULO>
                <ANIO>2007</ANIO>
            </PELICULA>
            <PELICULA>
                <TITULO>Fireflies in the Garden</TITULO>
                <ANIO>2008</ANIO>
            </PELICULA>
            </PELICULA>
</ACTOR>
<ACTOR>
    <NOMBRE>Meg Ryan</NOMBRE>
<DIRECCION>
            <CALLE>Management 360 Wilshire</CALLE>
            <CIUDAD>Beverly Hills, USA</CIUDAD>
    </DIRECCION>
    <PELICULAS>
            <PELICULA>
                <TITULO>French Kiss</TITULO>
                <ANIO>1995</ANIO>
            </PELICULA>
```

```
<PELICULA>
    <TITULO>Courage Under Fire</TITULO>
    <ANIO>1996</ANIO>
</PELICULA>
<PELICULA>
    <TITULO>Addicted to Love</TITULO>
    <ANIO>1997</ANIO>
</PELICULA>
<PELICULA>
    <TITULO>City of Angels</TITULO>
    <ANIO>1998</ANIO>
</PELICULA>
<PELICULA>
    <TITULO>You’ve Got Mail</TITULO>
    <ANIO>1998</ANIO>
</PELICULA>
<PELICULA>
    <TITULO>Hanging Up</TITULO>
    <ANIO>2000</ANIO>
</PELICULA>
<PELICULA>
    <TITULO>Kate&amp;amp;Leopold</TITULO>
    <ANIO>2001</ANIO>
</PELICULA>
<PELICULA>
    <TITULO>In the Cut</TITULO>
```

```
                <ANIO>2003</ANIO>
            </PELICULA>
            <PELICULA>
                <TITULO>Against the Ropes</TITULO>
                <ANIO>2004</ANIO>
            </PELICULA>
            <PELICULA>
                <TITULO>In the land of women</TITULO>
                <ANIO>2007</ANIO>
            </PELICULA>
            <PELICULA>
                <TITULO>My mom's new boyfriend</TITULO>
                <ANIO>2008</ANIO>
            </PELICULA>
            <PELICULA>
                <TITULO>The Women</TITULO>
                <ANIO>2008</ANIO>
            </PELICULA>
            </PELICULA>
        </ACTOR>
        <ACTOR>
            <NOMBRE>Charlize Theron</NOMBRE>
            <DIRECCION>
                <CALLE>Spanky Taylor 3727 W. Magnolia Suite
300</CALLE>
                <CIUDAD>Burbank, USA</CIUDAD>
```

```
            </DIRECCION>
            <DIRECCION>
                <CALLE>9220 Sunset Blvd. Suite 306</CALLE>
                <CIUDAD>Los Angeles, USA</CIUDAD>
            </DIRECCION>
            <PELICULAS>
                <PELICULA>
                    <TITULO>Children of the Corn III: Urban
Harvest</TITULO>
                    <ANIO>1995</ANIO>
                </PELICULA>
                <PELICULA>
                    <TITULO>That Thing You Do!</TITULO>
                    <ANIO>1996</ANIO>
                </PELICULA>
                <PELICULA>
                    <TITULO>The Devil's Advocate</TITULO>
                    <ANIO>1997</ANIO>
                </PELICULA>
                <PELICULA>
                    <TITULO>Mighty Joe Young</TITULO>
                    <ANIO>1998</ANIO>
                </PELICULA>
                <PELICULA>
                    <TITULO>The Astronaut's Wife</TITULO>
                    <ANIO>1999</ANIO>
```

```
</PELICULA>
<PELICULA>
    <TITULO>Men of Honor</TITULO>
    <ANIO>2000</ANIO>
</PELICULA>
<PELICULA>
    <TITULO>Sweet November</TITULO>
    <ANIO>2001</ANIO>
</PELICULA>
<PELICULA>
    <TITULO>Trapped</TITULO>
    <ANIO>2002</ANIO>
</PELICULA>
<PELICULA>
    <TITULO>Monster</TITULO>
    <ANIO>2003</ANIO>
</PELICULA>
<PELICULA>
    <TITULO>Head in the Clouds</TITULO>
    <ANIO>2004</ANIO>
</PELICULA>
<PELICULA>
    <TITULO>North Country</TITULO>
    <ANIO>2005</ANIO>
</PELICULA>
<PELICULA>
```

```
                <TITULO>In the Valley of Elah</TITULO>
                <ANIO>2007</ANIO>
            </PELICULA>
            <PELICULA>
                <TITULO>Hancock</TITULO>
                <ANIO>2008</ANIO>
            </PELICULA>
            <PELICULA>
                <TITULO>Battle in Seattle</TITULO>
                <ANIO>2008</ANIO>
            </PELICULA>
            </PELICULA>
        </ACTOR>
    </Actores>
```

Documento Comentarios1.xml

```
<?xml version="1.0"?>
    <COMENTARIOS>
        <COMENTARIOS_PELICULA>
            <TITULO_P>Closer</TITULO_P>
            <ANIO_P>2004</ANIO_P>
            <COMENTARIO> &quot;Closer&quot;, título muy
apropiado, es, en efecto, una aproximación sin excesivas con-
templaciones a las dinámicas que unen y separan a un grupo de
perdedores sentimentales, que al final terminan tomándose este
```

asunto del amor como un juego de competición en donde la infidelidad opera como triunfo o fracaso, por más que todos se metan goles en su propia portería. Y como todo cine adulto que se precie, su visionado es de los que genera tertulias posteriores. Estupendo regreso al cine para Nichols.</COMENTARIO>
</COMENTARIOS_PELICULA>
<COMENTARIOS_PELICULA>
<TITULO_P>Charlie And The Chocolate Factory-</TITULO_P>
<ANIO_P>2005</ANIO_P>
<COMENTARIO>Orgullo, gula, pereza, avaricia y bondad, una moraleja escrita por el talentoso autor Roald Dahl, de cuyas obras adaptadas al séptimo artes son "Las brujas", "Matilda", "James y el melocotón gigante" y "Charlie y la fábrica de chocolate", siendo esta última la más conocida en el medio literario pero no precisamente en le medio cinematográfico.</COMENTARIO>
<PREMIOS>
<PREMIO CATEGORIA=“PREMIOS OSCAR” ANIO_PREMIO=“2005”>
<NOMINACIONES>
<NOMINACION>Mejor diseño de vestuario</NOMINACION>
<RESULTADO>Nominada</RESULTADO>
</NOMINACIONES>

