UOC

UNIVERSITAT OBERTA DE CATALUNYA

Segundo ciclo Ingeniería Informática

XML / XQuery Languages
Estado del arte

Alumne: Carlos Marín Muñoz
Dirigit per: Alex Alfonso Minguillón

CURS: Primer quatrimestre 2003-2004

A los creadores de los Simpsons y de Chin-Chan
por darme los únicos momentos de descanso y evasión durante todo este tiempo
A mis amigos, por su comprensión ante tantos desplantes
A la UOC, por ser la herramienta que me ha permitido alcanzar esta meta

Resumen de la memoria

El presente trabajo ha representado una síntesis de recopilación de información del estándar XML y los lenguajes de consultas XQuery, lenguajes de consultas diseñados para realizar accesos sobre este tipo de datos.

Se pretendía con este trabajo, y eso espero haber conseguido, hacer un análisis de cómo se encuentra actualmente el estándar XML y sobretodo como las bases de datos actuales, empiezan a asumir, si más no mediante herramientas y funcionalidades externas, la importancia de este estándar.
Así inicialmente, se plantea una primera parte de estudio, análisis y recopilación de información sobre XML, los principales elementos de este estándar y como no, dos de los más importantes estándares de transformación, el XSLT y el Xpath. La idea de esta primera parte es introducir al posible lector en el estándar de una manera clara y, espero, sencilla.

Ya he comentado que el mayor peso de este trabajo se lo ha llevado la relación que XML tiene con las bases de datos, es por este motivo que una segunda parte de la memoria se centra exclusivamente en los lenguajes de consultas sobre XML, los XQuery Languages, donde especialmente nos centramos en el estándar XQuery 1.0 del W3C exponiendo y detallando toda la sintaxis de este lenguaje.

Un análisis teórico sin una parte práctica no es completo, y de ahí la importancia de la tercera parte del trabajo, ver, analizar y comprobar (aunque esto haya llevado en alguna ocasión a la desilusión) como dos bases de datos totalmente diferentes, como es Oracle 9i, inminentemente relacional, y Tamino de AG, base de datos nativa, permiten, en mayor o menor medida el trabajo con el lenguaje XML. Esta parte del trabajo ha sido por descontado la más interesante de todas, ya que habiendo asentado los principios teóricos del XML y los lenguajes de consultas, es donde se pueden detectar las principales ventajas e inconvenientes que representa trabajar con una base de datos relacional o con una base de datos nativa.
Inicialmente la idea era realizar un caso práctico, el mismo, en ambos sistemas de bases de datos, pero debido a problemas surgidos a la hora de instalar y sobretodo configurar el Tamino, el análisis de Tamino ha tenido que hacerse un poco más teórico, algunos incluso pensaran que excesivamente teórico; pero creo que se ha conseguido que el usuario se haga una idea del funcionamiento de Tamino mediante el ejemplo práctico que se ha desarrollado.

De esta forma, en el primer capítulo se sintetiza los aspectos básicos del estándar XML haciendo una especial incidencia en el motivo que lo ha llevado y lo está llevando cada vez más a ser la elección más correcta e idónea como estándar de transmisión de información e intercambio. También es este capítulo se detallan los elementos básicos de la sintaxis del XML incluyendo un análisis exhaustivo sobre los archivos de definición de datos o DTDs y los XML Schemas. Por último se recogen los principales estándares de transformación para XML, el XSLT y el XPath.

En el segundo capítulo nos centramos en los lenguajes de consulta XQuery Languages, detallando las características que deben tener estos lenguajes para más tarde comentar brevemente cada uno de los que actualmente existen y comparándolos entre ellos

En el capitulo tercero se especifica detalladamente la sintaxis del lenguaje de consultas XQuery 1.0 según las directrices y documentos de trabajo del World Wide Web Consortium. Se hace un detalle exhaustivo de las expresiones, operadores y funciones que soporta este lenguaje de consultas. El capítulo está repleto de ejemplos para clarificar los conceptos vistos.

Una vez se ha hecho el análisis teórico del lenguaje XML y los estándares asociados a él, en el capítulo cuarto nos centramos en como Oracle 9i trata la información XML. Primero nos centramos en los parsers y APIs en Java de que dispone Oracle para después ver las herramientas propias de Oracle como el XML SQL Utitlity y el tipo de dato XMLType. Durante todo el capítulo se trabaja de modo práctico a través de un ejemplo básico con el fin de clarificar los conceptos.

Después de analizar como una potente base de datos relacional como es Oracle realiza el almacenamiento y acceso a datos XML, en el capítulo quinto nos centramos en analizar como una base de datos nativa, como es Tamino de AG Software realiza el almacenamiento y el acceso a este tipo de datos. Debido a lo novedoso de este tipo de bases de datos, se presenta un análisis más teórico en cuanto a arquitectura del sistema Tamino, componentes integrados y funcionalidades para acabar describiendo muy por encima los pasos para crear una base de datos nativa y la ejecución de consultas utilizando los dos lenguajes de consultas que Tamino soporta, el lenguaje de consultas XQuery 1.0 y el Tamino X-Query basado en Xpath 2.0

Ya por último, se presenta una sintetización de los dos sistemas vistos y que ventajas y/o desventajas aportan el uno sobre el otro.

Índice de contenidos

Índice de figuras

Introducción

1 Descripción del proyecto

El presente proyecto es la primera parte de un proyecto más amplio que se plantea el estudio del modelo de datos XML, de los diferentes lenguajes de consulta y de las diferentes bases de datos existentes en el mercado y que utilizan esta tecnología, siendo el objetivo principal del proyecto realizar un estudio sobre XML y su relación con el campo de las bases de datos.

En este proyecto mediante la recopilación de información se pretende realizar un estudio sobre XML, el lenguaje de consultas XQuery y de las diferentes bases de datos que existen en el mercado y que permiten utilizar XML como fuente de datos: tanto las bases de datos relacionales como las bases de datos nativas; en concreto, se centrará el estudio sobre Oracle 9i como sistema gestor de bases de datos relacional que permite el uso de XML y Tamino como base de datos nativa sobre XML.

1.1 Objetivos del proyecto

El objetivo principal de este proyecto es realizar un estudio sobre XML y su relación con el campo de las bases de datos, concretamente Oracle 9i y Tamino

Para la consecución de este objetivo se pretende:

- Recopilar información sobre el XML así como los estándares de conversión XSLT, XPath y DOM
- Recopilar información sobre los XML Query Languages haciendo especial incidencia en el lenguaje de consultas XQuery
- Analizar como un Sistema Gestor de Bases de Datos Relacional como es Oracle permite trabajar con datos XML
- Analizar como un Sistema de Bases de Datos Nativo como Tamino permite trabajar con datos XML

1.2 Resultados esperados

Una vez finalizado el proyecto se pretende:

a) Obtener un estudio detallado del lenguaje de marcas XML así como del lenguaje de consultas XQuery

b) Obtener un estudio comparativo sobre el tratamiento que Oracle 9i y Tamino hacen respectivamente del uso de XML como fuente de datos, haciendo especial incidencia en la arquitectura de cada una, características principales y ventajas que ofrecen respecto a la otra

c) Analizar mediante un caso práctico de creación, acceso y manipulación de datos XML las prestaciones que tanto Oracle 9i como Tamino ofrecen

1.3 Riesgos esperados

Actualmente el principal riesgo que se observa en este proyecto es sobretodo la escasa información que hay sobre el lenguaje de marcas XML y las bases de datos y por su lado la desmesurada información que hay para XML y la web.

Existe muchísima información sobre XML y la web (entorno tradicional en que el XML se ha desarrollado más contundentemente) y muchos aspectos que se tratan en esta documentación no se discernir si son adecuados para un proyecto como este en que solamente interesan los aspectos del XML relacionados con las bases de datos

Otro riesgo que observamos es que para el estudio de las bases de datos se ha optado por Oracle 9i como base de datos relacional y Tamino como base de datos nativa; es por todos conocida la gran complejidad y múltiples funcionalidades que Oracle ofrece en sus productos; por lo que el estudio sobre Oracle y XML no será trivial; pero además, hoy por hoy, desconozco totalmente la base de datos Tamino, con lo que el grado de trabajo y esfuerzo se verá incrementado.

2 Alcance de la propuesta

La presente propuesta cubre los siguientes contenidos:

1. Recopilación de información y estudio del XML
2. Recopilación de información y estudio del lenguaje de consultas XQuery
3. Estudio del tratamiento que las bases de datos hacen del XML
    a. Tratamiento de XML sobre Oracle 9i
    b. Tratamiento de XML sobre Tamino
4. Realización de unos casos prácticos a modo de ejemplo sobre las bases de datos analizadas

2.1 Recopilación de información y estudio del XML

El primer desarrollo del proyecto consistirá en la recopilación de información sobre el lenguaje de marcas XML para el posterior estudio.

Se pretenden tratar los siguientes aspectos del XML:

1. Origen.
    a. Intercambio de información en la web
    b. El XML y los SGBD
2. Objetivos
3. Sintaxis XML. XML bien formado
4. Las hojas de estilo en cascada (CSS)
5. Estándares de transformación para XML: XSLT y XPath
6. DTD

2.2 Recopilación de información y estudio del lenguaje de consultas XQuery

Se pretende a través de la recopilación de información del World Wide Web Consortium (www.w3c.org) obtener un completo estudio sobre la sintaxis de este lenguaje de consultas. También se pretende comentar cual ha sido la evolución de los XML query languages hasta el estándar Xquery 1.0 del W3C.

El desarrollo de este apartado incluiría:

1. Origen y objetivos
2. Evolución histórica
3. Comparativa entre los principales XQuery Languages
    a. Lorel
    b. XML-QL
    c. XQL
    d. XQuery
4. Sintaxis del XQuery 1.0
    a. Elementos básicos de lenguaje
    b. Expresiones
    c. Módulos y Prolog

2.3 Estudio de las bases de datos que soportan XML

Una vez se disponga de la información necesaria sobre XML y el lenguaje de consultas XQuery se pretende analizar el funcionamiento de las bases de datos que soportan XML. Para este estudio comparativo se ha elegido una base de datos relacional y una base de datos nativa, siendo las escogidas Oracle 9i y Tamino.

Se pretende analizar la arquitectura de estas bases de datos, características y realizar un exhaustivo estudio comparativo de las ventajas y desventajas que ofrecen una respecto a la otra.

2.4 Desarrollo de un caso práctico XML Oracle vs. Tamino

Por último, el proyecto finalizará con la realización de un caso práctico de transformación de datos entre documentos XML tanto en Oracle como en Tamino ofreciendo una exhaustiva información sobre la realización del caso ejemplo

3 Organización del proyecto

3.1 Descripción de actividades

3.1.1 Arranque del proyecto

El inicio del proyecto se llevará a cabo con la entrega del presente documento, aunque es importante destacar que el proyecto en sí comenzó durante el mes de julio, con la asignación del proyecto; durante ese mes se hizo mucho trabajo de recopilación de datos e información

3.1.2 Estudio del lenguaje XML

A partir de la información recopilada se realizará el estudio sobre el lenguaje de marcas XML haciendo especial incidencia en los aspectos relacionados con las bases de datos.
El principal problema de esta actividad va ser decidir que información, de la gran cantidad obtenida, es adecuada para su inclusión en el proyecto

Duración esperada: 3 semanas

3.1.3 Estudio del lenguaje de consultas XQuery

Al igual que en la tarea anterior, se pretende realizar el estudio sobre los XML query languages, haciendo especial incidencia en la evolución de estos. Finalizaremos el estudio mediante la realización de un detallado estudio sobre el lenguaje XQuery 1.0 del World Wide Web Consortium.

Duración esperada: 2 semanas

3.1.4 Estudio del funcionamiento de Oracle 9i con XML

En esta tarea se pretende a través de la lectura de la documentación de Oracle 9i y la realización de pequeñas pruebas-ejemplo comprender y documentar el funcionamiento que Oracle tiene respecto a XML

Duración esperada: 2 semanas

3.1.5 Estudio del funcionamiento de Tamino con XML

En esta tarea se pretende a través de la lectura de la documentación de Tamino y la realización de pequeñas pruebas-ejemplo comprender y documentar el funcionamiento que esta base de datos nativa tiene respecto a XML

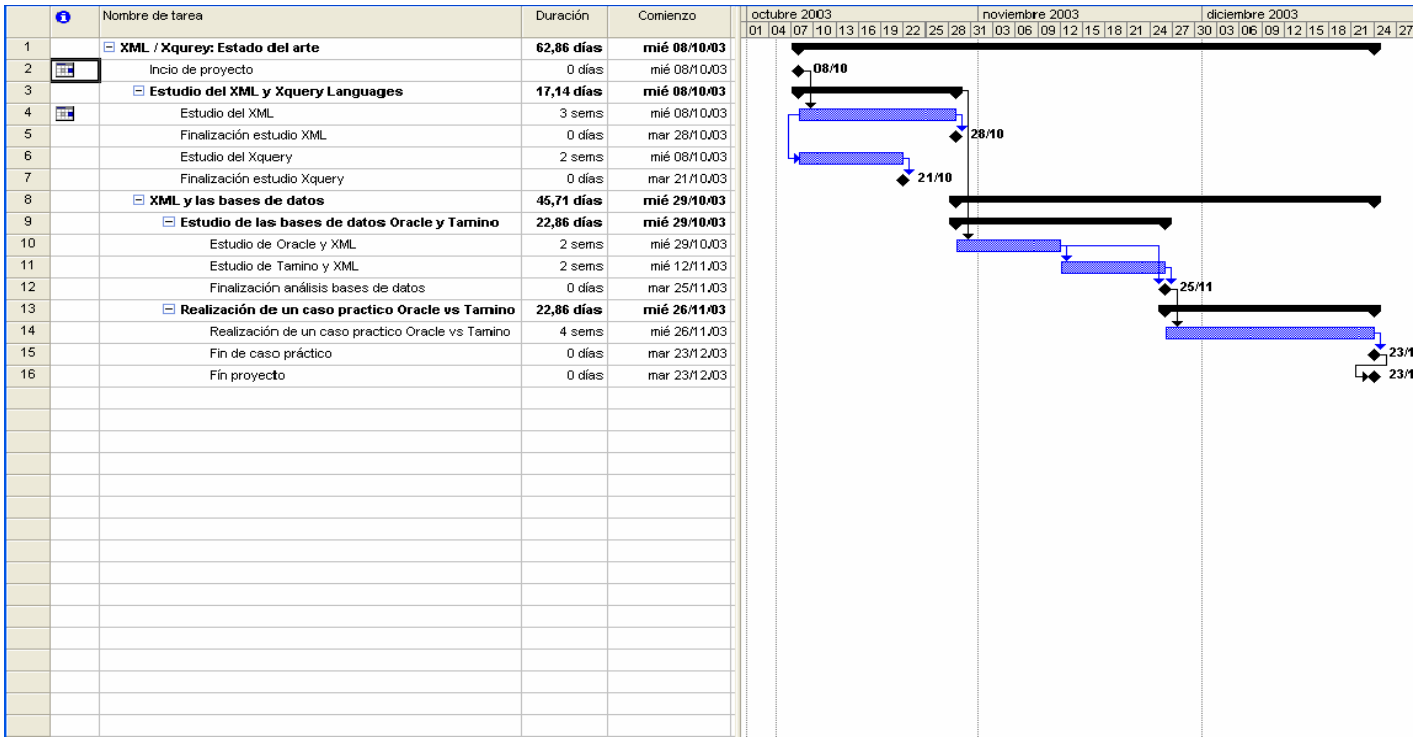
Duración esperada: 2 semanas

3.1.6 Realización de un caso práctico sobre XML en Oracle vs. Tamino

Por último se pretende realizar y documentar a través de un estudio comparativo la realización de un caso práctico en Oracle y Tamino con el fin de observar las diferencias que estas dos bases de datos ofrecen respecto al uso de XML

Duración esperada: 4 semanas

3.2 Calendario de Trabajo



	Nombre de tarea	Duración	Comienzo
1	**XML / Xqurey: Estado del arte**	**62,86 días**	**mié 08/10/03**
2	Incio de proyecto	0 días	mié 08/10/03
3	**Estudio del XML y Xquery Languages**	**17,14 días**	**mié 08/10/03**
4	Estudio del XML	3 sems	mié 08/10/03
5	Finalización estudio XML	0 días	mar 28/10/03
6	Estudio del Xquery	2 sems	mié 08/10/03
7	Finalización estudio Xquery	0 días	mar 21/10/03
8	**XML y las bases de datos**	**45,71 días**	**mié 29/10/03**
9	**Estudio de las bases de datos Oracle y Tamino**	**22,86 días**	**mié 29/10/03**
10	Estudio de Oracle y XML	2 sems	mié 29/10/03
11	Estudio de Tamino y XML	2 sems	mié 12/11/03
12	Finalización análisis bases de datos	0 días	mar 25/11/03
13	**Realización de un caso practico Oracle vs Tamino**	**22,86 días**	**mié 26/11/03**
14	Realización de un caso practico Oracle vs Tamino	4 sems	mié 26/11/03
15	Fin de caso práctico	0 días	mar 23/12/03
16	Fín proyecto	0 días	mar 23/12/03

3.3 Hitos Principales

Tal y como se observa en el calendario de trabajo anterior el proyecto se iniciará "oficialmente" el próximo 8 de Octubre de 2003 y finalizará el próximo 23 de Diciembre de 2003

Se han establecido los siguientes hitos en función de las actividades realizadas y concluidas

Hito 1: inicio del proyecto
Oficialmente el proyecto se iniciará con la entrega de este documento y debido a problemas profesionales comenzará el próximo día 8 de Octubre en lugar del 1 de Octubre como se indicó en un principio, de todas maneras destaco que mucha de las tareas relacionadas con la búsqueda de información sobre XML y XQuery ya fueron realizadas anteriormente

Hito 2: finalización del módulo sobre el lenguaje XML
Se incluye en esta hito las tareas de recopilación de información sobre XML así como su selección para la inclusión en la memoria del proyecto

Hito 3: finalización del módulo sobre el X Query Languages
Al igual que en las tareas anteriores se incluye la recopilación de información sobre los diferentes lenguajes de consultas sobre XML y su evolución así como la recopilación de la sintaxis de este lenguaje (extraída del WWW Consortium)

Hito 4: Finalización del estudio sobre el funcionamiento de Oracle y Tamino
Hito que contendrá las tareas relacionadas con el tratamiento que tanto Oracle 9i como Tamino hacen sobre datos XML: recopilación de información, lectura de documentación, pruebas,…

Hito 5: Estudio comparativo a partir de un caso práctico del tratamiento de XML que se hace en Oracle 9i y Tamino

Se presenta a continuación una posible temporalización de los diferentes hitos.

	Fecha finalización
Hito 1	8 de Octubre
Hito 2	28 de octubre
Hito 3	21 de octubre
Hito 4	25 de noviembre
Hito 5	23 de diciembre

El lenguaje de marcas XML

1 Origen

El origen de XML como estándar de intercambio de información se debe principalmente al hecho de una continua búsqueda de un estándar para el intercambio de información que no estuviese sujeto a un modelo o fabricante de software y que pudiese contener información adicional sobre la información del documento, sobretodo en Internet.
A partir de la fuerte apuesta que compañías como Microsoft, IBM y Netscape entre otras han hecho en los últimos tiempos por este estándar y debido a la gran proliferación de aplicaciones de la red vinculadas a accesos de Bases de Datos, es obvio destacar la importancia que el XML ha adquirido, y seguramente seguirá adquiriendo, como fuente de datos para las bases de datos. XML se utiliza para mejorar la comunicación de los datos y las bases de datos se centran en el almacenamiento y en la recuperación de los datos, esto hace que estas dos tecnologías se complementen entre si.

1.1 Intercambio de información el la red

Un archivo binario es el principal mecanismo de intercambio de datos. Cuando hablamos de un archivo binario, se entiende que es un archivo de datos en el que se han incluido ciertos códigos binarios que dan información sobre la información que contiene el archivo, los podemos considerar, en definitiva, metadatos. Por ejemplo en un archivo generado por un procesador de texto, los metadatos darán información sobre el formato del texto, caracteres especiales insertados, opciones de párrafos,…

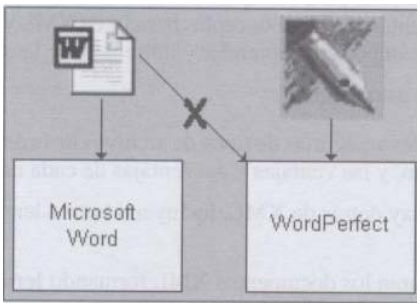


Figura 1.1. Problemas de intercambio de información mediante archivos binarios

La principal ventaja del uso de archivos binarios es que es totalmente idóneo para los ordenadores ya que pueden interpretar estos archivos a gran velocidad y con gran eficiencia al tener que trabajar sobre códigos binarios

El principal problema que plantea el uso de archivos binarios es, pues, que para su decodificación y posterior manipulación es necesario disponer de la misma aplicación que generó el archivo binario ya que los metadatos son diferentes en función de la aplicación que generó el archivo; así, por ejemplo, si disponemos de un documento generado por un determinado procesador de texto y almacenado en formato binario, necesitaremos para su posterior lectura y/u uso el mismo procesador que lo generó.