```
                </PREMIO>
                <PREMIO CATEGORIA=“GLOBOS DE ORO”
AN-IO_PREMIO="2005">
                    <NOMINACIONES>
                        <NOMINACION>Mejor actor - Come-
dia o Musical</NOMINACION>
                        <RESULTADO>Nominada</RESULT-
ADO>
                    </NOMINACIONES>
                </PREMIO>
                <PREMIO CATEGORIA=“PREMIOS BAFTA”
AN-IO_PREMIO=“2005”>
                    <NOMINACIONES>
                        <NOMINACION>Mejor diseño de produ-
cción</NOMINACION>
                        <RESULTADO>Nominada</RESULT-
ADO>
                    </NOMINACIONES>
                    <NOMINACIONES>
                        <NOMINACION>Mejor diseño de ves-
tuario</NOMINACION>
                        <RESULTADO>Nominada</RESULT-
ADO>
                    </NOMINACIONES>
                </PREMIO>
            </PREMIOS>
```

```
        </COMENTARIOS_PELICULA>
        <COMENTARIOS_PELICULA>
            <TITULO_P>The Interpreter</TITULO_P>
            <ANIO_P>2005</ANIO_P>
            <COMENTARIO>&amp;quot;La intérprete&amp;quot; no o-
frece más que un conato de thriller escondido en un drama
melancólico, elaborado con certera seriedad y corrección for-
mal, pero que tiene como mayor enemigo un guión que no
consigue insuflar ningún incentivo, ya sea algo de desasosiego,
hondura o emoción a una historia previsible y artificiosa con
unos personajes poco creíbles incluidos en un entorno real que
es desaprovechado en lo que podría haber sido una oportuni-
dad inmejorable por recuperar la zona oscura y clásica de un
géne-ro bastante quebrantado.</COMENTARIO>
        </COMENTARIOS_PELICULA>
        <COMENTARIOS_PELICULA>
            <TITULO_P>Confessions of a Dangerous Mind</-
TITULO_P>
            <ANIO_P>2002</ANIO_P>
            <COMENTARIO>Ubicada dentro de un cine más
independiente que comercial, con ambiciones estéticas y temáti-
ca poco complaciente, &amp;quot;Confessions of a dangerous mind&-
amp;quot; es una obra irregular pero digna de mención, que des-
cubre entre grandes fallos narrativos secuencias de sumo interés.-
</CO-MENTARIO>
            <PREMIOS>
```

```
                <PREMIO CATEGORIA=“Berlinale” ANIO_-
PREMIO="2002">
                    <NOMINACIONES>
                        <NOMINACION>Mejor pelicula</NO-
MINACION>
                        <RESULTADO>Nominada</RESULT-
ADO>
                    </NOMINACIONES>
                <NOMINACIONES>
                        <NOMINACION>Mejor actor</NOMI-
NACION>
                        <RESULTADO>Nominada</RESULT-
ADO>
                    </NOMINACIONES>
                </PREMIO>
                <PREMIO CATEGORIA=“Chicago Film Crit-
ics Association Awards” ANIO_PREMIO="2002">
                    <NOMINACIONES>Mejor guión</NOMI-
NACIONES>
                        <RESULTADO>Nominada</RESULT-
ADO>
                </PREMIO>
            </PREMIOS>
        </COMENTARIOS_PELICULA>
        <COMENTARIOS_PELICULA>
            <TITULO_P>Stardust</TITULO_P>
```

<ANIO_P>2007</ANIO_P>
<COMENTARIO>El joven director inglés Matthew Vaughn ya se había aventurado en la dirección de un largometraje cuando en el 2004 dirigió "Layer Cake", película protagonizada por Daniel Craig antes de que el actual James Bond fuera un personaje reconocido hasta para el más olvidadizo de los espectadores. Con esa cinta no llamó mucho la atención, recurriendo en su segundo intento a actores de renombre que le ejecutaran una campaña de marketing sin un esfuerzo significativo, de manera de situar su obra en la expectación del espectador. Es así como "Stardust", película de fantasía que sigue la línea de muchas otras, llama la atención desde un comienzo integrando en su reparto a actores de la categoría de Robert De Niro, Peter O'Toole e Ian McKellen.-
</COMENTARIO>
</COMENTARIOS_PELICULA>
<COMENTARIOS_PELICULA>
<TITULO_P>Wanted</TITULO_P>
<ANIO_P>2008</ANIO_P>
<COMENTARIO>Con Wanted, tenemos el parámetro o gen de acción, pero la historia queda relegada a un segundo plano sin segundos con los que contarla incluso con la ternura de su dueño y mentor, el escritor del guión, no interesa, la gran industria quiere destrozar y masacrar a base de acción a raudales, cuanto más elegante, directa y espectacular mejor, y ha resultado que con la diosa del momento y un jovencito

que tiene buena prensa va a transportarnos a un mundo que viajará entre Matrix y Misión imposible, una forma de aglomerar dejando a un lado la lógica y la consistente calidad.</COMENTARIO>

</COMENTARIOS_PELICULA>

<COMENTARIOS_PELICULA>

<TITULO_P>Australia</TITULO_P>

<ANIO_P>2008</ANIO_P>

<COMENTARIO>En Australia se puede observar que pese a la enorme simpatía que puedan ejercer sus escenas a lo largo de la primera parte de su metraje, con acertadas caricaturas de personajes, humor y drama rebosando por sus encuadres, con magníficas secuencias como la de la estampida (digital) o una llamativa fascinación hacia la iluminación artificial, contraluces de fantasía, ese aire de fotonovela más kitsch y toda la pomposidad que desea desprender, termina revelando que, casi 70 años después, la famosa e inmortal plantación "Tara" sigue ganando la partida a la de Faraway Downs.</COMENTARIO>

</COMENTARIOS_PELICULA>

<COMENTARIOS_PELICULA>

<TITULO_P>Handcock</TITULO_P>

<ANIO_P>2008</ANIO_P>

<COMENTARIO>En consecuencia, el guión se - vende a sí mismo rendido a la tentación de rascar el bolsillo de una audiencia más universal a cada minuto que pasa, olvi-

dando cual-quier atención y cuidado en la puesta en escena y dejándose llevar por un bombardeo estridente de efectos especiales, tan efectivos y espectaculares como inocuos ante la falta de una Némesis que realmente aporte emoción a una intriga aguada, sombra de sus propias presunciones. Por su incapacidad para alcanzar sus metas, puede resultar idónea para una sesión doble con "Mistery Men", que también buscó en su día auparse a los altares de los títulos de culto y se quedó a medio camino; con todo, recomendable como fugaz entretenimiento rápidamente olvidable, con momentos indudablemente acertados que, si bien sucumben y quedan sepultados una vez se imponen las concesiones a su verdadera naturaleza mercantilista generadora de pingües beneficios económicos, convierten a &-quot;Hancock" en un divertimento atípico, cáustico en ocasiones y saludable por lo que pretende significar.</COMEN-TARIO>

</COMENTARIOS_PELICULA>

<COMENTARIOS_PELICULA>

<TITULO_P>The Curious Case of Benjamin Button</TITULO_P>

<ANIO_P>2008</ANIO_P>

<COMENTARIO>Después de la incursión en buenas películas de acción y de intrigas mas emocionantes, David Fincher nos regala una superproducción donde logra crear una historia romántica poco común y donde la verdad los aspectos técnicos son el fuerte mas atrayente de la película. Cabe

mencionar que este film te logra cautivar y atraer pese a los aburridos largos lapsos donde se veía muchas veces la misma escena.</COMENTARIO>

<PREMIOS>

<PREMIO CATEGORIA="PREMIOS OSCAR" ANIO_PREMIO="2008">

<NOMINACIONES>

<NOMINACION>Mejor película</NOMINACION>

<RESULTADO>Nominada</RESULTADO>

</NOMINACIONES>

<NOMINACIONES>

<NOMINACION>Mejor director</NOMINACION>

<RESULTADO>Nominada</RESULTADO>

</NOMINACIONES>

<NOMINACIONES>

<NOMINACION>Mejor actor</NOMINACION>

<RESULTADO>Nominada</RESULTADO>

</NOMINACIONES>

<NOMINACIONES>

<NOMINACION>Mejor actriz de reparto-