Los archivos de texto son también una buena elección como codificación para compartir información, en este tipo de archivos la secuencia de bits está agrupada siguiendo un estándar lo que lleva a que los archivos de texto puedan ser compartidos por multitud de aplicaciones, desde un simple editor hasta navegadores.

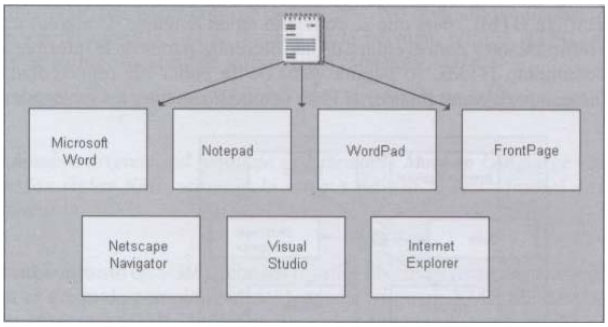


Figura 1.2. Elección de documentos texto para intercambio de información

La principal desventaja de este tipo de archivos es la enorme dificultad que hay de añadirle metadatos, o sea, información sobre la información que contiene el archivo, solo tendremos información sin ninguna orden adicional sobre formato y estilo.

Del intento de fusionar las ventajas que tienen estos dos tipos de archivos para su posterior intercambio de información nació SGML (Standard Generalized Markup Language), el primer gran intento de encontrar un lenguaje que fuese universalmente intercambiable entre diferentes aplicaciones informáticas. Su objetivo principal era combinar un formato de datos universal (archivos texto) con la posibilidad de almacenar información adicional sobre la presentación y formato.
A partir del SGML y debido a la excesiva complejidad de este lenguaje, y a pesar de los primeros esfuerzos por parte de muchas compañías de adoptarlo como estándar, poco a poco se fue dejando de utilizar.

Actualmente la aplicación mas difundida del SGML que encontramos es el HTML, es el lenguaje de marcas por definición, mucho más fácil y sencillo que SGML y universalmente reconocido en la web por todo tipo de navegadores o editores de páginas web, incluso actualmente tenemos procesadores de texto como MS Word y WordPerfect que permiten guardar los archivos con formato HTML.

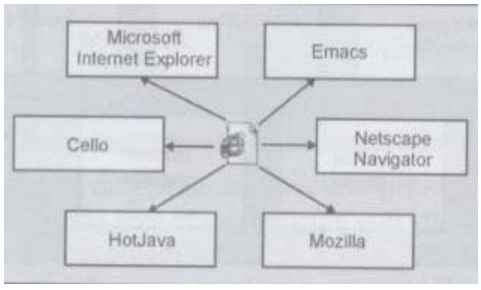


Figura 1.3 a. Uso de HTML como estándar de intercambio

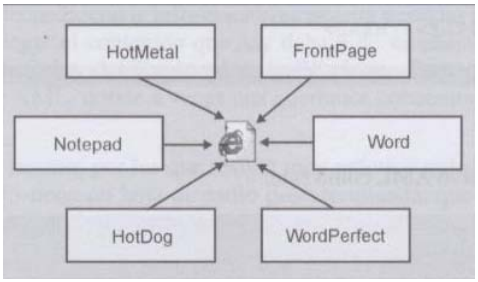


Figura 1.3 b. Uso de HTML como estándar de recepción de información

Las principales desventajas que encontramos en el HTML son dos:

- HTML no permite generar páginas dinámicas, solamente permite la creación de páginas estáticas, esto es, páginas que muestren información con un determinado formato. Mediante scripts, cgi's, javascripts,… se ha intentado solucionar esta limitación, pero lo único que se ha conseguido es desvirtuar el lenguaje HTML como lenguaje mezclando en el mismo archivo dos lenguajes diferentes
- La segunda desventaja está muy relacionada con el hecho de añadir a HTML mecanismos para dotarlo de dinamismo, y consiste en que al añadir este tipo de mecanismos a HTML muchas veces se pierda la portabilidad que este lenguaje tiene con las diversas plataformas, así es posible que una página a la que le añadamos código asp (active server pages) sigua funcionando en plataformas Windows pero no funcione en plataformas Unix

Es pues HTML un lenguaje únicamente de acceso y presentación de la información, HTML no puede representar el estándar para que el software pueda buscar, desplazar, presentar y manipular los datos; HTML no aporta ninguna funcionalidad referente a la administración de los datos, es por este motivo que aparece XML como subconjunto del estándar SGML; en definitiva, XML es el formato y el modelo para intercambiar información entre componentes, aplicaciones y empresas a través de Internet.
XML es un estándar abierto a Internet (W3C) que nos proporciona un formato para describir datos estructurados, facilitando realizar declaraciones más precisas de contenido y permitiendo obtener resultados de búsquedas más significativas.

1.2 El XML y las Bases de datos

El número de bases de datos, tanto relacionales como nativas que aceptan archivos XML como fuentes de datos cada día crece a más velocidad.
Tenemos así que los principales sistemas gestores de bases de datos relacionales representados por las principales compañías de desarrollo de bases de datos como Oracle 8i y 9i, DB2, Informix o Microsoft SQL Server 2000 aceptan ya archivos XML como fuentes de datos. Lo mismo ocurre con las bases de datos nativas, bases de datos que almacenan los datos XML de forma "nativa", generalmente como texto indexado, cada vez son más los productos que encontramos, podemos destacar 4Suite, DbXML, EXcelon, Lore o Tamino entre otros.

La elección de una base de datos relacional o una base de datos nativa vendrá determinada por el tipo de archivo con el que estemos trabajando; tenemos dos tipos de documentos XML: "basados en datos" y "basados en documentos".

Los documentos XML "basados en datos" son en los que XML es usado como un transporte de datos. Estos son por ejemplo órdenes de compra, registros de pacientes y datos científicos. Para almacenar y recuperar datos de un documento de este tipo necesitaremos una base de datos relacional.

Los documentos XML "basados en documentos" son aquellos en los que XML es usado para representar documentos, como un manual de usuario, páginas estáticas, folletos de marketing. Este último tipo de documento se caracteriza por su estructura irregular. Para el almacenamiento y recuperación de estos datos necesitaremos una base de datos de XML o nativa.

Ya se ha comentado anteriormente que XML es un estándar idóneo para el intercambio y transmisión de información y es por este motivo que conviene descubrir la relación del XML con las bases de datos.

Actualmente la mayoría de las aplicaciones se estructuran por capas, siendo la estructura más normal la que se muestra a continuación

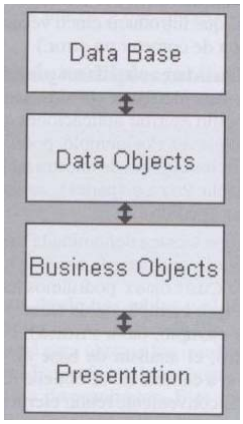


Figura 1.4. Arquitectura de las aplicaciones actuales

Estas capas no reflejan exactamente la distribución de la aplicación, los componentes de la aplicación pueden encontrarse en distintas máquinas o incluso en la misma.
Para aplicaciones más complejas el esquema anterior se refina como se muestra a partir de la división de las capas en sub-capas

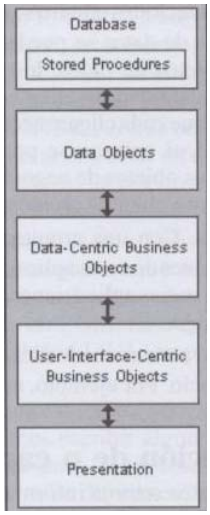


Figura 1.5. Arquitectura en capas y subcapas de las aplicaciones

Este enfoque en múltiples capas tiene muchas ventajas

- Se reduce la complejidad para los programadores
- Se facilita el proceso de cambios
- Las aplicaciones son más fácilmente escalables
- Reutilización de los componentes
- Mayor seguridad

En esta tipología de aplicaciones es fácil entender la necesidad de utilizar XML juntamente con las bases de datos; supongamos que la capa de presentación de nuestra aplicación está utilizando XML, lo lógico entonces, es que los componentes de la capa de negocio también estén utilizando XML para comunicarse entre los componentes de esta capa y para comunicarse con la capa de presentación. En este contexto, parece lógico plantearnos porque la capa de objetos, en lugar de trabajar con recordsets (objeto que encapsula los resultados obtenidos por una consulta SQL) porqué no hacer que esta capa devuelva datos XML. De la misma manera parece más sencillo que si hay que manipular y/o actualizar la base de datos, la capa de negocio haga una petición a la capa de objetos en XML y esta capa analice el código XML de la petición y realice las operaciones sobre la base de datos.

2 Objetivos

XML fue desarrollado por un grupo de trabajo bajo los auspicios del consorcio World Wide Web (W3C) a partir de 1996. El World Wide Web Consortium fue constituido en 1994 con el objetivo de desarrollar protocolos comunes para la evolución de Internet.

Se trata de un consorcio de la industria internacional con sedes conjuntas en el Instituto Tecnológico de Massachussets, de Estados Unidos, el Instituto Nacional de Investigación en Informática y Automática europeo y la Keio University Shonan Fujisawa Campus de Japón. El W3C tiene como misión la publicación para uso público de protocolos o estándares globales de uso libre.

Al comenzar el proyecto, los objetivos planteados por el grupo de desarrollo del XML se resumieron en estos diez:

1. XML debe ser directamente utilizable sobre Internet.
2. XML debe soportar una amplia variedad de aplicaciones.
3. XML debe ser compatible con SGML.
4. Debe ser fácil la escritura de programas que procesen documentos XML.
5. El número de características opcionales en XML debe ser absolutamente mínimo, idealmente cero.
6. Los documentos XML deben ser legibles por los usuarios de este lenguaje y razonablemente claros.
7. El diseño de XML debe ser formal, conciso y preparado rápidamente.
8. XML debería ser simple pero perfectamente formalizado.
9. Los documentos XML deben ser fáciles de crear.
10. La brevedad en las marcas XML es de mínima importancia.

Ahora, pues, ya estamos en condiciones de definir que es el XML. XML es un lenguaje de marcas que ofrece un formato para la descripción de datos estructurados, el cual conserva todas las propiedades importantes del antes mencionado SGML. Es decir, XML es un metalenguaje,

dado que con él podemos definir nuestro propio lenguaje de presentación y, a diferencia del HTML, que se centra en la representación de la información, XML se centra en la información en si misma.

La particularidad más importante del XML es que no posee etiquetas prefijadas con anterioridad, ya que es el propio diseñador el que las crea a su antojo, dependiendo del contenido del documento. De esta forma, los documentos XML con información sobre libros deberían tener etiquetas como <AUTOR>, <EDITORIAL>, <Nº_DE_PÁGINAS>, <PRECIO>, etc., mientras que los documentos XML relacionados con educación incluyen etiquetas del tipo de <ASIGNATURA>, <ALUMNO>, <CURSO>, <NOTA>, etc.
Por ejemplo en la siguiente tabla se muestra la información incluida por un código típico HTML y su versión equivalente en XML.

HTML	XML

<TABLE>
  <TR>
  <TD>Título**</TD>**
  <TD>Autor**</TD>**
  <TD>Precio**</TD>**
  </TR>
  <TR>
  <TD>AutoSketch**</TD>**
  <TD>Ramón Montero**</TD>**
  <TD>33**</TD>**
  </TR>
  <TR>
  <TD>Windows 98**</TD>**
  <TD>Jaime Perez**</TD>**
  <TD>3.250**</TD>**
  </TR>
  <TR>
  <TD>Web Graphics**</TD>**
  <TD>Ron Wodaski**</TD>**
  <TD>8.975**</TD>**
  </TR>
</TABLE>
	<LIBROS>
	  <LIBRO>
	  <TITULO>AutoSketch**</TITULO>**
	  <AUTOR>Ramón Montero**</AUTOR>**
	  <PRECIO>33**</PRECIO>**
	  </LIBRO>
	  <LIBRO>
	  <TITULO>Windows 98**</TITULO>**
	  <AUTOR>Jaime Perez**</AUTOR>**
	  <PRECIO>3.250**</PRECIO>**
	  </LIBRO>
	  <LIBRO>
	  <TITULO>Web Graphics**</TITULO>**
	  <AUTOR>Ron Wodaski**</AUTOR>**
	  <PRECIO>8.975**</PRECIO>**
	  </LIBRO>
	</LIBROS>

Figura 1.6. Equivalencia entre documento HTML y documento XML

Se puede apreciar en este ejemplo, que es mucho más fácil de entender la representación en XML. Mientras que en el documento HTML la información adquiere significado semántico dependiendo donde se encuentre ubicado en la tabla a la hora de presentar la información al usuario, en el documento XML la información tiene significado semántico propio debido a las marcas que dan información adicional a los propios datos del documento. Es el creador del documento XML el que decide qué tipos de datos va ha utilizar y qué etiquetas son las más adecuadas para ese documento en concreto

3 Sintaxis XML. XML bien formado

Hasta ahora se ha analizado por que XML es una buena alternativa cuando se desean comunicar datos. En este apartado veremos la sintaxis para la correcta creación de documentos XML. Un documento XML diremos que está bien formado si cumple las reglas gramaticales definidas en la especificación XML 1.0 del W3C.

3.1 Elementos

Los elementos son las unidades básicas que permiten construir documentos. Cada elementos representa un componente lógico del documento.

Para describir los elementos se utilizan etiquetas. Una etiqueta consta del nombre del elemento y opcionalmente de una lista de atributos que describen propiedades del elemento.

El principio de un elemento se marca de la siguiente manera:
<nombre_tipo_elemento [atributos]>

El final de un elemento se marca de la siguiente manera:
</nombre_tipo_elemento>

Así:
<first> es una etiqueta de inicio
</first> es una etiqueta de fin

<first> John </first> es un elemento

o con el caso de elementos con atributos:

```
<mensaje lenguaje”en”> Good morning, Vietnam </mensaje>
```

En el caso de los elementos vacíos, sin contenido, se marcan de la siguiente manera:

<nombre_tipo_elemento/>

El texto que se incluye entre las etiquetas de inicio y etiqueta de fin se denomina contenido de elemento. El contenido entre las etiquetas casi siempre serán simplemente datos, en este caso se le denomina PCDATA

Existen algunos caracteres reservados que no se pueden incluir en nuestro PCDATA porque son caracteres que se usan en la sintaxis XML, por ejemplo los caracteres ‘<’, ‘>’ o ‘&’
Así el siguiente PCDATA daría un error:

```
<!-- Esto no es un XML bien formado -->
<comparacion>6 es < 7 & 7  > 6</comparacion>
```

Para resolver esta clase de problemas debemos codificar los caracteres reservados:

```
<!-- Esto es un XML bien formado -->
<comparacion>6 es &lt; 7 &amp; 7  &gt; 6</comparacion>
```

Las codificaciones para algunos de los caracteres reservados serían:

Carácter Reservado	Codificación
<	&lt;
&	&amp;
>	&gt;
“	&quot;

Las reglas para los elementos que se deben conocer para que se consideren bien formados son:

- Cada etiqueta de inicio debe tener su etiqueta de fin correspondiente
- Las etiquetas no se deben superponer
- Los documentos XML solo deben tener un elemento raíz
- Los nombres de los elementos deben cumplir las convenciones de nomenclatura de XML
- XML diferencia entre mayúsculas y minúsculas
- XML mantendrá los espacios en blanco en el texto

3.2 Atributos

Los atributos describen las propiedades de los elementos, representan información adicional, son simples pares nombre/valor separados por un signo = que se asocian a un elemento. El valor del atributo ha de ir entre comillas simples o dobles.
Los atributos se adjuntan en la etiqueta de inicio, tal como se muestra a continuación:

```
<name apodo='Shiny John'>
        <first>John</first>
        <middle>Fitzgerald Johansen</middle>
        <last>Doe</last>
</name>
```

3.3 Comentarios

Los comentarios nos ofrecen el modo de insertar texto dentro del documento XML que en realidad no forma parte de documento pero por algún motivo es adecuado para facilitar la legibilidad del documento.
Los comentarios empiezan con la cadena <!-- y finalizan con la cadena --> .

Veamos un ejemplo:

```
<name apodo='Shiny John'>
        <!-- nombre de John-->
        <first>John</first>
        <!-- segundo nombre de John-->
        <middle>Fitzgerald Johansen</middle>
        <!-- apellido de John-->
<last>Doe</last>
</name>
```

3.4 Elementos vacíos

Algunas veces nos encontraremos con elementos XML que no tienen datos. Supongamos el ejemplo anterior, en que John Doe no tiene segundo nombre, tendríamos algo así:

```
<name apodo='Shiny John'>
        <first>John</first>
        <middle> </middle>
        <last>Doe</last>
</name>
```

En este caso tenemos la opción de rescribir la etiqueta <middle></middle> por <middle/>

3.5 Declaración XML

A menudo necesitaremos identificar un documento como perteneciente a cierto tipo, XML suministra la declaración XML para etiquetar a los documentos como XML, además de dar a los analizadores otras informaciones adicionales.
Una declaración típica sería:

```
<?xml version='1.0' encoding='UTF-16' standalone='yes' ?>
```

En la declaración XML aparecen ciertos detalles que se deben tener en cuenta:

- La declaración XML empieza con los caracteres <?xml, y finaliza con los caracteres ?>
- Si se incluye la declaración, esta obligatoriamente debe contener la versión, pero los atributos encoding y standalone son opcionales
- Los atributos version, enconding y standalone deben aparecer en este orden
- Actualmente, la versión debería ser la 1.0. Si se utiliza otra versión los analizadores escritos para la especificación 1.0 rechazarán el documento
- La declaración XML debe estar al principio del archivo.

3.6 Instrucciones de procesamiento

Algunas veces se precisará incluir instrucciones para una aplicación específica dentro del propio documento para indicar que se realice algún proceso, aunque esto no suele ser habitual. XML suministra un mecanismo para poder realizar esto, denominado Instrucciones de procesamiento o Proccessing Instructions que nos permitirán incluir instrucciones dentro del código XML, pero que en realidad no forman parte del documento sino que se pasan directamente a la aplicación.

```
<?xml version='1.0' encoding='UTF-16' standalone='yes' ?>
<name apodo='Shiny John'>
        <first>John</first>
        <middle>Fitzgerald Johansen</middle>
        <?nombreaplicacion SELECT * FROM blah?>
        <last>Doe</last>
</name>
```

Normalmente las instrucciones de procesamiento siguen la siguiente sintaxis, empiezan con el carácter <? seguido del nombre de la aplicación que recibirá la instrucción y a continuación la instrucción a ejecutar seguida de los parámetros o atributos de la instrucción, si los tuviera.

El uso de instrucciones de procesamiento suele ser poco común y se suele evitar su uso siempre que se pueda.

3.7 El árbol XML

Un documento XML también se puede representar como un árbol además de poderlo representar como una lista secuencial de información.
De hecho, la regla que dictamina que en un documento XML solo puede haber un elemento que englobe a todos los otros nos permite que el documento XML pueda ser representado como un árbol con una única raíz.

Los analizadores sintácticos XML, y en general todas las herramientas informáticas que tratan documentos XML, construyen una representación del documento en forma de árbol.

El árbol XML se construye de la siguiente forma:

- Los elementos del documento XML se corresponden con los nodos del árbol
- El nodo que representa un elemento A es el nodo padre de otro elemento B si B esta contenido dentro de A
- Los nodos hermanos, situados en el mismo nivel del árbol, están ordenados según el orden de aparición en el documento
- Los datos de carácter (PCDATA) del documento XML se consideran nodos del árbol etiquetados con el nombre Text
- Los atributos se consideran información asociada a cada nodo del árbol, y por lo tanto no aparecerán en la representación en forma de árbol.

Veamos un ejemplo:

```
<name apodo='Shiny John'>
        <first>John</first>
        <middle>Fitzgerald Johansen</middle>
        <?nombreaplicacion SELECT * FROM blah?>
        <last>Doe</last>
</name>
```

Su representación en forma de árbol sería:

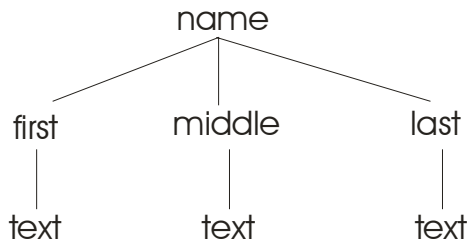


Figura 1.7. Representación en árbol de un documento XML

4 Las hojas de estilo en cascada (CSS)

Las hojas de estilo sirven para mejorar la expresividad de los datos XML, y las hojas de estilo en cascada (Cascading Style Sheet, CSS), se utilizan para establecer las características visuales de la información, como por ejemplo, tamaño de la fuente y tipo de fuente, etc.

La primera cuestión que nos surge cuando se empieza a trabajar con XML es sobre como el navegador sabe como deben visualizarse las etiquetas arbitrarias que aparecen en un documento XML. Los navegadores no reconocen etiquetas como <name>, <first> o <middle>, así pues,

este fue el primer gran conflicto que tuvieron los navegadores cuando se empezó a desarrollar XML.

4.1 Concepto

XML aporta la separación de la recuperación de la información del proceso de la presentación de la misma, este modelo conocido comúnmente como paradigma Contenido/Presentación significa que se separan los datos de la forma en que se visualizarán.

En HTML disponemos de herramientas para la creación de estilos (etiqueta <STYLE>) y otros elementos para la selección de atributos para la representación de la información (etiqueta <FONT>, etiquetas de cabecera del tipo <HX>,…

Veamos un ejemplo para ver como se crean los estilos en HTML:

```
<H1 style=  “voice-family:PDomingo;volumen:x-loud;”>
        Bienvenido a mi web site
</H1>
<p style=  “voice-family:MCallas;volumen:x-medium;”>
        Este sería un website en que el contenido nos sería leído utilizando voces diferentes
</p>
```

Como se puede observar el atributo style aporta un estilo determinado al elemento que queramos aplicar.
Existe, de todas maneras, otra opción cuando trabajamos con XML, el elemento <STYLE> nos permite asociar un estilo determinado a un elemento específico

```
<STYLE>
H1      { voice-family:PDomingo;volumen:x-loud;}
        p       { voice-family:MCallas;volumen:x-medium;}

</STYLE>

<H1>Esto se lee con la voz de Placido Domingo</H1>
<P> Esto se lee con la voz de Maria Callas</P>
<P> y esto también </P>
<H3> Esto en cambio no se leerá ya que no existe una definición de una regla </H3>
```

Tenemos que la declaración CSS para cada tipo se denomina regla, y la colección de todas las reglas de la etiqueta <STYLE> se denomina hoja de estilo.

En nuestro ejemplo anterior, tenemos que { voice-family:PDomingo;volumen:x-loud;} y { voice-family:MCallas;volumen:x-medium;} serán las reglas y H1 y P son los selectores de las reglas

4.2 Hojas de estilo en cascada y XML

Las hojas de estilo en cascada implica que cada elemento hijo contiene todos los elementos que están en un nivel superior al que está el propio elemento. Podemos entender que cuando se aplica un estilo a un elemento, este estilo afecta a todos los elementos que estén dentro de este elemento, a todos los elementos hijos de ese elemento al que se le ha aplicado el estilo.

Cuando a un elemento se le aplica un estilo, muchas de las propiedades se aplican en cascada a todos los elementos hijo de ese elemento, a no ser que el elemento o elementos hijos no definan explícitamente un estilo propio con la misma propiedad CSS y diferente valor.

La diferencia entre CSS en HTML y CSS en XML es tan pequeña que los mismos conceptos aplicados en hojas de estilo en HTML nos sirven para XML.

La primera diferencia que encontramos es que las etiquetas propias de HTML para asignar estilos y clases en XML no existen; para asignar estilos en XML debemos utilizar la instrucción de procesamiento <?xml-stylesheet ?>
La sintaxis completa de la instrucción xml-stylesheet sería:

```
<?xml-stylesheet type=nombretipo href=urlhojaestilo?>
```

En realidad esta sintaxis se parece mucho al elemento LINK de HTML, en este caso type hace referencia a cualquier lenguaje de hoja de estilo que soporte el navegador, en general encontraremos "text/css" o "text/xsl". Href hace referencia a la ubicación externa o referencia de la hoja de estilo CSS que se desea utilizar.

Veamos un ejemplo:

```
<?xml-stylesheet type="text/css" href="nota1.css"?>
<!-- Nota1.xml -->
<documento>
        <body>
                advertencia>Esto es una advertencia</advertencia>
        En caso contrario
                consejo>Esto es un consejo</consejo>
        ,mientras que < recurso >esto es un recurso</ recurso >
        </body>
</documento>
```

Como vemos, este documento nota1.xml hace referencia hace referencia a una hoja de estilo CSS llamada nota1.css, el cual podría contener:

```
consejo         margin-left:.5in;width:250px;border:solid 3px
        black;position:relative; background-color:yellow }

recurso         margin-left:.5in;width:250px;border:solid 3px
        black;position:relative; background-color:green }

advertencia     margin-left:.5in;width:250px;border:solid 3px
        black;position:relative; background-color:red}
```

Como vemos, esta hoja de estilo CSS para XML es casi idéntica a las hojas de estilo CSS para HTML y sencillamente lo que se hace es asignar unos determinados parámetros de visualización a cada una de las etiquetas del documento XML.

Hasta ahora hemos visto como podíamos utilizar hojas de estilo externas al documento XML mediante el atributo href de la instrucción xml-stylesheet. Este tipo de hojas de estilo son muy prácticas para mantener el mismo estilo en varios documentos XML; pero habrá algunas veces en que resulta más útil trabajar con estilos dentro de los documentos XML.

Ya se ha comentado que la etiqueta <STYLE> que utilizaríamos en HTML en XML no tiene ningún significado específico, por lo tanto para incluir reglas de hojas de estilo en un elemento dentro del mismo documento deberemos aplicar la siguiente solución: mediante un identificador

determinado incluiremos las reglas de hoja de estilo a un elemento y referenciaremos a la hoja de estilo mediante el identificador de la instrucción xml-stylesheet
Veamos un ejemplo:

```
<?xml-stylesheet          type="text/css"          href = "#miEstilo"?>
<documento>
        <st id="miEstilo">
                title       {font-size:24pt;display:block;}
                p           {font-size:11pt; display:block;}
                p:first-letter      {float:left;font-size:36pt;}
                st          {display:none;}
                b           {font-weight:bold;}
        </st>
        <title>Hojas de estilo CSS dentro de los documentos XML</title>
<p>Se utiliza el signo # para indicar que se hace referencia a una <b>hoja de estilo</b> dentro del documento</p>
</documento>
```

En este caso la hoja de estilo está contenida en el nodo <st> pero está referenciada mediante su identificador miEstilo. El modo de funcionamiento del símbolo # en la declaración xml-stylesheet es similar a los enlaces o hipervínculos relativos a una página en HTML.

5 Definición de tipo de documento. Los DTD's

Lo primero que debemos saber es que hay dos tipos de documentos XML: **válidos y bien formados**. Éste es uno de los aspectos más importantes de este lenguaje, así que hace falta entender bien la diferencia:

- **Bien formados:** son todos los que cumplen las especificaciones del lenguaje respecto a las reglas sintácticas que después se van a explicar, sin estar sujetos a unos elementos fijados en un DTD. De hecho los documentos XML deben tener una estructura jerárquica muy estricta, de la que se hablará más tarde, y los documentos bien formados deben cumplirla.
- **Válidos:** Además de estar bien formados, siguen una estructura y una semántica determinada por un DTD: sus elementos y sobre todo la estructura jerárquica que define el DTD, además de los atributos, deben ajustarse a lo que el DTD dictamine

El DTD (definición del tipo de documento, Document Type Definition) proporciona la gramática para una clase de documentos XML. Esta gramática contiene la definición del conjunto de etiquetas que puede contener esa clase de documentos XML, los nombres que pueden utilizarse en los elementos, dónde pueden aparecer y cómo se interrelacionan entre ellos. Se puede decir que un DTD es una definición exacta de la gramática de un documento. Estas normas o gramática se expresan mediante una notación llamada **declaración de marcaje**

La declaración de marcaje es una notación descrita en la especificación XML (de W3C) siendo una manera de expresar un conjunto de reglas gramaticales que permiten indicar:

- Los elementos y atributos válidos dentro de un documento XML
- Los elementos que se pueden utilizar dentro de otros elementos
- Los elementos y los atributos opcionales

Los documentos XML enviados con un DTD se reconocen como "XML válido". En este caso, un intérprete de XML podría comparar los datos entrantes con las normas definidas en el DTD para comprobar que los datos se han estructurado correctamente. Los datos enviados sin un DTD se reconocen como "bien formados". En XML no existen DTDs predefinidos, por lo que es labor del diseñador especificar su propia DTD para cada tipo de documento XML. En la especificación de XML se describe la forma de definir DTDs particularizadas para documentos

XML, que pueden ser internas (cuando van incluidas junto al código XML) o externas (si se encuentran en un documento propio).

Un ejemplo de un DTD que defina la estructura de un documento XML relacionado con libros, podría ser un documento conteniendo el siguiente código:

```
<!ELEMENT LIBROS (LIBRO)+>
<!ELEMENT LIBRO (TITULO, AUTOR, PRECIO)>
<!ELEMENT TITULO (#PCDATA)>
<!ELEMENT AUTOR (#PCDATA)>
<!ELEMENT PRECIO (#PCDATA)>
```

En el DTD del ejemplo se definen los elementos (!ELEMENT) que integran el documento XML. En la primera línea se indica que el elemento principal es LIBROS, del que dependen uno o más (+) elementos LIBRO. La segunda línea sirve para especificar que el elemento LIBRO contiene tres elementos (TITULO, AUTOR, PRECIO) que se deben marcar en dicho orden. Las restantes líneas aclaran que los elementos TITULO, AUTOR y PRECIO contienen valores de cadenas de texto, como ya habíamos comentado PCDATA's

Como síntesis de lo descrito podemos decir:

- Crear un DTD (Definición de tipo de Documento), es como crear nuestro propio lenguaje de validación para una aplicación específica
- DTD es también un archivo de texto
- DTD contiene
    - Tipos de elementos
    - Atributos
    - Entidades
    - Restricciones de combinación y valores que contendrá el XML
- Solo lo que se contiene es válido
- Un DTD puede residir en un archivo externo (quizá compartido por varios documentos XML), o bien puede estar contenido en el propio documento XML
- Los documentos que se ajustan a su DTD se denominan *Válidos*.
- Una vez bajado un documento XML, el procesador sintáctico (Aplicación o navegador), puede aplicar un DTD para conocer si cumple con las normas establecidas
- No confundir 'bien formado' con 'válido'
    - Bien formado es que respeta la sintaxis de XML
    - Válido es que respeta las reglas del DTD

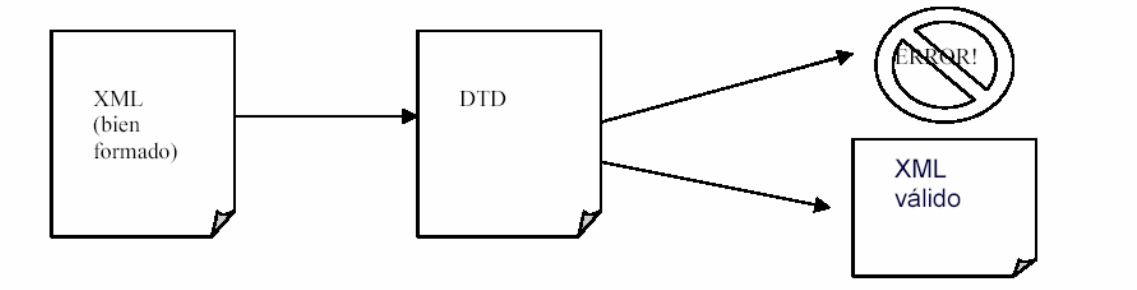


Figura 1.8. Esquema de un documento XML válido

- Un documento XML puede ser válido y bien formado a la vez
- El DTD es opcional

- No se puede decir que el documento es válido si no se utilizan DTD's (pero tampoco que sea no válido)
- ¿Que ventajas tiene?
    - Pueden ser publicados o compartidos por varias empresas para definir 'sus estándares', pudiéndose generar de esta forma, un estándar de validaciones
    - Un programa emite un XML, y otro lo consume, pero validando que cumpla con el contrato establecido
    - Muchas empresas están publicando estándares de validación, los cuales pueden ser reutilizados
- Si bien pueden residir en el mismo documento, es posible su enlace externo para su reutilización

```
<?xml version="1.0"?>
<!DOCTYPE persona SYSTEM http://www.lapiedra.com/persona.dtd>
```

5.1 Sintaxis de los documentos DTD

5.1.1 Declaración de elementos

Para declarar elementos en un documento DTD utilizaremos la expresión:

<!ELEMENT element-name category>

donde category podrá tomar los valores EMPTY, #PCDATA o ANY.

<!ELEMENT element-name EMPTY>	Permite declarar elementos que no tienen contenido
	Ejemplo DTD: <!ELEMENT br EMPTY>
	Ejemplo XML:

<!ELEMENT element-name (#PCDATA)>	Permite declarar elementos que tienen como contenido una cadena de caracteres
	Ejemplo DTD: <!ELEMENT nombre (#PCDATA)>
	Ejemplo XML: <nombre>Johann<nombre/>

<!ELEMENT element-name ANY>	Permite declarar elementos que pueden tener como contenido cualquier otro elemento
	Ejemplo DTD: <!ELEMENT libro ANY>
	Ejemplo XML: <libro> … <titulo> … <autor> … <autor/> … <titulo/> …<libro/>

También podemos declarar elementos como:

<!ELEMENT element-name (element-content)>

Donde element-content especifica las etiquetas que puede contener el elemento element-name. Existe un conjunto de operadores que permiten indicar la forma en que puede aparecer cada elemento:

Lista de Operadores	
,	Indica una secuencia de elementos Ej.: (nombre-elemento1, nombre-elemento2,nombre-elemento3)
|	Indica el operador OR Ej.: (nombre-elemento1 | nombre-elemento2)
*	Indica la aparición de 0 a N elementos Ej.: (nombre-elemento*)
+	Indica la aparición de 1 a N elementos Ej.: (nombre-elemento+)
?	Indica la aparición de 0 o 1 elementos Ej.: (nombre-elemento?)

Algunos ejemplos serían:

```
<!ELEMENT LIBROS (LIBRO)+>

<!ELEMENT LIBRO (TITULO, AUTOR*, PRECIO)>
```

5.1.2 Declaración de atributos

Para declarar los atributos que aparecerán en los documentos XML se realiza mediante el comando ATTLIST y cuya sintaxis es:

<!ATTLIST element-name attribute-name attribute-type default-value>

donde:

attibute-type indica el tipo de atributo que es. Podemos encontrar los siguientes valores:

CDATA	indica que el atributo es de tipo string
(en1 | en2 | …)	indica que el atributo es de tipo enumeración
ID	indica que el atributo es de tipo identificador
IDREF	indica que el atributo es de tipo referencia a un identificador de otro elemento

default-value da indicaciones sobre el valor del atributo, si es fijo o no y el tipo de requerimientos que tiene. Podemos encontrar los siguientes valores:

Value	indica el valor por defecto que tendrá el atributo
#REQUIRED	indica que el atributo es obligatorio
#IMPLIED	indica que el atributo no es obligatorio
#FIXED Value	Indica un valor fijo para el atributo

Veamos, ahora un ejemplo completo de DTD y un posible documento XML válido:

```
<!ELEMENT Company (Delegation+)>
<!ATTLIST Company
name CDATA #REQUIRED
founded CDATA #REQUIRED
income CDATA #REQUIRED
>
<!ELEMENT Delegation (#PCDATA)>
<!ATTLIST Delegation
name CDATA #REQUIRED
numberOfEmployees CDATA #REQUIRED
>
```

y el posible documento XML asociado al DTD anterior, podría ser:

```
<?xml version="1.0" encoding="UTF-8"?>
<Company name="ABC, S.A." founded="2001-01-01" income="10000">
<Delegation name="Central Madrid" numberOfEmployees="150">Texto asociado
Madrid</Delegation>
<Delegation name="Sucursal Jaén" numberOfEmployees="270">Texto asociado Jaén</Delegation>
</Company>
```

5.2 XML Schema

A pesar de todo lo visto anteriormente, los DTD's tienen dos problemas importantes:

- Se especifican en un lenguaje totalmente diferente de XML
- Tienen muy pocos recursos para expresar tipos de datos.

Para solucionar estos dos problemas se generó la especificación XML Schema, que es una notación para escribir los DTD's alternativos a las declaraciones de marcaje y aporta básicamente tres ventajas:

- Es notación XML: esto hace que las herramientas que se utilizan para analizar los documentos XML puedan utilizarse para analizar los esquemas XML
- Soporta diferentes tipos de datos: los XML schema ofrecen múltiples tipos de datos ya definidos lo que permite una validación de documentos mucho mas eficaz
- Es extensible: además de los tipos de datos predefinidos permite nuevas definiciones de tipos de datos

Últimamente debido a las ventajas que se han comentado anteriormente y uniendo a los inconvenientes que presentan los DTD's, se está imponiendo esta otra forma mucho más eficaz de definir la estructura de un documento XML.

Un esquema, en definitiva, es una especificación formal de las normas de un documento XML, que indica qué elementos se permiten en un documento y en qué combinaciones están permitidas. Los nuevos lenguajes de esquemas, definidos en las propuestas XML-Data (Datos de XML) y DCD (Descripción del contenido del documento) enviadas por el XML-Data Working Group al W3C, proporcionan las mismas funciones que un documento DTD. No obstante, puesto que estos lenguajes de esquema son extensibles, los desarrolladores pueden ampliarlos con información adicional, como los tipos de datos, la herencia y las normas de presentación.
Esto hace que los nuevos lenguajes de esquemas sean mucho más potentes que los DTD.

La expresión de esquemas dentro de XML aumenta la potencia del formato XML, pues permite que el software examine determinados datos para comprender su estructura, sin necesitar ninguna descripción previa incorporada de la estructura de los datos. Con un esquema, un autor puede definir exactamente qué nombres de elementos se permiten en un documento y, dentro de cada elemento, qué sub-elementos, atributos y relaciones se admiten. El autor puede importar fragmentos de otros esquemas, así como ampliar tipos a través de la herencia. Todo ello permite establecer relaciones complejas entre los elementos sin perder la simplicidad de la estructura de árbol.

En resumen podemos decir:

- Al igual que los DTD's, definen los elementos, el orden, los rangos, etc.
- Los Schemas o esquemas son similares a los DTD's, entonces...
- ¿Por qué debería utilizar un esquema?
- Los esquemas utilizan sintaxis XML
- Tienen mejor y más manejo de los tipos de dato
- Manejan más reglas
- Los tipos de datos son extensibles
- Es mas fácil plasmar un esquema en una tabla, que hacer lo mismo con un DTD
- Microsoft e IBM están trabajando en conjunto para ofrecerle un borrador de estándar a W3C
- Se pueden agregar nuevos elementos, y seguir siendo compatible con lo 'anterior'
- A tener en cuenta...
    - El procesador puede no soportar esquemas
    - El esquema de Microsoft es parecido al esquema propuesto
    - Las extensiones de esquema de Microsoft solamente están disponibles bajo Windows

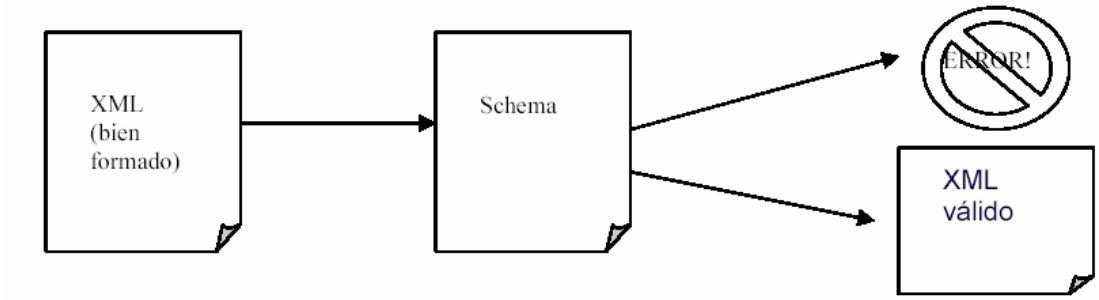


Figura 1.9. Esquema de un documento XML bien formado

5.2.1 Namespaces

Los namespaces se desarrollaron como un método para poder diferenciar elementos de un documento XML con el mismo nombre pero que aparecen en distintos niveles. Los namespaces nos proporcionan la forma de agrupar los elementos en un contexto específico.
Supongamos el siguiente documento XML:

```
<?xml version="1.0" encoding="UTF-8">
<car>
        <color>blue</color>
        <interior>
                <upholstery>leather</ upholstery>
                <color>tan</color>
        </interior>
</car>
```

Podemos observar que el documento anterior tiene dos marcas llamadas <color>. Un caso como este podría provocar confusión al analizador XML, la solución a este posible problema es definir nameespaces separados con el objetivo de distinguir los dos elementos dependiendo del contexto en que se han definido:

```
<veh:car
            xmlns:veh=http://mycarnamespace.com/cars/
            xmlns:vehinterior=http://myvehicleinteriors.com/interiors/">
          <veh:color>blue</veh:color>
          <vehinterior:interior>
                    <vehinterior:upholstery>leather</vehinterior:upholstery>
                    <vehinterior:color>tan</vehinterior:color>
          </vehinterior:interior>
</veh:car>
```

Observemos que se han declarado dos namespaces diferentes asociados a los prefijos veh (correspondiente al vehículo) y vehinterior (correspondiente a la información del interior del vehículo) haciendo desaparecer la ambigüedad.