```
</NOMINACION>
                    <RESULTADO>Nominada</RESULT-
ADO>
                </NOMINACIONES>
                <NOMINACIONES>
                    <NOMINACION>Mejor guión adaptado-
</NOMINACION>
                    <RESULTADO>Nominada</RESULT-
ADO>
                </NOMINACIONES>
                <NOMINACIONES>
                    <NOMINACION>Mejor banda sonora-
</NOMINACION>
                    <RESULTADO>Nominada</RESULT-
ADO>
                </NOMINACIONES>
                <NOMINACIONES>
                    <NOMINACION>Mejor fotografía</NO-
MINACION>
                    <RESULTADO>Nominada</RESULT-
ADO>
                </NOMINACIONES>
                <NOMINACIONES>
                    <NOMINACION>Mejor montaje</NO-
MINACION>
                    <RESULTADO>Nominada</RESULT-
```

```
ADO>
                    </NOMINACIONES>
                    <NOMINACIONES>
                        <NOMINACION>Mejor dirección artísti-
ca</NOMINACION>
                        <RESULTADO>Ganadora</RESULTA-
DO>
                    </NOMINACIONES>
                    <NOMINACIONES>
                        <NOMINACION>Mejor diseno de ves-
tuario</NOMINACION>
                        <RESULTADO>Nominada</RESULT-
ADO>
                    </NOMINACIONES>
                    <NOMINACIONES>
                        <NOMINACION>Mejor maquillaje</NO-
MINACION>
                        <RESULTADO>Ganadora</RESULTA-
DO>
                    </NOMINACIONES>
                    <NOMINACIONES>
                        <NOMINACION>Mejor efectos visuales-
</NOMINACION>
                        <RESULTADO>Ganadora</RESULTA-
DO>
                    </NOMINACIONES>
```

```
            </PREMIO>
            <PREMIO CATEGORIA="GLOBOS DE ORO"
ANIO_PREMIO="2008">
                <NOMINACIONES>
                    <NOMINACION>Mejor película - Drama-
</NOMINACION>
                    <RESULTADO>Candidata</RESULTA-
DO>
                </NOMINACIONES>
                <NOMINACIONES>
                    <NOMINACION>Mejor director</NO-
MINACION>
                    <RESULTADO>Candidata</RESULTA-
DO>
                </NOMINACIONES>
                <NOMINACIONES>
                    <NOMINACION>Mejor actor - Drama-
</NOMINACION>
                    <RESULTADO>Candidata</RESULTA-
DO>
                </NOMINACIONES>
            </PREMIO>
            <PREMIO CATEGORIA="PREMIOS BAFTA"
ANIO_PREMIO="2005">
                <NOMINACIONES>
                    <NOMINACION>Mejor película</NO-
```

```
MINACION>
                    <RESULTADO>Candidata</RESULTA-
DO>
                </NOMINACIONES>
                <NOMINACIONES>
                    <NOMINACION>Mejor director</NO-
MINACION>
                    <RESULTADO>Candidata</RESULTA-
DO>
                </NOMINACIONES>
                <NOMINACIONES>
                    <NOMINACION>Mejor diseño de produ-
cción</NOMINACION>
                    <RESULTADO>Ganadora</RESULTA-
DO>
                </NOMINACIONES>
                <NOMINACIONES>
                    <NOMINACION>Mejor maquillaje</NO-
MINACION>
                    <RESULTADO>Ganadora</RESULTA-
DO>
                </NOMINACIONES>
                <NOMINACIONES>
                    <NOMINACION>Mejor efectos visuales-
</NOMINACION>
                    <RESULTADO>Ganadora</RESULTA-
```

```
DO>
                    </NOMINACIONES>
                </PREMIO>
            </PREMIOS>
        </COMENTARIOS_PELICULA>
    </COMENTARIOS>
```

Documento Productos1.xml

```
<?xml version="1.0"?>
<Ordenes>
    <orden id="r1">
        <producto>
            <descrip>Shampoo Pantene, 2 en 1 de 790ml</des-
crip>
            <precio>76.50</precio>
        </producto>
        <producto>
            <descrip>Cajeta envinada, 350 ml</descrip>
            <precio>26.60</precio>
        </producto>
        <producto>
            <descrip>Korn Flakes</descrip>
            <precio>28.00</precio>
        </producto>
    </orden>
```