5.2.2 Tipos de datos nativos

Como en otros lenguajes, XML Schema tiene un conjunto de tipos de datos primitivos de los que es posible obtener otros tipos de datos más complejos.

Tipos de datos	Descripción
string	representa cadenas de caracteres en XML
boolean	permite definir los elementos bolenaos true y false
decimal	define el conjunto de valores del tipo i*10-n (siendo i y n enteros tal que n>=0)
float	representa los números en coma flotante
double	representa los números con doble precisión IEEE, tipo flotante de 64 bits
duration	representa un longitud en el tiempo. El valor de la duración viene determinado por el año, mes, día, hora, minuto y segundo
dateTime	el tipo dateTime representa un instante de tiempo específico
time	Representa un instante de tiempo que ocurre cada día. Viene determinado por el espacio de tiempo en horas
date	El tipo date representa un instante de tiempo que ocurre cada año. El valor de date es el espacio de tiempo en días
gYearMOnth	Representa un mes gregoriano en un determinado año gregoriano dado
gYear	Representa un año del calendario gregoriano
gMonthDay	representa una fecha gregoriana que se repite del tipo un día del año (p.e. 1º de mayo)
gDay	representa un día gregoriano que se repite del tipo día del mes (p.e. el día 5 del mes)
gMonth	representa un mes gregoriano que se repite cada año (p.e. mayo)
hexBinary	representa datos representados en binarios
base64Binary	representa datos binarios codificados
QName	representa nombres XML cualificados es decir nombres XML válidos

6 Estándares de transformación para XML: XSLT y XPath

Un navegador, después de chequear la sintaxis del código del documento, debe presentar la información del documento con un formato determinado. Los documentos HTML utilizan las descripciones de formatos internas del propio navegador, o si existen descripciones CSS (que son opcionales), utilizan la información de la hoja de estilo para ajustar la presentación en la pantalla. Los documentos XML siempre necesitan normas que describan su presentación.

Para describir cómo se deben presentar los documentos XML podemos optar por dos soluciones: las mismas descripciones CSS que se utilizan con HTML (vistas en el apartado 3 de este mismo capítulo) y/o las descripciones que se basan en XSL (eXtensible Stylesheet Language).

Si ya existía una forma de definir las presentaciones de los documentos Web, el interrogante que puede surgir es cual fue el motivo que llevó a desarrollar otra forma específica para XML?

La respuesta es que CSS es eficaz para describir formatos y presentaciones, pero no sirve para decidir qué tipos de datos deben ser mostrados y cuáles no. Esto es, CSS se utiliza con documentos XML en los casos en los que todo su contenido debe mostrarse sin mayor problema.

XSL no solo permite especificar cómo queremos presentar los datos de un documento XML, sino que también sirve para filtrar los datos de acuerdo a ciertas condiciones. XSL se parece un poco más a un lenguaje de programación. XSL es una especificación del W3C que tiene como objetivo proporcionar una serie de reglas que permitan definir diferentes presentaciones para los documentos XML.

Como estándares de presentación encontramos dos bien importantes, XSLT (subcomponente del lenguaje XSL) y XPath basado en encontrar la coincidencia de las partes.

6.1 XSLT

El lenguaje de estilo extensible o ampliado (XSL) es un lenguaje basado en XML que se usa para crear hojas de estilo. Un motor de hoja de estilo usa hojas de estilo para transformar documentos XML en otros tipos de documentos y formatear la salida. En estas hojas de estilo se define el diseño del documento de salida y desde donde se obtienen los datos en el documento de entrada. En terminología XSL, el documento de entrada se denomina árbol de origen y el documento de salida se denomina árbol de resultado.

En XSL existen dos lenguajes completamente diferentes:

- Un lenguaje de transformación denominado XSLT
- Un lenguaje utilizado para describir la visualización de los documentos XML, que se denomina XSL Formatting Objects.

Las hojas de estilo XSLT se construyen sobre estructuras denominadas plantillas. Una plantilla especifica que es lo que hay que buscar en el árbol origen y que es lo que hay que colocar en el árbol resultado. El proceso de transformación consiste en ir recorriendo el documento XML original (que ya dijimos que se podía representar como un árbol) y para cada elemento que encuentra el procesador XSLT se comprueba si hay definida alguna transformación en el documento XSLT; si está se aplica al elemento en cuestión la transformación y si no la hubiera

se continua la exploración del documento. Las transformaciones se escriben en un archivo de salida que será el documento que se tendrá como resultado cuando el proceso de transformación acabe. Este documento también es tratado como una estructura de árbol.
Normalmente los documentos de salida son o bien ficheros HTML o bien texto regular.

6.1.1 Sintaxis del lenguaje de transformación XSLT

En este apartado vamos a ver cual es la sintaxis del lenguaje de transformación XSLT; solamente se pretende dar unas pequeñas nociones básicas sobre el lenguaje XSLT y no ser una exhaustiva descripción del lenguaje; para ello, a partir de un ejemplo de transformación XSLT veremos como se pueden activar las reglas XSLT y los elementos más significativos de este lenguaje.

Como se ha comentado anteriormente el lenguaje XSLT recibe como entrada un documento XML y genera como salida un documento HTML.
Podríamos decir que las hojas de estilo XSLT son simplemente una colección de plantillas que se aplican a los documentos XML de entrada para crear el documento de salida.

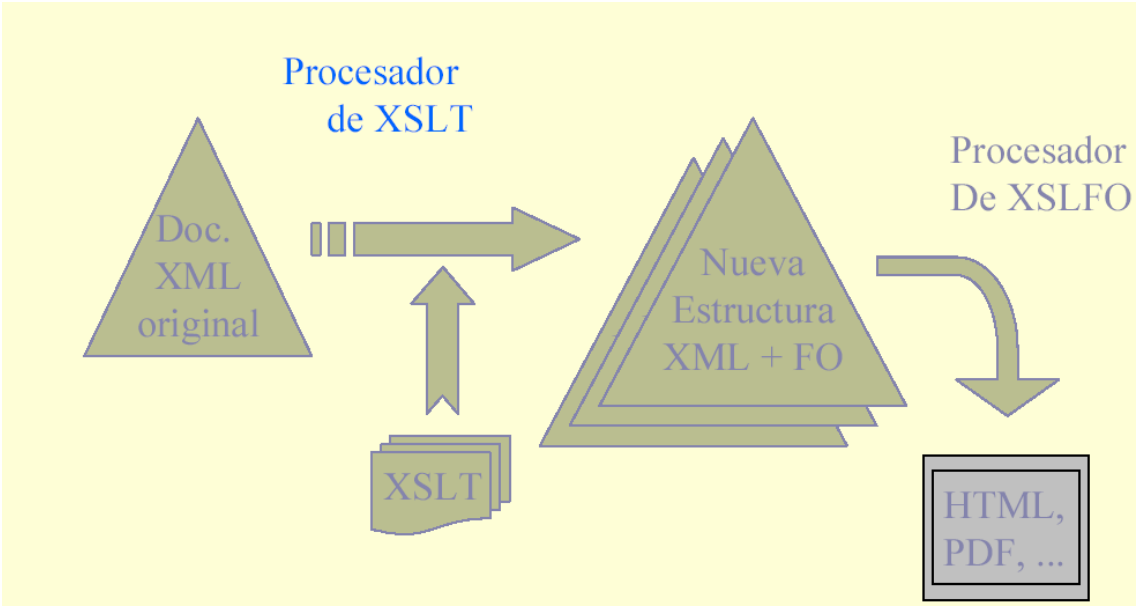


Figura 1.10. Funcionamiento del proceso de transformación XSLT

Supongamos para analizar nuestro ejemplo el siguiente documento original XML llamado mail.xml:

```
<?xml version="1.0"?>
<email>
  <from>carlos@ceslesheures.com</from>
  <to>cmarinmu@uoc.edu</to>
  <subject>Correo en XML</subject>
  <message>Esto es un documento XML que representa un e-mail</message>
</email>
```

El archivo de transformación XSLT (mail.xslt) podría ser:

```
<xsl:stylesheet version="1.0"
  xmlns:xsl= http://w3.org/1999/XSL/Transform>
        <xsl:output method="html"/>
        <xsl:template match= "/">
          <html>
            <head>
                <title>Correo electronico</title>
            </head>
            <xsl:apply-templates select = "mail"/>
          </html>
        </xsl:template>

        <xsl:template match= "mail">
          <body>
            <xsl:apply-templates select = "*"/>
          </body>
        </xsl:template>

        <xsl:template match= "from">
          <p><b>De:</b> <xsl:value-of select="."/></p>
        </xsl:template>

        <xsl:template match= "to">
          <p><b>Para:</b> <xsl:value-of select="."/></p>
        </xsl:template>

        <xsl:template match= "subject">
          <p><b>Asunto:</b> <xsl:value-of select="."/></p>
        </xsl:template>

        <xsl:template match= "message">
          <p><b>Mensaje:</b></p>
          <p><xsl:value-of select="."/></p>
        </xsl:template>
</xsl:stylesheet>
```

Al aplicar sobre el documento original mail.xml el documento XSLT mail.xslt tendríamos como resultado un documento HTML llamado mail.html con el siguiente código:

```
<html>
  <head>
        <title>Correo electronico </title>
  </head>
  <body>
        <p><b>De:</b>carlos@ceslesheures.com </p>
        <p><b>Para:</b> cmarinmu@uoc.edu </p>
        <p><b>Asunto:</b>Correo en XML</p>
        <p><b>Mensaje:</b></p>
        <p> Esto es un documento XML que representa un e-mail</p>
  </body>
</html>
```

En todo archivo XSLT tenemos que distinguir dos partes importantes:

- La sección del árbol de origen sobre la que se aplica una plantilla
- La salida que se insertará en el árbol resultado

La sección del árbol de origen con la que se esta realizando la comparación se especifica en el atributo match. Si aparece match=”/” significa que la plantilla se compara con la raíz del documento original en XML.

Todo lo que habrá dentro de la plantilla, entre las etiquetas de inicio (<xsl:template match= “subject”>) y fin (</xsl:template>) es lo que aparecerá en el árbol de salida. Todos los elementos XML o texto que encontramos dentro de la plantilla aparecerán literalmente en el árbol salida; en cambio, todos los elementos que aparecerán dentro de la plantilla con el prefijo xsl, indican al procesador XSLT que debería hacerse alguna tarea

Veamos en detalle cada uno de los elementos XSLT que podemos encontrar:

<xsl:stylesheet>

El elemento <xsl:stylesheet> es el elemento raíz de casi todas las hojas de estilo XSLT y se utilizan de la siguiente manera:

```
<xsl:stylesheet version=”1.0”
  xmlns:xsl= http://w3.org/1999/XSL/Transform>
```

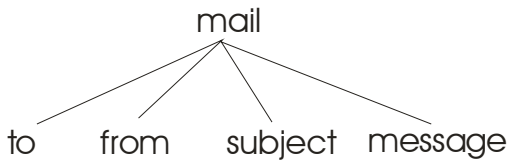
En este elemento indicamos la version de transformación que utilizamos que si seguimos las especificaciones actuales de XSLT del W3C deberíamos indicar siempre la versión 1.0.
Además se incluye en este elemento el atributo xmlns:xsl y se debe completar con el valor que aparece en el ejemplo lo que permitirá a los procesadores XSLT reconocer nuestros elementos XSLT

<xsl:template>

Tal y como se ha visto en el ejemplo inicial <xsl:template> es el elemento utilizado para definir las plantillas que formaran parte de nuestra hoja de estilo XSLT. Su sintaxis es:

```
<xsl:template match= “XPath Expresión”>
```

El atributo match se utiliza para indicar el modelo XPath que se compara con el árbol de origen (Ver apartado 6.2 de este mismo capitulo). Si recordamos que un documento XML se podía rescribir como un árbol donde los nodos eran los diferentes elementos del documento XML, es fácil observar que mediante este atributo comparamos cada uno de los nodos. Para nuestro ejemplo inicial tendríamos que el árbol que representaría el documento mail.xml sería:



<xsl:apply-templates>

El elemento <xsl:apply-templates> se usa dentro de la plantilla para llamar a otras plantillas:

```
<xsl:apply-templates select = “XPath expresión”>
```

Si se indica el atributo select entonces se evalúa la expresión XPath y el resultado se utilizará como nodo de contexto, que será utilizado por las otras plantillas. Así en nuestro ejemplo, teníamos:

```
<xsl:apply-templates select = "*"/>
```

Lo que indica que queremos recorrer todos los nodos del árbol original y sobre cada uno de estos nodos se aplicara la plantilla.

Por ejemplo en el caso que tuviéramos

```
<xsl:apply-templates select = "from"/>
```

indicaríamos que solo queremos recorrer las etiquetas from contenidas en la etiqueta actual.

<xsl:value-of>

El elemento <xsl:value-of> busca el nodo del contexto para el valor especificado en la expresión XPath del atributo select y lo inserta en el árbol resultado. Así en nuestro ejemplo

```
<xsl:template match= "from">
  <p><b>De:</b> <xsl:value-of select="."/></p>
</xsl:template>
```

inserta un PCDATA desde el nodo de contexto, en este caso "from". Por lo tanto inserta el texto que aparece dentro de la etiqueta <from> del fichero XML original.

También podríamos utilizar una sintaxis de este tipo:

```
<xsl:value-of select = "costumer/id">
```

que añadiría en el árbol de salida el texto del atributo id del elemento <costumer>

<xsl:output>

Permite definir propiedades sobre el tipo de salida que se quiere generar. Algunos ejemplos:

```
<xsl:output method="HTML"/>
```

indica que la salida será HTML

```
<xsl:output method= "xml" encoding= "ISO-8859-1"/>
```

indica que la salida será XML utilizando la codificación ISO-8859-1

<xsl:element>

Permite crear un nuevo elemento de forma dinámica en el árbol resultado. Podemos añadir el atributo name que especifica el nombre del elemento:

```
<xsl:element name= "eti">Mi nueva etiqueta</xsl:element>
```

produciría en el árbol salida el resultado:

```
<eti>Mi nueva etiqueta</eti>
```

Para añadir la creación dinámica de nuevos elementos deberíamos utilizar el atributo name de la siguiente forma:

```
<xsl:element name= “{.}”>Mi nueva etiqueta dinámica</xsl:element>
```

de manera que todo lo que se inserte entre las llaves del atributo name se evalúa como una expresión XPath. Podemos crear un elemento y le damos el nombre desde el valor del nodo de contexto; es decir, si tenemos una plantilla como esta:

```
<xsl:template match = “name”>
        <xsl:element name= “{.}”>Mi nueva etiqueta dinamica</xsl:element>
</xsl:template>
```

entonces al ejecutar:

```
<name> Andrea </name>
```

la salida será:

```
<Andrea>Mi nueva etiqueta dinámica</Andrea>
```

y si ejecutamos:

```
<name> Maite </name>
```

la salida será:

```
<Maite>Mi nueva etiqueta dinámica</Maite>
```

<xsl:attribute>

El elemento <xsl:attribute> funciona de forma similar que el elemento <xsl:element> añadiendo de forma dinámica un atributo a un elemento en el árbol resultado:

```
<name><xsl:attribute name= “id”>213</xsl:attribute>Carlos</name>
```

producirá el resultado:

```
<name id= “231”>Carlos</name>
```

<xsl:text>

El elemento <xsl:text> permite escribir texto de manera literal en el árbol de salida. Así el elemento:

```
<xsl:text>Texto insertado literalmente</xsl:text>
```

incluiría el texto “Texto insertado literalmente” en el árbol de salida

<xsl:if>

Este elemento permite definir una estructura condicional simple. El elemento <xsl:if> evalúa la expresión del atributo test y si es verdadera entonces se evalúa el contenido del elemento, en caso contrario no se evalúa.

```
<xsl:if test= “name”>Nos encontramos en un elemento name </xsl:if>
```

En este caso, si existe un elemento <name> que es hijo del nodo de contexto, entonces se insertará el texto “Nos encontramos en un elemento name” en el árbol resultado. Si por el contrario, no existe un elemento <name>, entonces no sucederá nada.

<xsl:choose>

Este elemento nos permite definir una estructura condicional múltiple. Posee mucha mas flexibilidad que el elemento <xsl:if> ya que incluso mediante el elemento <xsl:otherwise> podemos añadir un valor predeterminado. Su funcionamiento sería como sigue:

```
<xsl:choose>
        <xsl:when test= “number[. &gt; 2000]”>Un numero grande</xsl when>
        <xsl:when test= “number[. &gt; 1000]”>Un numero mediano</xsl when>
        <xsl:otherwise>Un numero pequeño</xsl otherwise>
</xsl:choose>
```

De manera que si el elemento number contiene un valor numérico mayor que 2000 aparecerá en el árbol de resultado el texto “Un numero grande”. Si el numero number es mayor que 1000 aparecerá el texto “Un numero mediano”; y en caso contrario, en el árbol de salida aparecerá el texto “Un numero pequeño”.

<xsl:for-each>

En muchos casos es necesario realizar algún procesamiento específico para varios nodos del árbol origen. El contenido de <xsl:for-each> permite la iteración, permite hacer un recorrido por todos los nodos seleccionados por la expresión XPath del atributo select.
Por ejemplo:

```
<xsl:for-each select= “name”>
        Es un nombre de elemento
</xsl:for-each>
```

Generará una instancia de esta plantilla para cada elemento <name> que sea hijo del nodo de contexto. En este caso, lo único que haría esta plantilla será producir una salida de texto para cada elemento

<xsl:copy-of>

El elemento <xsl:copy-of> nos permite tomar secciones del árbol de origen y copiarlas al árbol resultado; esto es mucho mas sencillo que crear manualmente cada uno de los elementos/atributos y copiar mas tarde los valores usando el elemento <xsl:value-of>.

```
<xsl:copy-of select= “.”/>
```

Copiaría todos los elementos, incluidos atributos y elemento hijos a partir del nodo de contexto en el que nos encontremos.

<xsl:copy>

El elemento <xsl:copy> funciona de manera similar al elemento <xsl:copy-of> con la diferencia que ni los atributos ni los elementos hijos del nodo de contexto no se copian automáticamente al árbol resultado.

<xsl:sort>

El elemento <xsl:sort> nos permite ordenar los nodos según los valores de un elemento o atributo
Por ejemplo:

```
<xsl:for-each select =”*”>
        <xsl:sort select = “@name”/>
        <xsl-copy-of “.”/>
</xsl:for-each>
```

En este ejemplo para todos los nodos, se ordenaran por el valor del atributo name y se copiarán (ordenados) en el árbol de salida.

<xsl:variable>

El elemento <xsl:variable> permite definir constantes. El funcionamiento es muy simple:

```
<xsl:variable name = “nombre”>Carlos</xsl:variable>
<xsl:value-of select = “$nom”/>
```

escribiría en el documento de salida el valor de la constante nombre, en este caso, Carlos

6.2 XPath

Todo el procesamiento realizado con un fichero XML está basado en la posibilidad de direccionar o acceder a cada una de las partes que lo componen, de modo que podamos tratar cada uno de los elementos de forma diferenciada.

El tratamiento del fichero XML comienza por la localización del mismo a lo largo del conjunto de documentos existentes en el mundo. Para llevar a cabo esta localización de forma unívoca, se utilizan los **URI** (*UnifomResource Identifiers*), de los cuales los **URL** (*Unifom Resource Locators*) son sin duda los más conocidos.
Una vez localizado el documento XML, la forma de seleccionar información dentro de él es mediante el uso de **XPath**, que es la abreviación de lo que se conoce como *XML Path Language*. Con XPath podremos seleccionar y hacer referencia a texto, elementos, atributos y cualquier otra información contenida dentro de un fichero XML.

XPath en sí es un lenguaje sofisticado y complejo, pero distinto de los lenguajes procedimentales que solemos usar (C, C++, Basic, Java...). Además, como casi todo en el mundo de XML, aún está en estado de desarrollo, por lo que no es fácil encontrar herramientas que incorporen todas sus funcionalidades.

XPath es a su vez la base sobre la que se han especificado nuevas herramientas que aprovechan el tratamiento de documentos XML. Herramientas tales como **XPointer**, **XLink** y **XQL** (el lenguaje que maneja los documentos XML como si de una base de datos se tratase), que también están en estado de desarrollo, pero que sin duda cambiarán el modo en que actualmente concebimos la navegación por la Web. Así, XPath sirve para decir cómo debe procesar una hoja de estilo el contenido de una página XML, pero también para poder poner enlaces o cargar en un navegador zonas determinadas de una página XML, en vez de toda la página.

La gran importancia que tiene el XPath como estándar de transformación de XML se debe a que cualquier transformación que queramos realizar a partir de un documento XML, incluso si utilizamos el estándar de transformación XSLT, la transformación del documento origen se hace a partir de un recorrido por el árbol del documento analizando cada uno de los nodos de este árbol.
El Xpath es un estándar mucho más enfocado a la búsqueda de elementos dentro de un documento XML.

Xpath es la especificación responsable de la sintaxis de acceso y consulta a los elementos de un documento XML, se trata de un lenguaje de especificación de expresiones que permite definir la manera de localizar un elemento dentro de un documento XML. Quizás la característica más interesante del XPath es que no es un lenguaje basado en XML, como podría serlo el XSLT.

XPath es pues, un lenguaje de especificación de expresiones no basado en XML que permite la búsqueda y consulta de la información almacenada dentro de un estructura XML.

Para direccionarnos dentro del documento XML se utilizan expresiones XPath por medio de una ruta de acceso de ubicación, que nos conduce, paso a paso, a la parte del documento a la que estamos accediendo. Obviamente necesitamos lo que se conoce como nodo de contexto que indica la sección del documento XML desde donde se empieza el trayecto.

6.2.1 El modelo de datos del XPath

Un documento XML es procesado por un analizador (o *parser*) construyendo un **árbol de nodos**. Este árbol comienza con un elemento raíz, que se diversifica a lo largo de los elementos que cuelgan de él y acaba en nodos hoja, que contienen solo texto, comentarios, instrucciones de proceso o incluso que están vacíos y solo tienen atributos

La forma en que XPath selecciona partes del documento XML se basa precisamente en la representación arbórea que se genera del documento. De hecho, los "operadores" de que consta este lenguaje nos recordarán la terminología que se utiliza a la hora de hablar de árboles en informática: raíz, hijo, ancestro, descendiente, etc...

Un caso especial de nodo son los nodos **atributo**. Un nodo puede tener tantos atributos como desee, y para cada uno se le creará un nodo atributo. No obstante, dichos nodos atributo NO se consideran como hijos suyos, sino más bien como etiquetas añadidas al nodo elemento.

A continuación se muestra un ejemplo de cómo se convierte en árbol un documento XML.

```
<libro>
  <titulo>Dos por tres calles</titulo>
  <autor>Josefa Santos</autor>
  <capitulo num="1">
    La primera calle

   <parrafo>
        Era una sombría noche del mes de agosto...
   </parrafo>

    <parrafo destacar="si">
        Ella, inocente cual
        <enlace href="http://www.enlace.es">mariposa</enlace>
        que surca el cielo en busca de libaciones...
   </parrafo>

  </capitulo>

  <capitulo num="2" public="si">
    La segunda calle

   <parrafo>Era una obscura noche del mes de septiembre...</parrafo>

   <parrafo>
        Ella, inocente cual
        <enlace href="http://www.abejilla.es">abejilla</enlace>
        que surca el viento en busca del néctar de las flores...
   </parrafo>

   </capitulo>

  <apendice num="a" public="si">
  La tercera calle

  <parrafo>
        Era una densa noche del mes de diciembre...
  </parrafo>

  <parrafo>
        Ella, cándida cual
        <enlace href="http://www.pajarillo.es">abejilla</enlace>
        que surca el espacio en busca de bichejos para comer...
  </parrafo>

</apendice>

</libro>
```

El árbol generado sería:

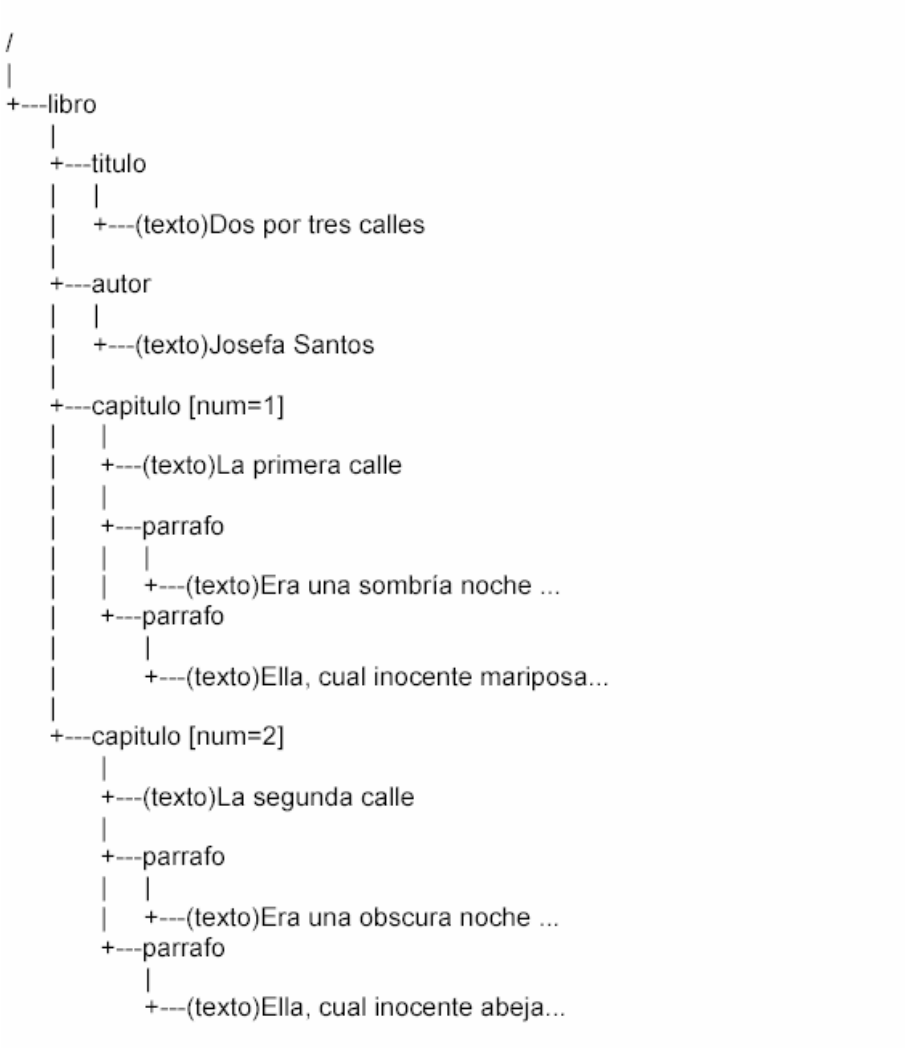


Figura 1.11. Representación del arbol de documento XML según XPath

6.2.2 Tipos de nodos

Existen distintos tipos de nodos en un árbol generado a partir de un documento XML, a saber: **raíz**, **elemento**, **atributo**, **texto**, **comentario** e **instrucción de procesamiento** (respectivamente; *root, elements, attribute, text,comment* y *processing instruction*).

Nodo Raíz
Se identifica por /. No se debe confundir el nodo raíz con el elemento raíz del documento. Así, si el documento XML de nuestro ejemplo tiene por elemento raíz a libro, éste será el primer nodo que cuelgue del nodo raíz del árbol, el cual es: /.

Nodo Elemento
Cualquier elemento de un documento XML se convierte en un nodo elemento dentro del árbol. Cada elemento tiene su nodo padre. El nodo padre de cualquier elemento es, a su vez, un elemento, excepto el elemento raíz, cuyo padre es el nodo raíz. Los nodos elemento tienen a su vez hijos, que son: nodos elemento, nodos texto, nodos comentario y nodos de instrucciones de proceso. Los nodos elemento también tienen propiedades tales como su nombre, sus atributos e información sobre los "espacios de nombre" que tiene activos.
Cualquier elemento de un documento XML se convierte en un nodo elemento dentro del árbol. Cada elemento tiene su nodo padre. El nodo padre de cualquier elemento es, a su vez, un

elemento, excepto el elemento raíz, cuyo padre es el nodo raíz. Los nodos elemento tienen a su vez hijos, que son: nodos elemento, nodos texto, nodos comentario y nodos de instrucciones de proceso. Los nodos elemento también tienen propiedades tales como su nombre, sus atributos e información sobre los "espacios de nombre" que tiene activos.
Una propiedad interesante de los nodos elemento es que pueden tener identificadores únicos (para ello deben ir acompañados de un DTD que especifique dicho atributo toma valores únicos), esto permite referenciar a dichos elementos de una forma mucho más directa.

Nodos texto
Por texto vamos a hacer referencia a todos los caracteres del documento que no está marcados con alguna etiqueta. Un nodo texto no tiene hijos, es decir, los distintos caracteres que lo forman no se consideran hijos suyos.

Nodos atributo
Como ya hemos indicado, los nodo atributo no son tanto hijos del nodo elemento que los contiene como etiquetas añadidas a dicho nodo elemento. Cada nodo atributo consta de un nombre, un valor (que es siempre una cadena) y un posible "espacio de nombres".
Aquellos atributos que tienen por valor el valor por defecto asignado en el DTD se tratarán como si el valor se le hubiese al escribir el documento XML. Al contrario, no se crea nodo para atributos no especificados en el documento XML, y con la propiedad #IMPLIED definida en su DTD. Tampoco se crean nodos atributo para las definiciones de los espacios de nombre. Todo esto es normal si tenemos en cuenta que no es necesario tener un DTD para procesar un documento XML.

Nodos comentario y de instrucciones de proceso
Aparte de los nodos indicados, en el árbol también se generan nodos para cada nodo con comentarios y con instrucciones de proceso. Al contenido de estos nodos se puede acceder con la propiedad string-value.

6.2.3 Sintaxis del lenguaje de transformación XPath

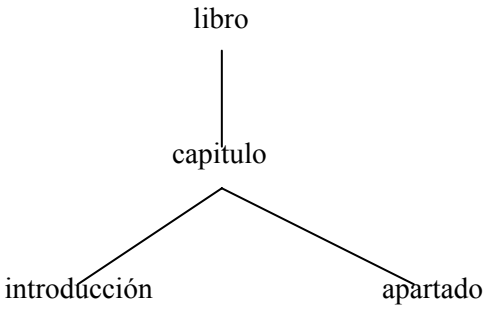
Expresiones XPath

Ya se ha comentado anteriormente que XPath es un lenguaje de expresiones que permite realizar búsquedas y consultas sobre la información almacenada dentro de una estructura XML. Es, de todas formas, importante recordar que XPath es un lenguaje no basado en XML.

Para el acceso a uno o diversos nodos del árbol XML hay dos formas de realizarlo:

- Indicando las localizaciones absolutas desde la raíz del árbol. La manera de expresar estas localizaciones es similar a la forma en que se referencian directorios en el sistema Operativo Unix viendo una similitud entre el directorio raíz y la raíz del árbol XML y los diferentes directorios que cuelgan de este y los diversos nodos que cuelgan de la raíz en el documento.

Supongamos que tenemos el siguiente árbol generado a partir de un documento XML:

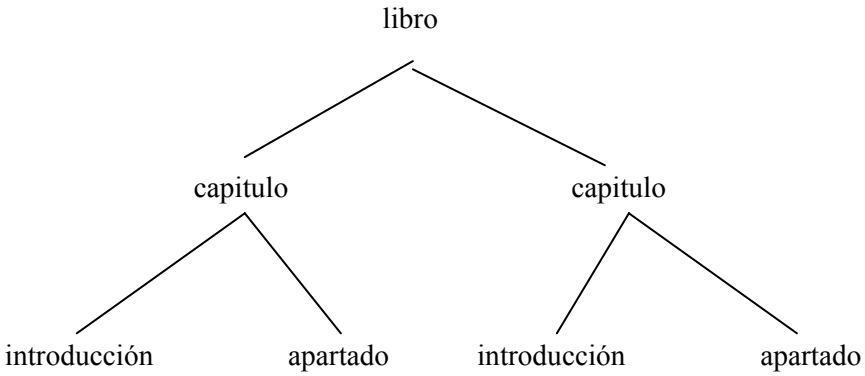


Tendremos las siguientes expresiones XPath:

/libro/capitulo/apartado	Accedería al elemento apartado del capitulo del libro
/libro/capitulo/*	Accedería a todos los elementos que se derivan de capitulo, en este caso introducción y apartado

Una diferencia que encontramos al realizar esta similitud entre estructura de árbol XML y estructura de directorios es que en un árbol XML un nodo puede tener nodos hermanos (situados en el mismo nivel de profundidad) con el mismo nombre, mientras que en una estructura de directorios un directorio no puede contener dos directorios que se llamen de igual forma.

Supongamos que la estructura de árbol que tenemos ahora es:



tendríamos la posibilidad que utilizar las expresiones:

/libro/capitulo/apartado	Accedería a todos los elementos apartado de todos los elementos capitulo del elemento libro
/libro/capitulo/*	Accedería a todos los elementos que cuelgan de capitulo (de todos los que hayan), que en este caso son introducción y apartado

- Indicando localizaciones relativas a una posición determinada, a partir de la que es posible desplazarse en diferentes direcciones según un conjunto de modificadores de dirección. Esta forma de acceso a nodos se conoce con el nombre de ejes (axis)

El siguiente esquema es un mapa de áreas en que se hace referencia a algunos de estos modificadores o ejes

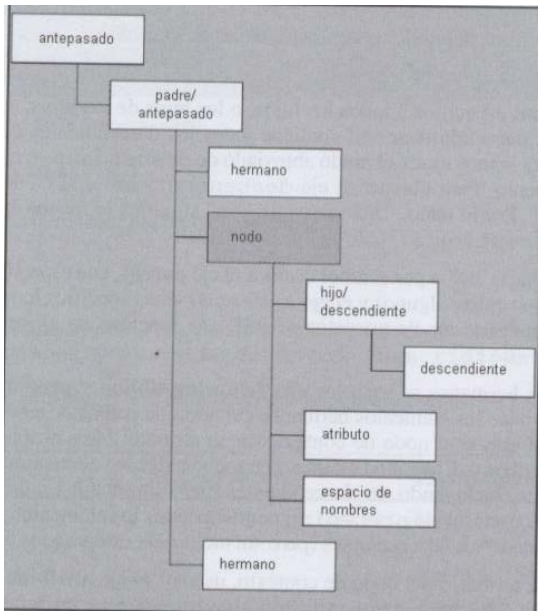


Figura 1.12. Representación de los ejes (axis) XPath

La siguiente tabla lista varias expresiones XPath usando ejes:

Axis de XPath 2.0	Descripción
child::order order order/* order/@*	Selecciona todos los elementos <order> que sean hijos del nodo actual o de contexto Selecciona todos los hijos de <order> Selecciona todos los atributos que pertenecen a <order>
self::description-desdendant-or- .//description	Selecciona todos los elementos <description> que son descendents del nodo actual, incluyendo el nodo actual
number/order/attribute:: order/@number	Selecciona el atributo number del nodo <order>
//date/parent::order //date/../order	Selecciona todos los elementos <order> que sean padres del elemento <date>
::description/order/descendant	Selecciona todos los elementos <description> que sean descendientes del elemento <order>
ancestor::order//quantity/	Selecciona todos los elementos <order> que sea antepasado de un elemento <quantity>
//description/following-sibling:quantity	Selecciona un elemento <quantity>, si ese elemento es hermano de un nodo <description> y que provenga después de un nodo <description> en el documento

Axis de XPath 2.0	Descripción
//quantity/preceding-sibling:description	Selecciona un elemento <description> si ese es hermano del nodo <quantity> y se encuentra antes que el nodo <quantity>
/order/date/following::*	Selecciona todos los elementos que aparezcan en el documento después del elemento <date> no incluyendo los descendientes
/order/item/preceding::*	Selecciona todos los elementos que aparezcan antes del elemento <item>, no incluyendo los antepasados

En esta tabla se ha incluido la sintaxis abreviada de algunas de las expresiones XPath utilizando la abreviatura de los modificadores:

Sintaxis Abreviada	Expresión XPath	Descripción
//	descendant	//order es equivalente a desscendant::order
.	self	./order es equivalente a self::*/order
..	parent	../order es equivalente a parent::*/order
@	attribute	@number es equivalente a number/order/attribute::
	child	order es equivalente a child::order

Una vez indicada la localización de un nodo, puede darse el caso que una expresión XPath pueda contener una o diversas condiciones que permitan refinar la búsqueda que se está realizando. Estas condiciones se expresan mediante predicados que se evalúan para cada nodo localizado. Si el resultado de la evaluación es falso, entonces el nodo será descartado.

Los predicados son expresiones que van entre los signos []

Veamos algunos ejemplos:

Ejemplos de predicados en Xpath	Descripción
capitulo[1]	Selecciona el primer nodo capitulo hijo del nodo actual
capitulo[position()=1]	Ídem que el anterior
capitulo[last()]	Selecciona el ultimo nodo capitulo hijo del nodo actual
capitulo[@titulo=”Mi casa”]	Selecciona el nodo capitulo que tiene como valor del atributo titulo “Mi casa”

Funciones XPath

Funciones de nodo

Las funciones de nodo se usan para trabajar con nodos en general. Estas funciones devuelven información acerca del nodo.

Función	Ejemplo	Descripción
name()	<xsl:template match="name"> <xsl:for-each select="*"> <p><xsl:value-of select= "name()"/> </xsl:for-each> </xsl:template>	Devuelve el nombre del nodo. En el ejemplo devolverá el nombre de cada uno de los nodos hijos del nodo <name>
node()		Devuelve el propio nodo
processing-instruction()	<xsl:template match="processing-instruction()"> <xsl:value-of select= "."/> </xsl:template>	Devuelve instrucciones de procesamiento En el ejemplo se devuelven todas las instrucciones de procesamiento del documento
comment()	<xsl:template match="comment()"> <xsl:value-of select= "."/> </xsl:template>	Devuelve comentarios. En el ejemplo mostraría los comentarios del documento
text()	<xsl:template match="padre"> <xsl:value-of select= "text()"/> </xsl:template>	Obtiene el contenido de un elemento. En el ejemplo obtenemos el texto PCDATA del elemento <padre>

Funciones posicionales

Para los nodos que perteneces a un conjunto de nodos las funciones posicionales permiten buscar coincidencias en la posición de estos nodos dentro del conjunto.

Función	Ejemplo	Descripción
position()	<xsl:template match= "/libro/capitulo[position() = 2]">	Devuelve la posición del nodo dentro de un conjunto de nodos. En el ejemplo buscamos la información del nodo hijo de capitulo que ocupa la segunda posición
last()	<xsl:template match= "/libro/capitulo[position() = last()]"/>	Devuelve el nodo que ocupa la ultima posición en un conjunto de nodos
count()	<xsl:template match= "count(nodo)"/>	Devuelve el numero de nodos dentro del conjunto de nodos

Funciones numéricas

Funciones para el tratamiento de números.

Función	Ejemplo	Descripción
number()		Convierte texto PCDATA en un valor numérico
sum()	><xsl:value-of select= “sum(/libro/capitulo/paginas)”/>	Se usa para sumar todos los valores numéricos de un conjunto de nodos

Funciones Booleanas

Funciones utilizadas para aplicar la lógica booleana

Función	Ejemplo	Descripción
bolean()	boolean(nombre)	evalúa una expresión XPath para saber si es cierta o falsa En el ejemplo se comprueba si existe un elemento <nombre>
not()	not(nombre)	Operación unaria de negación
true() y false()		Funciones que devuelven cierto y falso respectivamente

Funciones de cadena

Funciones en XPath para el tratamiento y manipulación de cadenas de caracteres

Función	Ejemplo	Descripción
string()		Convierte cualquier valor en un cadena de caracteres
string-length()		Permite obtener la longitud de una cadena de caracteres
concat()		Función que permite concatenar múltiples cadenas
contains()		Indica si una cadena contiene dentro de sí misma una cadena determinada
starts-with()		Indica si una cadena tiene como inicio el contenido de una segunda cadena
substring() substring-after() substring-before()		Permite obtener una subcadena a partir de una cadena más larga Susbtring-after y substring-before() devuelve la cadena a partir del carácter indicado hasta el final o el principio respectivamente
translate()	translate(“hola mundo”,”hm”, “HM)	A partir de una cadena a traducir, una cadena para buscar y una cadena de

		reemplazo se traduce la cadena original En el ejemplo se sustituiría las letras ‘h’ y ‘m’ minúsculas por sus respectivas mayúsculas

Los lenguajes de consulta XML

1 Características de los lenguajes de consulta XML

Tradicionalmente las búsquedas sobre texto se han venido aplicando a documentos como un todo, mientras que las consultas de datos se han aplicado a registros compuestos por varios campos. Dado que XML presenta datos semiestructurados es necesario que el lenguaje de consultas aplicado sobre XML pueda acceder a datos de la parte estructurada del documento, pero que además pueda realizar búsquedas sobre el texto y permita realizar consultas mixtas de contenido y estructura.
En cuanto a los resultados, tradicionalmente las consultas sobre texto devuelven una lista de documentos con cierta información acerca de ellos ordenados según un determinado criterio, mientras que una consulta sobre datos devuelve generalmente un determinado campo, registro o conjunto de registros que pueden verse afectados por un proceso de cálculo, transformación o combinación.