```
    <orden id=“r2">
        <producto>
            <descrip>Korn Flakes</descrip>
            <precio>28.00</precio>
        </producto>
    </orden>
    <orden id=“r3">
        <producto>
            <descrip>Cuaderno Scribe, 100 hojas</descrip>
            <precio>44.80</precio>
        </producto>
        <producto>
            <descrip>Caja de Lápices, Mirado</descrip>
            <precio>25.00</precio>
        </producto>
    </orden>
</Ordenes>
```

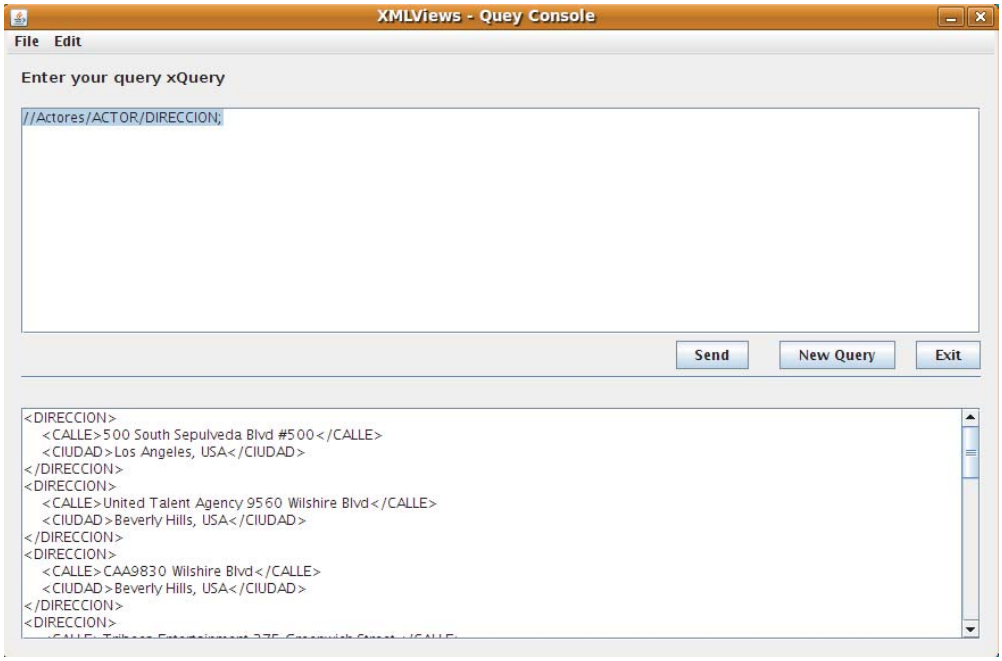

Figura A.1: Consulta XPath que obtiene las direcciones de los actores encontrados en el documento Actores1.xml.

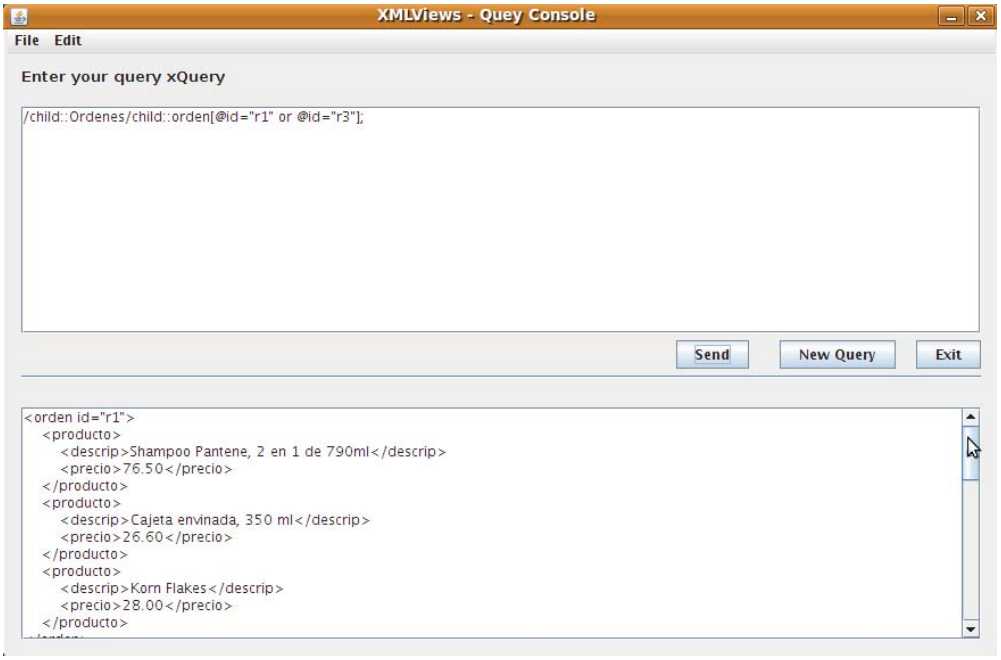

Figura A.2: Consulta XPath que obtiene las ordenes con id = “r1” ó id = “r3” del documento XML Productos1.xml.

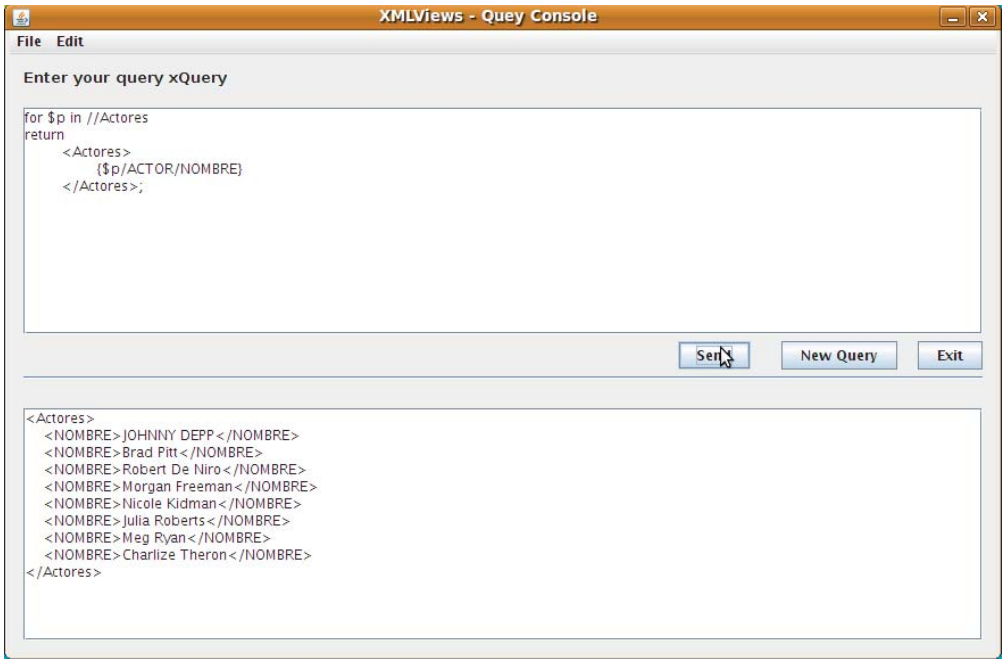

Figura A.3: Consulta XQuery que obtiene los nombres de los actores almacenados en el documento XML Actores1.xml.

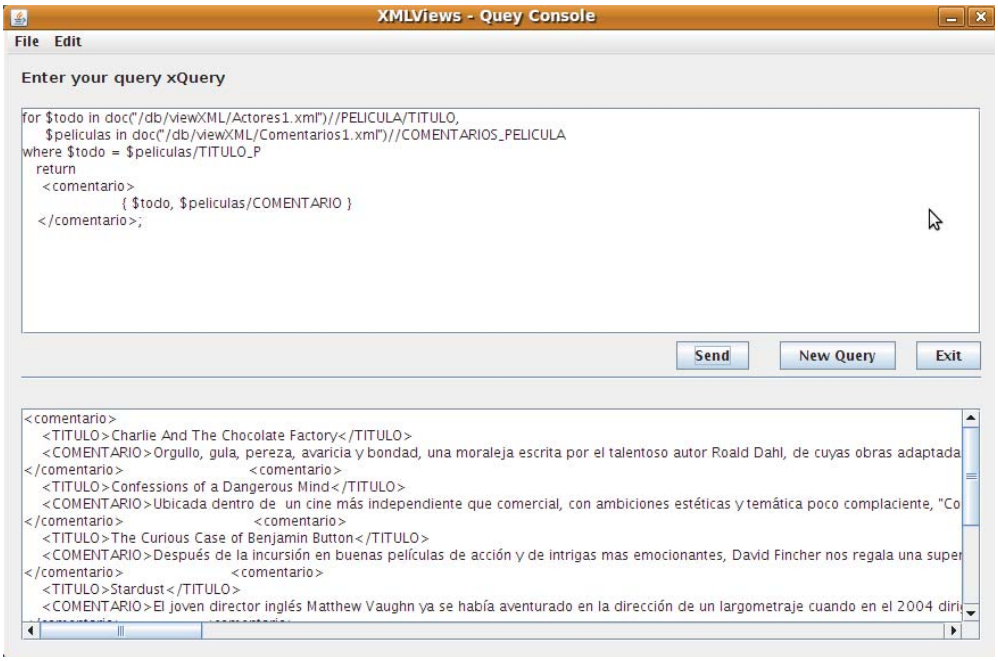

Figura A.4: Consulta XQuery que obtiene los comentarios en Comentarios1.xml para aquellos títulos de películas almacenados en el documento Actores1.xml.

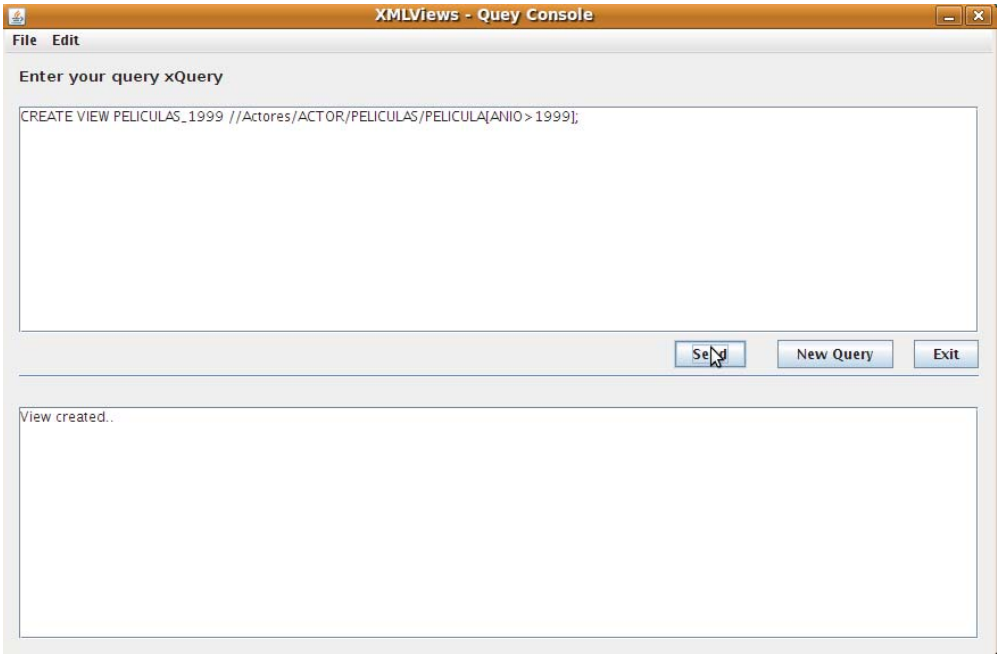

Figura A.5: Creación de una vista con nombre PELICULAS_1999, la cuál consta de una definición que obtiene como resultados todas las películas que fueron creadas después del año 1999.

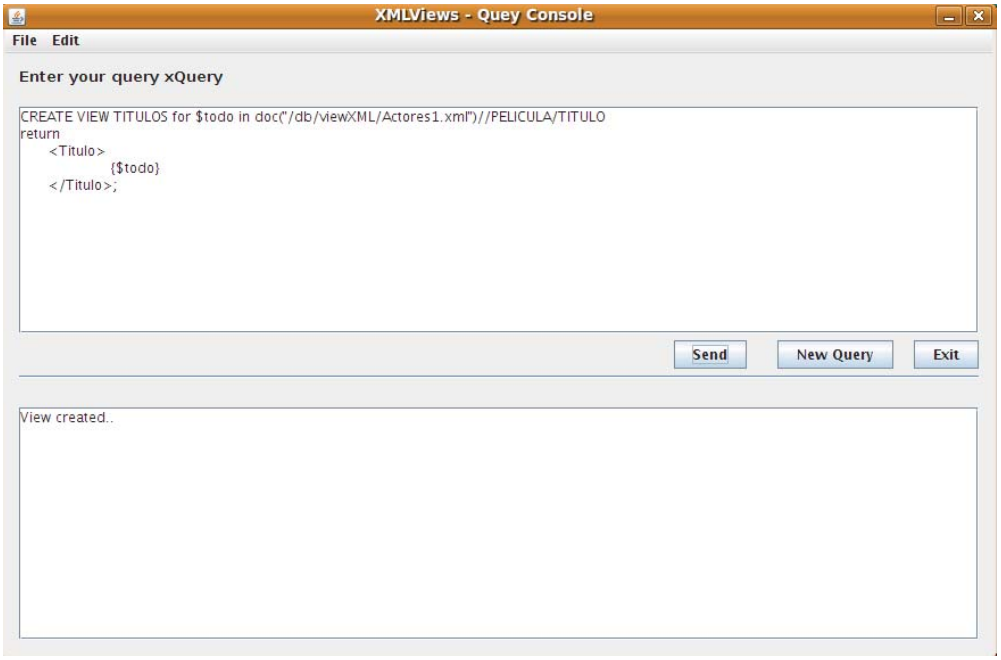

Figura A.6: Creación de una vista con nombre TITULOS, la cuál consta de una definición que obtiene como resultados los títulos de todas las películas almacenadas en el documento Actores1.xml.

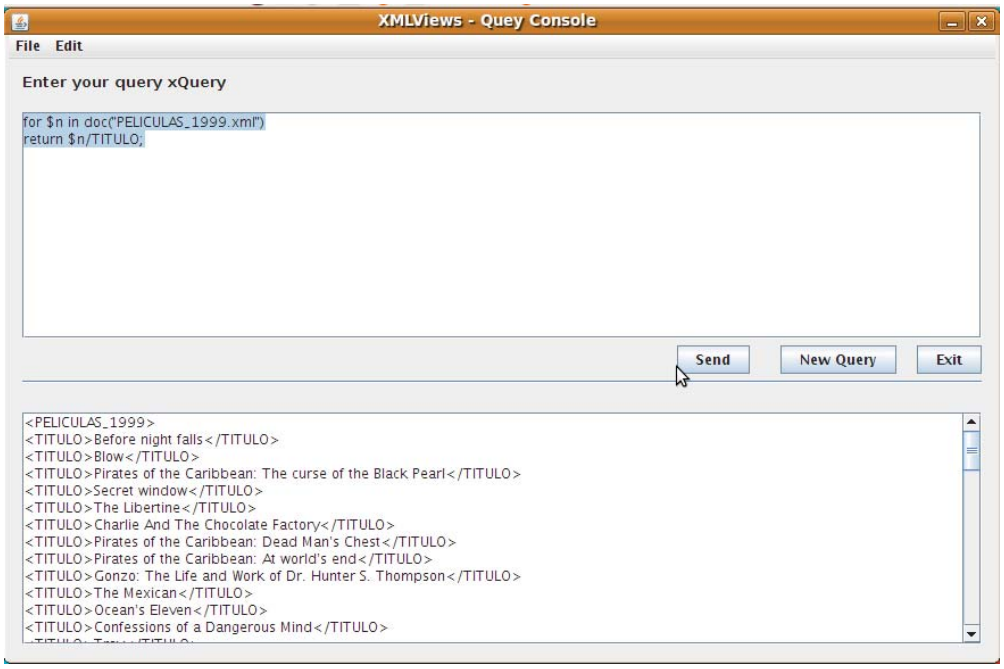

Figura A.7: Consulta sobre la vista PELICULAS_1999.xml, la cual obtiene los títulos de las películas creadas después del año 1999.