En definitiva, se observa que en las consultas de datos se pone especial énfasis en los grandes almacenes de datos, la integración de datos heterogéneos y la transformación de datos en formatos comunes de intercambio; en cambio, las consultas de texto ponen el énfasis en las búsquedas de texto, la manipulación de conjuntos de resultados, las relaciones de inclusión y la ordenación de los documentos obtenidos como resultado.

Así, pues el lenguaje de consultas sobre XML, debería reunir lo mejor de los dos tipos de consultas comentadas; debería ser la unión de las ventajas obtenidas por un lenguaje de consultas sobre documentos y por un lenguaje de consultas de datos.

1.2 Características deseables en un lenguaje de consultas XML

1.2.1 Desde la perspectiva de datos semiestructurados

Un lenguaje de consultas para XML debería poseer las características comunes de los lenguajes de consultas de datos semi-estructurados, tanto las puramente relacionales (operaciones de selección, filtrado, reducción y combinación) como características similares a la de los lenguajes de consulta de bases de datos orientadas a objeto

A continuación se explican en mayor detalle las principales características deseables para la consulta de datos:

- Operación de selección: seleccionar un documento o elemento basándose en el contenido, estructura o atributos que satisfagan una condición específica. Las consultas de selección constan generalmente de 3 partes:

- cláusula *patrón*: equipara elementos anidados en el documento de entrada y les asocia variables. Equivale a la cláusula ***FROM*** de SQL, donde los patrones son tablas, vistas o alias. En el caso de XML, un patrón es un elemento XML en el cual algunos de los ítems (nombres de etiqueta, contenido, nombres de atributo o valores de atributo) son eventualmente sustituidos por variables.

- cláusula *filtro*: comprueba que las variables asociadas cumplan las condiciones establecidas. Equivale a la cláusula ***WHERE*** de SQL. El significado de los patrones y filtros en los lenguajes para datos semiestructurados es una relación (producto cartesiano de todas las variables asignadas que satisfacen los patrones y filtros), la cual tiene una estructura plana y sin orden.

- cláusula *constructor*: específica el resultado en términos de las variables asociadas, es decir qué formato ha de tener la respuesta. Equivale a la cláusula ***SELECT*** de SQL. Las construcciones deberían poder incluir consultas anidadas, así como poder crear nuevos elementos.

- Operación de filtrado: extraer determinados elementos de los documentos conservando la jerarquía y secuencia.

- Operación de reducción: proyectar como salida la poda de los elementos especificados en la selección que satisfacen las condiciones, en vez de devolver un subárbol con todos los elementos y atributos.

- Operaciones de reestructuración, como por ejemplo la agrupación de datos relacionados (mediante funciones *Skolem1* o mediante operadores de agrupación), y la ordenación.

- Operación de combinación de datos de diferentes porciones de documentos (correspondiente al *join* relacional) o combinación de diferentes partes del mismo documento (*semi-join*).

- Uso de variables etiqueta o expresiones de camino para permitir consultas sin conocimiento preciso de la estructura del documento y acceso a datos anidados de forma arbitraria.

- El lenguaje de consulta debe, además, poderse usar aún cuando no se conozca un esquema (**DTD** o **XML *Schema***) a *priori*.

- **Uso de operadores de navegación** que simplifiquen el manejo de datos con referencias (atributos **ID**, **IDREF**(S)).

- **Uso de funciones de agregación**.

- **Uso de la cuantificación** existencial y universal.

- **Manejo de tipos de datos**, en particular los del XML *Schema*

- Operaciones de inserción, borrado y modificación.

1.2.2 Desde la perspectiva búsqueda de información

Los documentos XML pueden considerarse como una colección de documentos de texto con etiquetas adicionales que guardan cierta relación. Las técnicas de recuperación de información tradicionales han de adaptarse a la búsqueda por estructura y contenido de tales documentos y hay que plantearse cómo ordenar por relevancia los resultados teniendo en cuenta que en una consulta XML las unidades de información devueltas pueden ser tanto documentos XML enteros, un fragmento de documento XML (por ejemplo un elemento y su contenido), así como una combinación de documentos XML o de fragmentos.

Analizaremos los diferentes tipos de consultas formuladas normalmente a los sistemas de recuperación de información clásicos:

- Consultas basadas en palabras-clave. Se refiere a consultas de contenido, tanto para buscar datos sin estructura, como para buscar datos estructurados pero cuya estructura desconoce el usuario, o consultar múltiples documentos XML con la misma ontología pero diferentes DTDs.

- Consultas de una sola palabra. La búsqueda de texto puede ser tan simple como determinar si una palabra concreta está o no presente en el texto. En este caso el resultado serían los documentos o elementos que contienen tal palabra, ordenados arbitrariamente. Los lenguajes de consulta para XML generalmente suavizan el problema del desconocimiento de la estructura exacta mediante la utilización de expresiones regulares de camino, si bien en general requieren un conocimiento parcial de la misma del que un usuario puede carecer totalmente.

- Consultas booleanas: Constan de una combinación de varias palabras ligadas con operadores booleanos. Los documentos o elementos recuperados dependerán de los operadores utilizados.

- Consultas de contexto. Implican distancias textuales, restricciones que se pueden imponer respecto a las posiciones de los términos, como que formen una secuencia (frase) o que estén próximos. La proximidad textual también es uno de los factores a considerar para efectuar la ordenación de resultados por relevancia.

- Consulta difusa. La consulta consta de una o más palabras y el resultado de la búsqueda es un conjunto de documentos o elementos que contienen al menos una de ellas. Los documentos elementos recuperados serán más o menos relevantes a la consulta y se ordenarán por similitud (*ranking*), aplicando una función generalmente ligada a la frecuencia de un término (nº de veces que una palabra aparece en un documento o elemento y a la frecuencia inversa de documento o elemento (nº de documentos o elementos en los cuales aparece el término).

- Consultas de concordancia de patrones, para recuperar fragmentos de texto que concuerden con el patrón especificado (cadenas, prefijos, sufijos, subcadenas, o en general expresiones regulares).

- Consultas estructurales. Básicamente podemos resaltar:

- Relaciones de inclusión: se refieren a la selección de elementos que incluyan o estén incluidos en otros (por ejemplo seleccionar capítulos que incluyan una figura o seleccionar figuras incluidas en una sección). Este tipo de relación abarca también la selección de ancestros o descendientes directos de un elemento. También consideraremos la inclusión posicional: seleccionar el i-ésimo elemento (o un rango de elementos) de los incluidos por un elemento de otro conjunto (por ejemplo seleccionar el tercer párrafo de los capítulos que contengan una figura).

- Relaciones de distancia (medida en función del nº de arcos que hay que atravesar desde un nodo a otro) para poder expresar un orden de precedencia o una proximidad estructural, la cual se podría tener en cuenta para calcular la relevancia. Los tipos de consulta concretos que se puedan formular dependerán en parte del modelo de recuperación que el sistema adopte para un análisis de técnicas de indización para documentos XML y modelos de recuperación, y [10] para un estudio de modelos de datos para manejar documentos estructurados a nivel de usuario, conceptual y físico). Además de permitir algunos de los tipos de consulta mencionados anteriormente, un lenguaje para XML también sería deseable que posibilitara:

- La asignación de pesos a los términos de la consulta.

- La utilización de metadatos (**RDF**).

- El soporte de *XLink* y *XPointer*. Dado que generalmente las consultas están orientadas a la Web, una consulta debe poder atravesar las referencias internas y externas para proporcionar documentos relacionados.

- La manipulación de conjuntos: unión, diferencia e intersección de resultados.

Así, pues vemos que un buen lenguaje de consultas XML deberá ser el resultado de fusionar lo que se espera de un lenguaje de consultas sobre texto y las características de los lenguajes de consultas sobre datos.

2 Evolución histórica

En febrero de 2001, el W3C publicó el primer borrador de XQuery, que se pretendía se constituyese el lenguaje de consulta estándar para XML.
Pero hasta llegar a este proceso de estandarización han ido apareciendo otros lenguajes de consulta y se ha ido trabajando siguiendo las directrices del W3C, así en diciembre de 1998 se celebra la workshop QL'98 del W3C sobre lenguajes de consulta. El XML Query WG se establece en agosto de 1999. En enero de 2000 se publica la primera versión de los requisitos de XML Query, apareciendo en 2001 el primer borrador de XQuery; por último la última revisión se ha producido el 2 de mayo de 2003

Algunos de los lenguajes que fueron apareciendo durante este proceso fueron: **Lorel** (representativo de los lenguajes para datos semiestructurados), XML-QL (primer lenguaje que utiliza sintaxis XML), XSLT (lenguaje de transformación de árboles XML) y **XQL** 99 (lenguaje que aporta la navegación por documentos jerárquicos).

La siguiente figura muestra las relaciones entre distintos lenguajes, y como a partir de los diferentes lenguajes se derivó a la aparición del XQuery

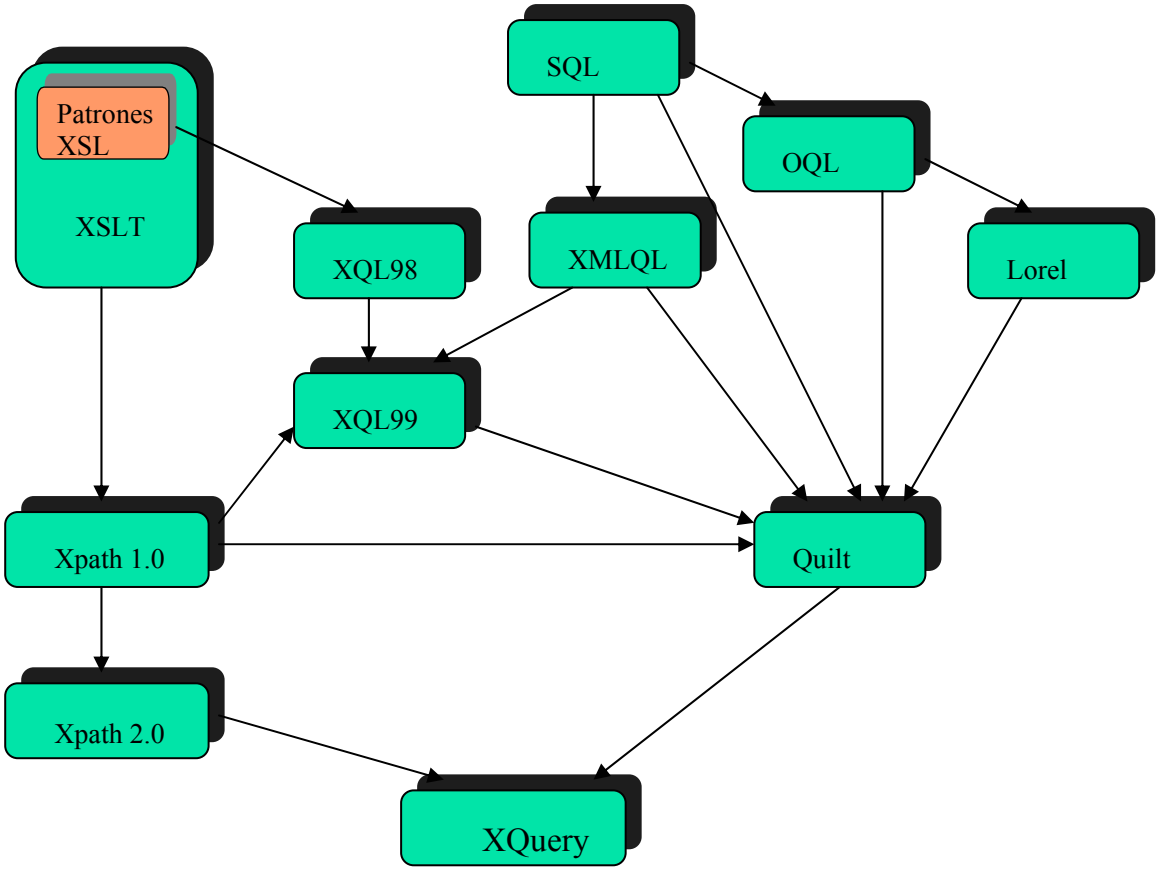


Figura 2.1. Evolución histórica de los lenguajes de consulta XQuery Languages

3 Comparativa entre los XQuery Languages

3.1 Lorel

Lorel es una extensión del lenguaje **OQL**, y está ligado al gestor de bases de datos semiestructurados Lore.
Utiliza un modelo de datos propio representado en forma de grafo donde cada elemento es un par <identificador, valor>, aunque también permite conmutar a una estructura de árbol, en la cual los atributos **IDREF**(S) son cadenas de texto.
En las consultas de selección, los patrones aparecen en la cláusula *from* y en la *where*.

Lorel proporciona potentes expresiones de camino que consisten en una secuencia de etiquetas (separadas por un punto) que pueden incluir comodines y operadores de expresiones regulares:

"*" (0 o más ocurrencias),
"|" (disyunción),
"+" (una o más ocurrencias) y
"?" (0 ó 1 ocurrencia).

También emplea calificadores para distinguir si navegar sólo por subelementos ">" o atributos "@", y también posee índices posicionales.

Los filtros aparecen en la cláusula *where* y admite los operadores booleanos *not*, *and* y or, así como los operadores relacionales "<", "<=", "=", "<>", ">=", ">".

Lorel posee operadores para búsqueda de patrones tales como *grep* (para expresiones regulares de cadena) y *soundex* (para emparejamientos fonéticos). Lorel soporta uniones entre **DTDs**. El constructor aparece en la cláusula *select*. Lorel permite la creación de nuevos elementos mediante la función *xml (tipo, etiqueta, valor)*.

El resultado de una consulta es una lista ordenada de identificadores de los elementos que concuerdan con el criterio de búsqueda, los cuales pueden agruparse mediante la cláusula *group by*. Admite las funciones de grupo *min*, *max*, *count*, *sum* y *avg*.

Veamos un ejemplo de consulta Lorel sobre el siguiente XML llamado bib.xml

```
<bib>
  <book year="1994">
    <title>TCP/IP Illustrated</title>
    <author><last>Stevens</last><first>W.</first></author>
    <publisher>Addison-Wesley</publisher>
    <price>65.95</price>
  </book>
<book year="1999">
  <title>The Economics of Technology and Content for Digital TV</title>
  <editor>
    <last>Gerbarg</last><first>Darcy</first>
    <affiliation>CITI</affiliation>
  </editor>
  <publisher>Kluwer Academic Publishers</publisher>
  <price>129.95</price>
  </book>
<book year="2000">
    <title>Data on the Web</title>
    <author><last>Abiteboul</last><first>Serge</first></author>
    <author><last>Buneman</last><first>Peter</first></author>
    <author><last>Suciu</last><first>Dan</first></author>
    <publisher>Morgan Kaufmann Publishers</publisher>
    <price>39.95</price>
```

```
</book>
 <book year="1992">
   <title>Advanced Programming in the Unix environment</title>
   <author><last>Stevens</last><first>W.</first></author>
   <publisher>Addison-Wesley</publisher>
   <price>65.95</price>
  </book>
</bib>
```

Este documento esta basado en el siguiente archivo DTD, llamado bib.dtd

```
<!ELEMENT bib (book* )>
<!ELEMENT book (title, (author+ | editor+ ), publisher, price )>
<!ATTLIST book year CDATA #REQUIRED >
<!ELEMENT author (last, first )>
<!ELEMENT editor (last, first, affiliation )>
<!ELEMENT title (#PCDATA )>
<!ELEMENT last (#PCDATA )>
<!ELEMENT first (#PCDATA )>
<!ELEMENT affiliation (#PCDATA )>
<!ELEMENT publisher (#PCDATA )>
<!ELEMENT price (#PCDATA )>
```

La consulta a realizar es seleccionar todos los libros publicados por la editorial Addison-Wesley después del año 91, nos interesa solamente el año y el título

```
Select xml(bib:{
          (select xml{book:{@year:y, title:t})
           from bib.book b, b.title t, b.year y
           where b.publisher = “Addison-Wesley” and year>1991)})
```

3.2 XSLT

XSLT es un lenguaje creado para transformar documentos XML en otros documentos XML. Forma parte de XSL, *Extensible Stylesheet Language* que es un lenguaje de hojas de estilo para XML.
En el proceso de presentación del documento formateado hay dos aspectos:

a) construir el árbol resultado a partir de la transformación del árbol XML fuente, lo cual se consigue mediante XSLT *(XSL Transformations),* y
b) formatear el árbol resultado para su presentación en un determinado medio. XSL utiliza el modelo de datos de **XPath** según el cual un documento XML se modela como un árbol con siete tipos de nodos: raíz, elemento, texto, atributo, espacio de nombre, instrucción de procesamiento, y comentario.

XSL se puede utilizar no solo para consultar documentos individuales sino también para consultar colecciones de documentos, si se constituye un único árbol fuente concatenando los diversos documentos de forma lógica en algún orden; en tal caso el nodo raíz contendría todo el repositorio y cada hijo se correspondería con un documento XML bien-formado.
La función *document* permitiría recuperar un documento individual por su nombre. También se puede obtener una colección de documentos como salida mediante el elemento *xsl:document*.

Las consultas se pueden incrustar como cadenas en lenguajes de programación o en XML o atributos **HTML**. Este enfoque permite que los resultados puedan devolverse como texto XML, punteros a nodos, estructuras que representen regiones del documento, o como estructuras primitivas tales como cadenas, enteros, booleanos, y tablas.

XSL usa un lenguaje de patrones que permite identificar nodos en un documento XML. XSL define dos tipos de patrones:

- patrones de concordancia *(match patterns)* y
- patrones de selección *(select patterns)*.

Los primeros comprueban si un determinado nodo concuerda con el patrón, el resultado de su aplicación es un valor lógico, y se utilizan para construcción; mientras que los segundos recuperan el conjunto de nodos que cumplen el criterio que el patrón especifica y se utilizan para recuperación de información. Las consultas en XSLT se indican mediante reglas de transformación.
Cada regla se compone de un patrón y una acción o plantilla (*template*). El elemento *xsl:apply-templates* se usa para elegir el orden explícito de aplicación de plantillas que se definen en una hoja de estilos. Las plantillas se definen utilizando el elemento *xsl:template* cuyo atributo *match* especifica una porción del árbol XML de un documento. La plantilla contiene los patrones que permitirán seleccionar partes de un documento XML para su transformación. Para indicar los patrones se utiliza el elemento *xsl:for-each* que realiza un bucle para recorrer los elementos de un documento; su atributo select indica qué elementos se seleccionarán como parte del recorrido del bucle.

Se pueden usar variables, a las cuales asignarles una expresión, utilizando el elemento xsl:variable cuyo atributo name indica el nombre de la variable y el atributo select contiene la expresión que toma como valor.
XSLT utiliza expresiones XPath si bien el resultado es un conjunto de nodos sin orden. XSLT admite la ordenación de un conjunto de nodos utilizando el elemento *xsl:sort*. Los filtros se indican utilizando el elemento *xsl:if* cuyo atributo test contiene las condiciones a cotejar. En vez de *xsl:if* también se podría utilizar un predicado *XPath*. Admite operadores relacionales y los operadores lógicos *and* y *or*, así como la función *not*(). La cláusula de construcción se especifica mediante el elemento *xsl:copy-of*.

XSL permite la creación de nuevos elementos indicando los nombres de las etiquetas en la plantilla. El elemento *xsl:value-of* se usa para insertar el valor de un elemento o atributo en la salida resultante de una hoja de estilos. Permite insertar datos XML dentro del marcado **XHTML**. Se pueden expresar consultas anidadas mediante la anidación de plantillas. XSLT admite expresiones condicionales mediante los elementos *xsl:choose*, *xsl:when* y *xsl:otherwise*.
XSLT posee algunas funciones de manipulación de cadenas como: *concat* (concatena dos cadenas), *contains* (determina si una subcadena está contenida en una cadena), *starts-with*, *substringbefore*, *substring-after* y *normalize-space*.

Como carencias del lenguaje cabe indicar la imposibilidad de realizar combinaciones, así como la imposibilidad de integrar datos de múltiples fuentes XML, si bien esto último se podría solucionar añadiendo un atributo *source a xsl:for-each* y *xsl:apply-templates*.

Veamos como sería utilizando el lenguaje XSLT la misma consulta vista anteriormente sobre el documento XML bib: *"Seleccionar todos los libros publicados por la editorial Addison-Wesley después del año 91, nos interesa solamente el año y el título"*

```
<xsl:transform version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <xsl:for-each select="document(bib.xml)//book">
      <xsl:if test="publisher='Addison-Wesley' and @year>'1991'">
        <xsl:copy-of select="title"/>
      </xsl:if>
    </xsl:for-each>
  </xsl:template>
</xsl:transform>
```

3.3 XML-QL

XML-QL es un lenguaje declarativo que permite consultar, construir, transformar e integrar datos XML. Este lenguaje surge como una posible respuesta a cuestiones tales como:
qué técnicas y herramientas deberían existir para extraer datos de documentos XML largos, traducir datos entre diferentes ontologías (**DTDs**), integrar datos XML procedentes de múltiples fuentes XML, transportar grandes cantidades de datos XML a clientes o enviar consultas a fuentes XML.

XML-QL utiliza un grafo XML como modelo de datos, donde los nodos contienen o bien el **ID** del elemento (si éste posee uno) o bien un identificador único de objeto (**OID**). Los arcos de tipo jerárquico están etiquetados con el nombre del elemento a cuyo identificador apuntan. Los arcos cruzados unen el ID de un elemento con sus valores de los IDREFS correspondientes, y están etiquetados con los nombres de tales atributos. Los arcos de tipo **PCDATA** apuntan directamente a los valores de los elementos.

XML-QL soporta el orden en los datos de entrada y de salida, permitiendo consultas referenciando el índice de un subelemento.

La sintaxis de XML-QL combina elementos de la sintaxis de XML con la sintaxis de los lenguajes de consulta tradicionales.

En una consulta XML-QL los patrones y filtros aparecen en la cláusula *WHERE*. Las variables aparecen precedidas por el símbolo $. Los patrones son del estilo *pattern in source*, donde *pattern* indica cómo recorrer un objeto XML y *source* es una referencia a un objeto XML (generalmente su **URL**). Los patrones admiten expresiones regulares para indicar la ruta para recorrer un objeto XML, mediante los operadores "*" (*kleene-star*), "(alternativa)", "+" (una o más ocurrencias) y "." (concatenación).
Por ejemplo el patrón <bib.book.title>$t</> recorrería diversas etiquetas de una vez, y <bib.(book | cd).title>$t</> indicaría un patrón disyuntivo.

Los filtros aparecen a continuación del patrón y admiten operadores booleanos, así como los operadores de comparación ">", ">=", "<", "<=", "=" "!=".

XML-QL permite uniones entre DTDs.
El constructor, a modo de plantilla de un fragmento XML, figura en la cláusula *CONSTRUCT*. Admite las funciones de grupo *min*, *max*, *count*, *sum* y *avg*. Las consultas XML-QL pueden anidarse, incrustándolas en la cláusula *CONSTRUCT*.

XML-QL también permite consultas planas del estilo «muestra los títulos y autores de todos los libros», en este caso el constructor de XML-QL elimina los duplicados.

Veamos como se realizaría la consulta de selección de todos los libros publicados por la editorial Addison-Wesley después del año 91, nos interesa solamente el año y el título sobre el documento bib.xml

```
WHERE
  <bib>
    <book year=$y>
      <title>$t</title>
      <publisher><name>Addison-Wesley</name></publisher>
    </book>
  </bib> IN "www.bn.com/bib.xml",$y > 1991
CONSTRUCT <book year=$y><title>$t</title></book>
```

3.4 XQL-99

XQL 99 es una versión optimizada de **XQL 98;** es un lenguaje de consulta orientado al documento, que utiliza XML como modelo de datos, donde cada nodo del árbol representa un elemento o subelemento conteniendo su nombre. Todos los atributos son accesibles como campos desde los elementos a los que pertenecen. Las hojas contienen los valores de los elementos. En XQL 98 las relaciones entre nodos se basaban en el modelo de **XPointer** y podían ser de tipo jerárquico (padre/hijo, ascendiente/descendiente), posicional (absoluta, relativa, rango) y secuencial (precede, inmediatamente precede).
XQL 99 extiende este modelo para soportar relaciones establecidas vía uniones inter-documentos (se pueden combinar subárboles de documentos en las consultas) y de referenciación de enlaces.

La entrada de una consulta en XQL es un conjunto de uno o más documentos XML, que pueden provenir de diferentes fuentes. El resultado es una lista de nodos, los cuales se pueden representar de manera conveniente, por ejemplo como nodos **DOM**, texto XML serializado, texto XML, *XPointers*, enlaces, etc., según el entorno donde se ejecuta la consulta, y pueden servir de base para posteriores consultas.

Para las expresiones de camino XQL usa el operador "*" como máscara para indicar cualquier camino arbitrario en la consulta y los operadores "/" para la relación padre/hijo y "//" para la relación ancestro//descendiente. XQL permite referenciar un subelemento por su índice (posición relativa en el árbol) y preguntar por el i-ésimo elemento con etiqueta *X*. XQL también permite poder expresar una proximidad estructural (operadores inmediatamente precede "*;*" y precede "*;;*") y un orden de precedencia (operadores *before* y *after*). Los filtros aparecen entre los símbolos "[" y "]".
Las condiciones pueden ser evaluadas en cualquier nivel de un documento, sin tener que navegar desde la raíz. Admite los operadores booleanos: *and*, *or* y la función *not*(), así como los operadores relacionales: "=", "!=", "<", "<=", ">" y ">=". XQL también permite búsqueda de patrones a través de los operadores *contains*, e *icontains*, y el uso de máscaras "?" y comodines "*".
Una consulta XQL no contiene cláusula *constructor* que indique el formato de salida sino que las expresiones del patrón determinan el resultado. Permite indicar un rango de elementos a devolver, pero no permite la creación de nuevos elementos. Los resultados mantienen el anidamiento original de los nodos del documento de entrada, si bien mediante un operador de concatenación, "," se permite especificar un orden dentro de una lista resultado.

XQL permite agrupar resultados usando la estructura del documento original, y expresándolo mediante el uso de llaves: "*{*" y "*}*".
XQL no permite la ordenación de los resultados por un cierto criterio. Admite la función de agregado *count*().
XQL posee además dos operadores para la obtención de resultados: "*?*" y "*??*". El primero, "*?*", devuelve sólo el nodo al que se le aplica (reducción), sin incluir sus atributos ni hijos, mientras que el segundo, "*??*", devuelve el elemento con todos sus hijos.

Veamos como se realizaría la consulta de selección sobre XQL-99 de todos los libros publicados por la editorial Addison-Wesley después del año 91, nos interesa solamente el año y el título sobre el documento bib.xml

```
document("http://www.bn.com")/bib {
  book[publisher/name="Addison-Wesley" and @year>1991]{
    @year | title
  }
}
```

3.5 XQuery

XQuery es un lenguaje derivado directamente de **Quilt**, nacido para consultar fuentes de datos heterogéneas, el cual a su vez se inspira en el diseño de los siguientes lenguajes:
- XPath y XQL (lenguajes orientados al documento). De ellos adopta la sintaxis para navegar en documentos jerárquicos, ya que para consultar documentos se requiere preservar el orden y la jerarquía.
- SQL (lenguaje relacional). De SQL coge la idea del esquema de cláusulas *SELECT-FROM-WHERE* para reestructurar los datos y operaciones tales como uniones *(join)* y agrupaciones.
- OQL (lenguaje orientado a objetos). De éste toma la noción de lenguaje funcional, compuesto por diferentes clases de expresiones que puede anidarse.
- XML-QL (lenguaje orientados a información semi-estructurada). De este lenguaje adopta la idea de asignar variables que después se utilizarán para crear nuevas estructuras, especificadas en la Recomendación del W3C *XML Schema: structures*.

XQuery, al igual que Quilt, se trata de un lenguaje funcional mediante el cual una consulta se representa como una expresión. La estructura y apariencia de una consulta diferirá significativamente dependiendo del tipo de expresiones utilizadas.

XQuery está diseñado para aplicarse a todos los tipos de fuentes de datos XML. La entrada y la salida de una consulta son instancias del modelo de datos XPath 2.0: secuencia ordenada de nodos, cada uno de ellos pudiendo contener secuencias anidadas de nodos. Ello permite modelar no sólo un documento XML, sino también un fragmento bien formado de un documento, una secuencia de documentos, o una secuencia de fragmentos de documentos.

Al igual que en Quilt, las principales formas de una expresión XQuery son las siguientes:

- Expresiones XPath para navegar por los documentos.
- Constructores de elementos.
- Expresiones *FLWR* (*FOR/LET*, *WHERE*, *RETURN*) para iterar sobre los elementos de una colección.
- Filtrado de elementos de un documento conservando la jerarquía en los elementos seleccionado (operador *FILTER*)
- Expresiones que integran operadores, funciones predefinidas (de grupo: *min*, *max*. *avg*, *sum*, *count*), y funciones definidas por los usuarios (las cuales pueden ser recursivas).
- Expresiones condicionales (*IF*, *THEN*, *ELSE*), muy útiles para construir el resultado dependiendo de alguna condición
- Expresiones con cuantificadores (*SOME*, *ANY*) para chequear la existencia de algún elemento que cumpla una condición o determinar si todos los elementos de una categoría la satisfacen.
- Expresiones que testean o modifican tipos de datos.

3.5.1 Expresiones XPath

Una expresión **Xpath** consiste en una serie de pasos que representan movimientos a través de un documento en una determinada dirección, y cada paso puede aplicar un predicado para eliminar nodos que no cumplen una condición dada. Los nodos resultantes de un paso sirven de entrada para el siguiente paso.
Una expresión XPath puede empezar identificando un nodo específico, o una secuencia de nodos en un documento. A través de la función *document(string)* se obtiene el elemento raíz del documento indicado, y comenzando con "/" o "//" se representa de forma implícita un nodo raíz determinado por el entorno de ejecución.
Xpath permite seleccionar un nodo de una secuencia especificando su ordinal. También permite especificar una secuencia de ordinales, así como un rango.

Los nodos resultantes de una expresión XPath están ordenados según su posición en la jerarquía original. Si los nodos proceden de diversos documentos, el orden dependerá de la implementación.
A los operadores usuales de XPath, Xquery añade un operador denominado *dereference* ("=>") cuyo operando izquierdo es un nodo de tipo IDREF(S) (sólo se puede usar en documentos que poseen Schemas o DTDs ya que el operador necesita encontrar los nodos de tipo ID, IDREF(S)).

3.5.2. Constructores de elementos

La manera más simple de generar un nuevo elemento es incrustándolo directamente en la consulta usando notación XML. Los constructores de elementos constan de una etiqueta de inicio y final (expresadas con una constante o una variable), que encierran una lista opcional de expresiones (que también pueden ser variables) las cuales proporcionan el contenido del elemento.

Ejemplo de consulta en XQuery: *Generar un elemento <producto_proveedor> que contenga el atributo "precio" y los elementos anidados <codprod> y <cod prov>:*

```
<producto_proveedor precio = $pr>
<codprod> $cpd </codprod>
<codprov> $cpv </codprov>
</producto_proveedor>
```

Cuando el contenido de un elemento o el valor de un atributo se ha de calcular mediante una expresión ésta se encierra entre llaves. En el caso en que sea el nombre de un elemento o atributo el que se haya de calcular mediante una expresión ésta también se encierra entre llaves pero solamente en la etiqueta de inicio, apareciendo sin nombre la etiqueta del final.

3.5.3. Expresiones FLWR

Las expresiones *FLWR* (*FOR, LET, WHERE, RETURN*) se utilizan siempre que es necesario iterar sobre los elementos de una colección. Los patrones aparecen en la cláusula *FOR*, donde se asocian expresiones a variable. Se puede utilizar la función *DISTINCT* para eliminar duplicados. La cláusula *LET* se utiliza generalmente para asignar una variable al valor de una expresión (sin iteración). El valor de la variable será una lista de nodos y sus descendientes (bosque ordenado), a los que generalmente se les aplicará posteriormente alguna función. La cláusula *WHERE* contiene los filtros, seleccionando las tuplas de la cláusula *FOR* que cumplan una determinada condición. La cláusula *RETURN* construye el resultado, para ello utiliza texto estructurado en XML, las variables asignadas en *FOR/LET*, expresiones XPath basadas en tales variables y subconsultas. Cuando el contenido de un elemento de salida se extrae del contenido de un elemento del documento de entrada pero se encapsula con otras etiquetas, entonces se aplica la función *text*(). Si el orden de los resultados no es importante se puede utilizar la función *unordered*. Si los resultados se desean ordenados según uno o más criterios se puede utilizar la cláusula *SORTBY* .

Ejemplo de consulta en XQuery: *Listar las editoriales, ordenadas por nombre, tales que el precio medio de sus libros sea menor que 30 euros:*

```
FOR $pub IN DISTINCT (document (“bib.xml”)//publisher)
LET $a := avg(document(“bib.xml”)
/book[publisher = $pub] / price)
WHERE $a < 30
RETURN
<publisher>
<name>$pub</name>,
<avgprice> $a </ avgprice >
</publisher> SORTBY (name)
```

3.5.4 Operadores

XQuery admite los operadores relacionales “=”, “!=”, “<”, “<=”, “>” y “>=”, cuyos operandos son valores simples del mismo tipo.
XQuery añade unos operadores para comparaciones de identidad de nodos: “==” y “!==” cuyos operandos son nodos. Admite los operadores lógicos *AND* y *OR*, pero no *NOT* . También los operadores de secuencia *BEFORE* y *AFTER*.
En XQuery se pueden formar secuencias de nodos mediante el operador «,», rangos (operador “*TO*”) y también formar una nueva secuencia de nodos combinando otras mediante la unión, la intersección y diferencia de conjuntos (*UNION, INTERSECT* y *EXCEPT*).

3.6 Comparativa

Veamos ahora mediante unas tablas la comparativa entre los cuatro lenguajes de consulta vistos. La primera tabla comparativa se centra en las funciones principales, modelo de datos en que se basan y las fuentes de entrada/salida que aceptan y aportan

	Lorel	**XSLT**	**XML-QL**	**XQL 99**	**XQuery**
Funciones principales	Consultas de datos semi-estructurados	Transforma-ción de documentos	Consultas de datos, transformaciones, integración de datos XML de diferentes fuentes	Consultas en el interior de un documento y consultas en colecciones de documentos	Consultas sobre fuente de datos heterogéneas
Modelo de datos	Grafo / Árbol	Árbol (como Xpath 1.0)	Grafo	Árbol (DOM de XML)	Secuencia ordenada de nodos (como Xpath 2.0).
Fuente y formato de entrada	Documentos XML	Documentos XML XML + Hoja de estilo	Documentos XML de diferentes fuentes	Documentos XML	Documentos XML, fragmentos XML, colecciones de documentos XML
Información de salida	Documento XML (lista ordenada)	Documento XML, colecciones de docs.	Documento XML (fragmentos XML)	Documento XML (fragmentos XML, lista de elementos resultantes)	Documento XML, fragmento XML, colecciones XML.

A partir de esta tabla comparativa podemos extraer que Lorel y XML-QL poseen un modelo de datos propio (grafo), XQL adopta el modelo de datos implícito de XML (árbol), XSLT utiliza el mismo modelo de datos que XPath 1.0, mientras que XQuery se basa en el modelo de XPath 2.0, todavía en desarrollo.

Las consultas en XML se pueden aplicar tanto a un solo documento como a una colección de documentos.
Como información de salida generalmente se construye un nuevo documento XML (al cual se le puede volver a aplicar una consulta), a partir de los fragmentos XML seleccionados por la consulta. Tan solo XSLT, XQL y XQuery pueden producir como resultado múltiples documentos, a los cuales se les puede envolver en un nodo raíz común para crear como resultado un documento XML bien formado. En el caso de XQuery no es preciso ya que el resultado puede modelarse como una secuencia de nodos, cada uno de los cuales a su vez puede contener secuencias anidada de nodos.
Ninguno de los lenguajes estudiados devuelve un conjunto de puntos de concordancia *(match points)*, para identificar las porciones concretas del texto que son relevantes en un determinado documento, ni tampoco permiten regresar de un elemento resultado al documento que lo contenía. Tampoco, ninguno de los lenguajes analizados permite realizar búsquedas por similitud ni ordenaciones por relevancia.

Sintaxis del lenguaje XQuery 1.0

1 Documentos de referencia

Cada vez, mas información que se almacena, intercambia y presenta se hace utilizando XML. Una de las grandes ventajas de XML es la de su flexibilidad en la representación de los datos mediante diversos tipos de información. Para explotar esta flexibilidad un lenguaje de consultas XML debe proporcionar las características necesarias para recuperar e interpretar la información proveniente de estas fuentes diversas.

XQuery se diseña para resolver los requisitos identificados por el grupo W3C XML Query Working Group (XML 1.0 Requeriments) y los casos de uso (XML Query Use Cases).

XQuery Version 1.0 es una extensión del estándar XPath 2.0. Una expresión que es sintácticamente correcta y se ejecuta en ambos lenguajes debe devolver el mismo resultado en ambos lenguajes.

Hasta ahora se han definido ocho borradores de trabajo desde el W3C, tenemos, pues, que estos documentos definen y describen en su totalidad el lenguaje XQuery. Actualmente XQuery está formado por:

XML Query Requeriments
: Documento de planificación para el grupo de trabajo. Una lista de desiderátum de XQuery

XML Query Use Cases
: Distintos casos reales y fragmentos de XQuery que resuelven problemas específicos

XQuery 1.0: An XML Query Language
: Documento central que constituye una introducción al lenguaje y una descripción general de la mayor parte de conceptos. Este es el documento base para realizar el presente apartado

XQuery 1.0 and XPath 2.0 Data Model
: Ampliación de la información sobre XML. En este documento se describen elementos de datos que deben quedar claros en una implementación de una consulta, y los conceptos básicos de la semántica formal

XQuery 1.0 Formal Semantics
: Álgebra que define el lenguaje

XML Syntax for XQuery 1.0 (XQueryX)
: Sintaxis alternativa para aquellos que prefieren XML.

XQuery 1.0 and XPath 2.0 Functions and Operators Version 1.0
: Funciones y operadores básicos de tipos de datos de esquema y nodos y secuencias de nodos XQuery

XML Path Language (XPath) 2.0
: Documentación de XPath

2 XQuery Data Model

2.1 Introducción

El Modelo de Datos de XQuery 1.0 ha sido definido conjuntamente por el XML Query Working Group (Grupo de Trabajo de XML Query) y el XSL Working Group (Grupo de trabajo XSL) del W3C.
El Data Model tiene dos objetivos fundamentalmente:

- Definir de manera precisa la información que un procesador XSLT o XQuery puede aceptar
- En segundo lugar, define todos los valores permitidos de expresiones en los lenguajes XSLT, XQuery, y XPath

Los documentos que son aceptados por este Data Model son:

- Documentos bien formados
- DTD validos
- XML validados por un XML Schema

Un lenguaje se denomina cerrado respecto a un modelo de datos (Data Model) si el valor de cada expresión en el lenguaje se garantiza que pertenece al modelo de datos; XSLT 2.0, XQuery 1.0 y XPath 2.0 son lenguajes cerrados respecto a este Data Model

El modelo de datos se basa en el XML Information Set (información de XML fija o Infoset) pero requiere las siguientes características adicionales:

- Soporte de los esquemas de XML. Las recomendaciones del esquema de XML definen características, tales como estructuras y tipos de datos simples que amplían la información de XML fijada con el tipo exacto información.
- Representación de colecciones de documentos y valores complejos

Como en el Infoset, el Modelo de Datos de XQuery especifica que información de los documentos XML es accesible, pero no se especifica los lenguajes de programación ni interfaces usados para representar o acceder a estos datos

2.2 Nodos, Valor atómico y secuencia

Cada valor manejado por el modelo de los datos es una secuencia de cero o más elementos. Un elemento puede ser un nodo o un valor atómico.
Como ya se ha definido anteriormente un nodo es un elemento del arbol XML; los nodos XML pueden ser de siete tipos diferentes: documento, elemento, atributo, texto, namespace, instrucción de procesamiento y comentario. Un valor atómico es un valor en el espacio del valor de un tipo atómico etiquetado con ese tipo atómico; siendo un tipo atómico un tipo simple primitivo o un tipo derivado por la restricción de un tipo simple primitivo. Los tipos derivados por una lista o una unión es no atómica

Una secuencia es una colección ordenada de nodos, valores atómicos o cualquier conjunto de nodos y valores atómicos. Una secuencia no puede ser nunca parte de otra secuencia.