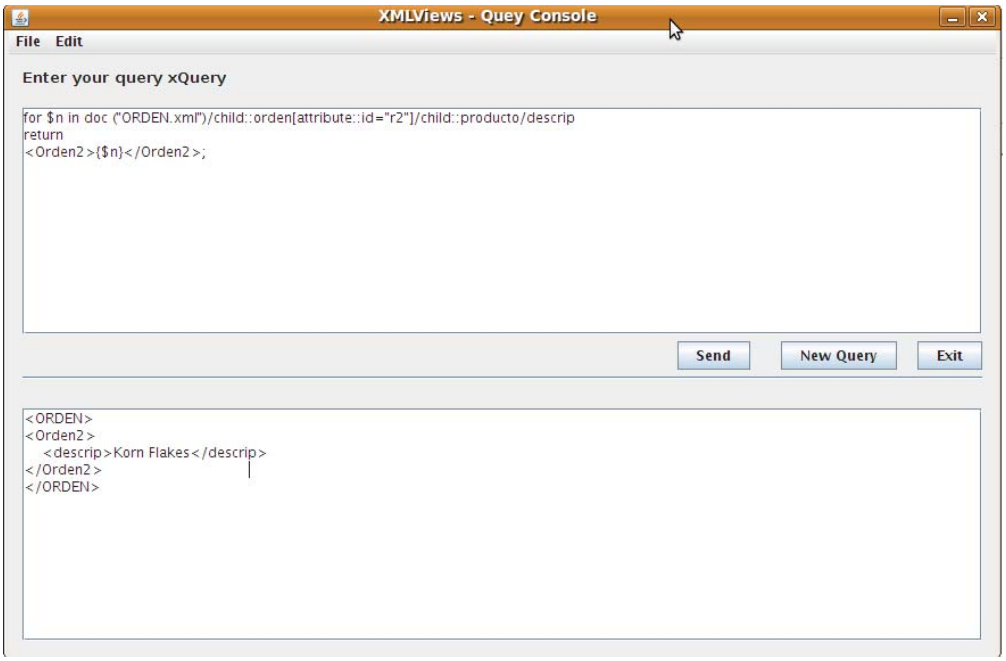

Figura A.8: Consulta sobre la vista ORDEN.xml, la cual obtiene la descripción de los productos de la orden con id = “r2”, almacenada en dicha vista.

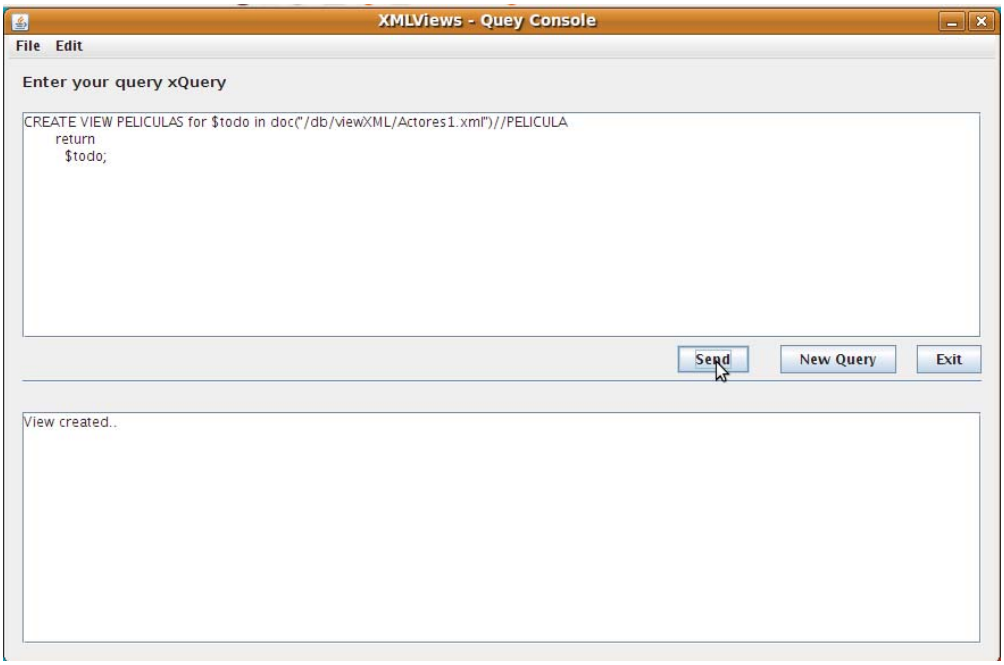

Figura A.9: Consulta para la creación de una vista llamada PELICULAS, cuya definición es una sentencia XQuery que obtiene el contenido de todos los elementos “PELICULA”, existentes en el documento Actores1.xml.

Resultados de la consulta sobre la vista PELICULAS. Esta consulta obtiene todos los títulos de películas almacenadas en el documento Actores1.xml.

<PELICULAS>
<TITULO>Nick Of Time</TITULO>

<TITULO>Dead Man</TITULO>
<TITULO>Donnie Brasco</TITULO>
<TITULO>Sleepy Hollow</TITULO>
<TITULO>The Astronaut's Wife</TITULO>
<TITULO>Before night falls</TITULO>
<TITULO>Blow</TITULO>
<TITULO>Pirates of the Caribbean: The curse of the Black Pearl</TITULO>
<TITULO>Secret window</TITULO>
<TITULO>The Libertine</TITULO>
<TITULO>Charlie And The Chocolate Factory</TITULO>
<TITULO>Pirates of the Caribbean: Dead Man's Chest</TITULO>
<TITULO>Pirates of the Caribbean: At world's end</TITULO>
<TITULO>Gonzo: The Life and Work of Dr. Hunter S. Thompson</TITULO>
<TITULO>Twelve Monkeys</TITULO>
<TITULO>Sleepers</TITULO>
<TITULO>Seven Years in Tibet</TITULO>
<TITULO>Devil's Own</TITULO>
<TITULO>Meet Joe Black</TITULO>
<TITULO>Fight Club</TITULO>
<TITULO>The Mexican</TITULO>
<TITULO>Ocean's Eleven</TITULO>
<TITULO>Confessions of a Dangerous Mind</TITULO>

<TITULO>Troy</TITULO>
<TITULO>The Curious Case of Benjamin Button</TITULO>
<TITULO>Casino</TITULO>
<TITULO>Sleepers</TITULO>
<TITULO>Wag the Dog</TITULO>
<TITULO>Great Expectations</TITULO>
<TITULO>Flawless</TITULO>
<TITULO>The Adventures of Rocky&Bullwinkle</TITULO>
<TITULO>15 minutes</TITULO>
<TITULO>City by the Sea</TITULO>
<TITULO>Meet the fockers</TITULO>
<TITULO>Hide and Seek</TITULO>
<TITULO>Stardust</TITULO>
<TITULO>What Just Happened?</TITULO>
<TITULO>Seven</TITULO>
<TITULO>Moll Flanders</TITULO>
<TITULO>Kiss the Girls</TITULO>
<TITULO>Deep Impact</TITULO>
<TITULO>Nurse Betty</TITULO>
<TITULO>Along Came a Spider</TITULO>
<TITULO>High Crimes</TITULO>
<TITULO>Levity</TITULO>
<TITULO>Million Dollar Baby</TITULO>
<TITULO>March of the Penguins</TITULO>
<TITULO>Lucky Number Slevin</TITULO>

<TITULO>Evan Almighty</TITULO>
<TITULO>Wanted</TITULO>
<TITULO>Batman Forever</TITULO>
<TITULO>The Leading Man</TITULO>
<TITULO>The Peacemaker</TITULO>
<TITULO>Practical Magic</TITULO>
<TITULO>Eyes Wide Shut</TITULO>
<TITULO>The Others</TITULO>
<TITULO>The Hours</TITULO>
<TITULO>Cold Mountain</TITULO>
<TITULO>Birth</TITULO>
<TITULO>The interpreter</TITULO>
<TITULO>Happy Feet</TITULO>
<TITULO>The Golden Compass</TITULO>
<TITULO>Australia</TITULO>
<TITULO>Something to Talk About</TITULO>
<TITULO>Michael Collins</TITULO>
<TITULO>My Best Friend's Wedding</TITULO>
<TITULO>Stepmom</TITULO>
<TITULO>Notting Hill</TITULO>
<TITULO>Erin Brockovich</TITULO>
<TITULO>Ocean's Eleven de Steven Soderbergh</TITULO>
<TITULO>Confessions of a dangerous mind</TITULO>
<TITULO>Mona Lisa Smile</TITULO>
<TITULO>Closer</TITULO>

<TITULO>Charlotte’s Web</TITULO>
<TITULO>Charlie Wilson’s war</TITULO>
<TITULO>Fireflies in the Garden</TITULO>
<TITULO>French Kiss</TITULO>
<TITULO>Courage Under Fire</TITULO>
<TITULO>Addicted to Love</TITULO>
<TITULO>City of Angels</TITULO>
<TITULO>You’ve Got Mail</TITULO>
<TITULO>Hanging Up</TITULO>
<TITULO>Kate&Leopold</TITULO>
<TITULO>In the Cut</TITULO>
<TITULO>Against the Ropes</TITULO>
<TITULO>In the land of women</TITULO>
<TITULO>My mom’s new boyfriend</TITULO>
<TITULO>The Women</TITULO>
<TITULO>Children of the Corn III: Urban Harvest</TITULO>
<TITULO>That Thing You Do!</TITULO>
<TITULO>The Devil’s Advocate</TITULO>
<TITULO>Mighty Joe Young</TITULO>
<TITULO>The Astronaut’s Wife</TITULO>
<TITULO>Men of Honor</TITULO>
<TITULO>Sweet November</TITULO>
<TITULO>Trapped</TITULO>
<TITULO>Monster</TITULO>
<TITULO>Head in the Clouds</TITULO>

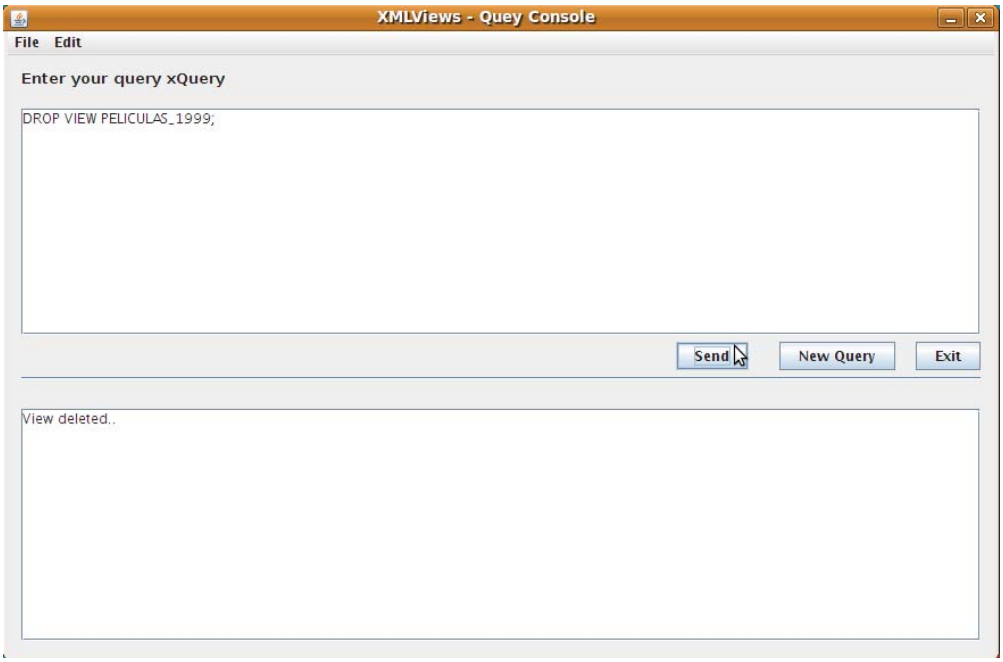

Figura A.10: Borrado de la vista PELICULAS_1999.

<TITULO>North Country</TITULO>
<TITULO>In the Valley of Elah</TITULO>
<TITULO>Hancock</TITULO>
<TITULO>Battle in Seattle</TITULO>
</PELICULAS>

Bibliografía

[1] Gummadi Abhilash, Yoon Jong P., Shah Biren, and et al. A bitmap-based access control for restricted views of xml documents. *ACM Workshop on XML Security*, pages 60–68, 2003.

[2] Fan Catalina, Funderburk John, Lam Hou-in, and et al. Xperanto: Bridging relational technology and xml. *Technology Incubation Center. IBM Software Labs, India*, pages 1–9, 2002.

[3] World Wide Web Consortium. Xhtml 1.0 the extensible hypertext markup language (second edition) [on line]. Available at `http://www.w3.org/TR/xhtml1/`. W3C Recommendation 26 January 2000, revised 1 August 2002. [Last viewed: June 10, 2009].

[4] World Wide Web Consortium. Xml path language (xpath) version 1.0 [on line]. Available at `http://www.w3.org/TR/xpath`. W3C Recommendation 16 November 1999. [Last viewed: June 12, 2009].

[5] World Wide Web Consortium. Xquery 1.0: An xml query language [on line]. Available at `http://www.w3.org/TR/xquery/`. W3C Recommendation 23 January 2007. [Last viewed: June 13, 2009].

[6] Srinivasa Deepak and Rajendra Rajeshwari. Xviews – view mechanism for native xml data. *Technology Incubation Center. IBM Software Labs, India*, pages 1–11, 2002.

[7] Gamma Erich, Helm Richard, Johnson Ralph, and et al. *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison-Wesley Pub Co, first edition, 1995. ISBN: 0201633612.

[8] Powell Gavin. *Beginning XML Databases*. Wiley Publishing, Inc, 2007. ISBN: 0-471-79120-2.

[9] García-Molina Hector, Ullman Jeffrey D., and Widom Jennifer. *Database Systems: The Complete Book.* Prentice-Hall, Inc, first edition, 2002. ISBN: 0-13-031995-3.

[10] Gutiérrez J. J., Escalona M. J., Mejías M., and et al. Xquery. 2005.

[11] Williams Kevin, Brundage Michael, Dengler Patrick, and et al. *Professional XML Databases*. Wrox Press Ltd, 2001. ISBN: 1861005059.

[12] Rajugan Rajagopal, Chang Elizabeth, Dillon Tharam S., and et al. Xml views: Part 1. *Springer-Verlag*, pages 148–159, 2003.

[13] Rajugan Rajagopal, Chang Elizabeth, Dillon Tharam S., and et al. Xml views, part ii: Modeling conceptual views using xsemantic nets. *The 30th Annual Conference of the IEEE Industrial Electronics Soclety, Busan, Korea*, pages 454–462, 2004.

[14] Bourret Ronald. Xml and databases [on line]. Available at `http://www.rpbourret.com/xml/XMLAndDatabases.htm`. W3C Recommendation 23 January 2007. [Last viewed: September 19, 2009].

[15] Natu Shalaka and Mendonca John. Digital asset management using a native xml database implementation. *CITC403*, pages 237–241, 2003.

[16] Stelting Stephen, Maassen, and Olav. *Applied Java Patterns*. Prentice-Hall PTR, first edition, 2001. ISBN: 0-13-093538-7.

[17] Meier Wolfgang. exist [on line]. Available at `http://exist.sourceforge.net//` [Last viewed: April 10, 2009].

[18] Meier Wolfgang. exist: An open source native xml database. *Web-Services, and Database Systems, NODe 2002 Web and Database-Related Workshops*, pages 169–183, 2002.