El modelo de los datos soporta algunas clases de valores que no se aceptan en XML. Ejemplos de éstos son fragmentos bien formados de documentos, secuencias de fragmentos o secuencias de documentos.
El modelo de los datos también soporta los valores que no son nodos. Los ejemplos de éstos son valores atómicos, secuencias de valores atómicos, o secuencias que mezclan nodos y valores atómicos. Éstos son necesarios para poder representar los resultados de expresiones intermedias en el modelo de los datos durante el procesamiento de la expresión.

2.3 Orden del documento

Respecto al orden de los documentos XML a los que queremos aplicar el presente Data Model, tenemos algunas restricciones que se deben cumplir:

- El nodo del documento es el primer nodo
- El orden relativo de los nodos hermanos se determina por el orden de representación en el documento XML
- Los nodos de elementos ocurren antes que sus nodos hijos; los nodos hijos ocurren antes que los nodos hermanos del nodo padre
- Los nodos del tipo namespace aparecen inmediatamente después del nodo con el que está asociado.
- Los nodos atributo aparecen inmediatamente después de los nodos con el elemento al que están asociados (dando valor)

2.4 Funciones de Acceso a Nodos

Como ya se ha comentado anteriormente, un nodo puede ser de siete tipos diferentes: documento, elemento, atributo, texto, namespace, instrucción de procesamiento y comentario.
El Data Model define una serie de funciones que permiten realizar el acceso a los nodos obteniendo información de estos.

Veamos cuales son estos operadores de accesos genéricos para cualquier tipo de nodo al que estemos accediendo:

Función	Descripción
base-uri($n as Node) devuelve xs:anyURI?	Devuelve la URI de referencia del elemento
node-kind($n as Node) devuelve xs:string	devuelve el tipo de nodo sobre el que accedemos
node-name($n as Node) devuelve xs:QName?	Devuelve el nombre del nodo
parent($n as Node) devueleve Node?	Devuelve el nodo padre del nodo actual
string-value($n as Node) devuelve xs:string	Devuelve el contenido del nodo
typed-value($n as Node) devuelve xdt:anyAtomicType*	devuelve el valor literal del nodo actual. Si el nodo es un comentario, documento, namespace, instrucción de proceso o texto el valor devuelto será el mismo que tiene. Si el nodo es un atributo devolvería el tipo de atributo
type($n as Node) devuelve xs:Qname?	Devuelve el nombre del tipo del nodo actual
children($n as Node) devuelve as Node*	Devuelve los nodos hijos del nodo actual en el orden de aparición en el documento
attributes($n as Node) devuelve AttributeNode*	Devuelve la secuencia de los atributos del nodo de contexto o actual
nilled($n as Node) as xs:bolean	Accede a la propiedad nilled del nodo si este es un elemento, para los demás tipos de nodo, devuelve falso

Observamos que para la representación de la sintaxis de los operadores de acceso se utiliza la misma notación vista en los DTD's, de forma que tenemos:

V* significa una secuencia de cero o mas ítems del tipo V
V? significa una secuencia de exactamente cero o un item del tipo V
V+ significa una secuencia de uno o más elementos del tipo V

donde el tipo V puede ser un Nodo o un Valor Atómico

Además de los operadores de acceso vistos anteriormente y aplicables a todo tipo de nodo, tenemos que para alguno de los tipos de nodo específicos disponemos de una serie de operadores propios. Los nodos document y element disponen de operadores de acceso adicionales, mientras que los nodos attribute, namespace, instrucciones de proceso, comentario y texto no disponen de ellos

2.4.1 Funciones adicionales de Documento

Los nodos documento encapsulan diversos documentos XML. Estos nodos documento contienen la siguiente información: base-uri, children, unparsed-entities y document-uri.
Los nodos de tipo documento deben satisfacer las siguientes restricciones:

1. Cada nodo documento debe tener una identidad única, distinta de el resto de los nodos.
2. Los nodos hijo deben consistir exclusivamente de elemento, instrucción de procesamiento. comentario, y nodos del texto si no es vacío. Los nodos atributo, namespace, y nodos documento nunca pueden aparecer como nodos hijos
3. La secuencia de los nodos hijos de un nodo viene dada por el orden según aparecen en el documento XML
4. La propiedad hijos no debe contener dos nodos consecutivos del tipo texto.
5. Si un nodo N es hijo de un documento D, entonces el nodo padre de N debe ser D
6. Si un nodo N tiene como nodo padre un documento D, entonces N debe ser uno de los hijos de D
7. Cada elemento hijo de un documento debe ser distinto a los demás

Operadores adicionales de Documento

Operador	Descripción
unparsed-entity-system-id($node as DocumentNode, $entityname as xs:string)	Devuelve el identificador de sistema de la entidad externa parser declarada en el documento
unparsed-entity-public-id($node as DocumentNode, $entityname as xs:string)	Devuelve el identificador publico de la entidad parser externa especificada en el documento
document-uri($node as DocumentNode)	Devuelve la URI absoluta del recurso a partir el documento nodo se ha generado

2.4.2 Funciones adicionales de Elemento

Los nodos elemento encapsulan elementos XML. Estos nodos tienen almacenada la siguiente información: base-uri, node-name, parent, type, children, attributes, namespaces y nilled

Los nodos elemento deben satisfacer las siguientes restricciones:

1. Cada nodo elemento debe tener un identificador único, distinto a los de todos los demás nodos
2. Los nodos hijos deben ser nodos elemento, instrucción de procesamiento, comentario o nodo texto. Nodos atributo, namespace y documento nunca podrán ser nodos hijos de un nodo elemento
3. La secuencia de aparición de los nodos hijos seguirán el orden de aparición de los elementos en el documento
4. La propiedad hijo no puede contener dos nodos de texto consecutivos
5. Todo nodo hijo de un nodo elemento deben ser distintos
6. Los atributos de un nodo elemento deben tener distintos nombres
7. Los nodos namespace de un nodo elemento deben tener distintos nombres
8. Si un nodo N es el hijo de un nodo elemento E, entonces el nodo padre de N debe ser E
9. Si un nodo N tiene como padre el nodo elemento E, entonces N debe aparecer entre los nodos hijos de E
10. Si un nodo atributo A tiene como padre un nodo elemento E, entonces A debe aparecer como uno de los atributos de E

Operadores adicionales de Element

Operador	Descripción
element-declaration($node as ElementNode)	Devuelve el elemento global de declaración del nodo elemento actual

2 Gramática del XQuery 1.0

2.1 Conceptos básicos

El bloque de construcción básica del lenguaje XQuery es la ***expresión***. Este lenguaje proporciona gran cantidad de expresiones construidas a partir de palabras reservadas, símbolos y operandos, por lo general estos operandos serán nuevas expresiones. Así, en definitiva diremos que el lenguaje de consultas XQuery es un lenguaje funcional lo que significa que las expresiones se pueden jerarquizar.

El resultado obtenido de una expresión es una secuencia, que como ya comentamos anteriormente es una colección ordenada de cero o varios elementos, pudiendo ser estos elementos nodos o valores atómicos. (Véase 2.2 de este mismo capitulo)

2.2 Tipos

XQuery es un lenguaje fuertemente tipificado lo que significa que los operandos de varias expresiones, operadores y funciones deben formase a partir de los tipos previstos y comentados en el Data Model, estos elementos o tipos están basados en los XML Schemas.

Los tipos se dividen en dos grupos:

- Tipos de secuencia
- Tipos de conversión

Los primeros nos permiten referenciar a algún tipo de nodo y por supuesto a alguno de los atributos de esos nodos.

Los segundos, en cambio, nos permiten hacer una conversión automática cuando el tipo que se necesita como operando o resultado de una expresión no son los que realmente esperábamos.

Veamos algunos ejemplos de tipos de secuencias

Tipos de secuencias	Descripción
attribute ( ) ?	referencia a un atributo opcional (cero o un atributo)
element ( )	referencia a cualquier elemento
element (po:shipto, po:address)	referencia a un elemento que como nombre de elemento tenga po:shipto y como tipo de elemento po:ardes
element (po:shipto, *)	referencia a un nodo elemento con nombre po:shipto y sin restricciones en el tipo de elemento
node ( ) *	referencia a una secuencia de cero o más nodos de cualquier tipo
node ( ) +	referencia a una secuencia de uno o más nodos de cualquier tipo
attribute (@price, currency)	referencia un nodo atributo que como nombre tenga price y su tipo sea currency.
attribute (@*, currency)	referencia cualquier nodo atributo del tipo currency sin importar el nombre que tenga
attribute (catalog/product/@price)	referencia los nodos atributos de nombre price que pertenecen al elemento product en el elemento catalog.

2.3 Expresiones

2.3.1 Expresiones primarias

Las expresiones primarias son las primitivas básicas del lenguaje. Se incluyen en este tipo de expresiones los literales, variables, llamadas a funciones y constructores. Se pueden utilizar los paréntesis para controlar la precedencia de los operadores.

Literales: un literal es la representación sintáctica de un valor atómico. Como por ejemplo:

"12.5"
12E4
"El me dijo"

No podemos usar como caracteres de un literal los siguientes caracteres, deberemos utilizar sus referencias predefinidas

Referencia a Entidad	Carácter Representado
&lt;	<
&gt;	>
&amp;	&
&quot;	"
&aspos;	'
&#8364;	€

Así tendremos que:

Ben & Jerry's representa la cadena *"Ben & Jerry's"*

o

99.50€ representa la cadena *"99.50€"*

Variables: Una referencia a una variable es un nombre precedido por el signo '$'. Un ejemplo de variables serían:

$titulo o $autor

Expresión del elemento de contexto: el signo '.' permite evaluar el elemento (nodo) de contexto, que puede ser un nodo o un valor atómico

Por ejemplo:

doc("bib.xml"(//book[count(./author)>1]) accede a los nodos book que tienen más de un autor

(1 to 100)[. mod 5 eq 0] devolvería todos los elementos de 1 a 100 que sean múltiplos de 5.

Llamadas a funciones: Una llamada a una función consiste en un nombre seguido de paréntesis donde se encuentran una lista de cero o más expresiones, que en este caso se llaman argumentos

función-de-tres-argumentos(1,2,3)

función-de-dos-argumentos((1,2), 3)

Comentarios XQuery: los comentarios XQuery se utilizan para dar información adicional a la expresión y nunca son evaluables. Se representan con unos literales encerrados entre los signos "(:" y ":)"

Por ejemplo:

```
(: Esto es un comentario en XQuery:)
```

2.3.2 Expresiones de recorrido

Las expresiones de recorrido se utilizan para localizar nodos dentro de una estructura de nodos. Una expresión de recorrido (Path expression) consiste en una serie de pasos que comienzan de manera opcional con los signos "/" y "//" y separados entre ellos por el signo "/" o el signo "//".

El signo "/" al principio de la expresión de recorrido representa el nodo inicial del arbol, la raíz. Por lo tanto todas las expresiones de recorrido que empiecen por este signo denotaran la ruta absoluta del nodo al que accedemos. Por el contrario, el signo"//" denota que como secuencia del nodo inicial se incluyen todos los nodos hijos del nodo de contexto.

En una expresión de recorrido un paso nos generará una secuencia de elementos que filtraremos con una secuencia de predicados (de manera opcional). El valor que el paso nos generará serán los elementos que satisfagan estas condiciones o predicados

XQuery nos proporciona nos tipos de pasos, los filtros y los ejes.

Los filtros son simplemente expresiones primarias seguidas de algún predicado (si lo hubiese) y el resultado obtenido serán los elementos devueltos por la expresión primaria y que cumplen los predicados impuestos sin ningún orden.

El resultado de los ejes consisten siempre en una secuencia de nodos que son devueltos siempre en el mismo orden que el orden del documento.

Para ampliar información sobre los ejes véase 6.2.3 Sintaxis del lenguaje de transformación XPath del primer capitulo "EL lenguaje de marcas XML"

Por su parte los predicados de las expresiones de recorrido consisten en una secuencia de expresiones evaluables de manera lógica y que sintácticamente se representan entre corchetes "[]"

Veamos algunos ejemplos de expresiones de recorrido:

Expresión de recorrido	Descripción
child::chapter[2]	devuelve el segundo elemento chapter hijo del nodo de contexto
descendant::toy[attribute::color= "red"]	devuelve los nodos descendientes del elemento toy cuyo atributo tenga el valor de rojo
child:: text()	selecciona todos los nodos del tipo texto hijos del nodo de contexto
attribute::name	selecciona el atributo name del nodo de contexto
child::chapter/descendant::para	selecciona los elementos del tipo para descendientes de chapter hijo del nodo de contexto

También podemos utilizar la notación abreviada para escribir nuestras expresiones de recorrido. La notación abreviada permite:

- el nombre del eje puede omitirse de la expresión de recorrido, el eje que se utiliza por defecto es child.
- la notación que se usa para el acceso a los atributos (attribute::) se puede sustituir por @
- los signos "//" sustituyen la expresion //descendant-or-self::node()
- los la expresion parent::node() puede sustituirse por la expresion ".."

Veamos algunos ejemplos utilizando la notación abreviada:

expresión de recorrido	Expresión de recorrido abreviada
child::section/child::para	section/para
child::section/attribute::attribute(@id)	section/@id
child::para[attribute::type="error"	para[@type="error"]
/descendant-or-self::node()/child::para	//para
parent::node()/child::title	../title

2.4 Expresiones de secuencias

XQuery soporta operadores que nos permiten generar y combinar secuencias de elementos

Para generar secuencias se puede utilizar el operador coma ",", el cual evalúa cada uno de los operandos y como resultado devuelve la concatenación de los operandos en el mismo orden de aparición.
Una secuencia puede tener valores de nodos duplicados pero jamás un secuencia puede tener otra secuencia como elemento.

Veamos algunos ejemplos:

Expresión	Resultado	Explicación
(10,(1,2), (), (3,4))	10,1,2,3,4	Construye una nueva secuencia a partir de las secuencia 10, (1,2), secuencia vacía y (3,4)
(salary, bogues)		Genera todos los salary hijo del nodo contexto seguido por el elemento bonus hijo
($price, $price)	10.5 , 10.5	Si el valor de price es 10,5 el resultado de evaluar esta secuencia es la concatenación de los valores
(10, 1 to 4)	10,1,2,3,4	Genera una secuencia formada por el elemento 10 y los elementos del 1 al 4

Para combinar secuencias XQuery nos aporta una serie de expresiones que nos permiten obtener secuencias a partir de las operaciones algebraicas sobre las secuencias de origen

Expresión	Descripción
$sec1 union $sec2	genera una secuencia que contiene la secuencia sec1 y la secuencia sec2
$sec1 intersect $sec2	genera una secuencia a partir de los elementos repetidos en las secuencias sec1 y sec2
$sec1 except $sec2	genera una secuencia a partir de los elementos de la secuencia sec1 que no se encuentran en la secuencia sec2

2.5 Expresiones Aritméticas

XQuery aporta operadores aritméticos para realizar la suma (+), resta (-), multiplicación (*), división real (div), división entera (idiv) y modulo (mod).
Su uso es idéntico al de cualquier otro lenguaje.

Expresión	Resultado
-3 div 2	-1.5
-3 idiv 2	-1
$emp/hiredate – $emp/birthday	devuelve el resultado de restar a la fecha actual la fecha de nacimiento del elemento emp

2.6 Expresiones Lógicas

Las expresiones lógicas permiten que dos valores sean comparados. XQuery permite cuatro tipos de expresiones de comparación: comparaciones de valor, comparaciones generales, comparaciones de nodos y comparaciones de orden

Operadores de Comparacion		Descripción
eq	=	igualdad de valores
is		
ne	!=	distintos
isnot		
<	lt	menor que
>	gt	mayor que
<=	le	menor o igual
>=	ge	mayor o igual

Ejemplos:

```
$book1/autor eq “Kennedy”
```

compara el subelemento autor de book con “Kennedy”

```
<a>5</a> eq <a>5</a>
```

compara los dos nodos generados entre ellos

Las comparaciones generales se aplican a comparaciones aplicadas a operandos de cualquier longitud (secuencias). Se utilizan los mismos operadores sobrecargados vistos en las comparaciones de valor.

Ejemplos:

```
(1,2) = (2,3)
```

```
($a, $b) = ($c, 3.0)
```

Las comparaciones de nodos se aplican a la comparación de nodos, de manera que si los operandos a comparar pertenecen al mismo nodo la comparación devolverá true y false en otro caso

```
//book[isbn="34-0015-34"] is //book[cal="QA76.9 C3845"]
```

Por ultimo las comparaciones de orden utilizan los operadores de comparación de orden (<< y >>). El primero devuelve true si el nodo identificado en la parte izquierda de la comparación aparece antes que el nodo identificado en la parte derecha de la expresión en el orden del documento; el segundo (>>) devuelve true si el elemento de la izquierda de la expresión aparece en el documento después del elemento de la derecha de la expresión

```
//parchase[parcel="28-451"] << //sale[parcel="33-567"]
```

Las expresiones de comparación, todas ellas sean del tipo que sean, se pueden combinar utilizando los operadores lógicos and y or.

```
1 eq 1 and 2 eq 2
```

```
1 eq 1 or 2 eq 3
```

```
1 eq 2 and 3 idiv 0 = 1
```

2.7 Constructores

XQuery aporta constructores que permiten crear estructuras XML dentro de una consulta. Encontramos tantos constructores como tipos de nodos tenemos en una estructura XML y documentados en el Data Model.
Los constructores en XQuery se dividen en dos tipos:

- Constructores directos: los cuales utilizan una notación XML
- Constructores programados: que utilizan la notación basada en expresiones anidadas

2.7.1 Constructores directos de elementos

Un constructor de elemento crea un elemento XML. Si el nombre, atributos y contenido del elemento son todos ellos constantes, el constructor se basa en la notación estándar XML y en este caso se llaman constructores de elementos directos.

Veamos un ejemplo en el que mediante un constructor de elemento directo creamos un elemento <book> que contiene atributos, sub-elementos y texto:

```
<book isbn="isbn-0060229357">
        <title>Harold and de Purple Crayon</title>
        <author>
                <first>Crockett</first>
                <last>Johnson</last>
        </author>
</book>
```

En los constructores directos de elementos, podemos incluir expresiones mediante los delimitadores { }. Estas expresiones incluidas son evaluadas y sustituidas por su valor, mientras que el texto externo a estas expresiones incluidas se tratan como literales.

Veamos el siguiente ejemplo de consulta:

```
<example>
        <p> Here is a query. </p>
                <eg> $i//title </eg>
        <p> Here is the result of the query. </p>
                <eg>{ $i//title }</eg>
</example>
```

La consulta anterior en relacion al elementos XML book visto en el ejemplo anterior generaría la siguiente salida:

```
<example>
        <p> Here is a query. </p>
        <eg> $i//title </eg>
        <p> Here is the result of the query. </p>
        <eg><title>Harold and the Purple Crayon</title></eg>
</exemple>
```

Puesto que XQuery utiliza los símbolos de llaves { } para delimitar las expresiones incluidas, en el caso de necesitar, durante la creación de un elemento XML la aparición del literal "{" o "}" se puede utilizar el par "{{" o el par "}}" para que XQuery que en este caso se trata del uso del literal y no el delimitador de expresiones.

El resultado de un constructor de elemento es un nuevo nodo elemento, con su propia identidad de nodo. Todos los atributos y nodos descendientes del nodo creado son también nuevos nodos con sus propias identidades, incluso en el caso que estos nuevos nodos sean copias de nodos ya existentes.

La etiqueta de comienzo de un constructor directo de elemento puede contener uno o más atributos. Como en XML, cada atributo es especificado por un nombre y un valor.
En un constructor directo de elemento, el nombre de cada atributo es especificado por un QName constante, y el valor del atributo es especificado por una cadena de caracteres incluidos entre comillas simples o dobles.
De manera idéntica a los constructores de elemento, un valor de atributo puede contener las expresiones incluidas entre los signos { } que serán evaluados y substituidos por su valor durante el proceso de construcción del elemento.

Veamos algunos ejemplos:

```
<shoe size="7"/>
```

Añadimos el valor 7 al atributo size del elemento shoe, de manera idéntica podemos incluir expresiones anidadas:

```
<shoe size="{7}"/>
```

En este ejemplo el valor del atributo size es la cadena de caracteres de longitud cero

```
<shoe size="{()}"/>
```

Podemos determinar que el valor de los atributos sean una secuencia de valores, en el ejemplo siguiente el valor del atributo ref es la secuencia "[1 5 6 7 9]"

```
<chapter ref="[{1, 5 to 7, 9}]"/>
```

Por último, en este ejemplo el valor del atributo size será la concatenación de la cadena "As big as" con el valor del nodo accedido a través de la expresión de acceso a nodos $hat/@size.

```
<shoe size="As big as {$hat/@size}"/>
```

La parte de un constructor directo de elemento entre la etiqueta de comienzo y la etiqueta final se llama contenido del elemento constructor. Este contenido puede consistir en caracteres literales de texto, constructores jerarquizados de elemento y expresiones incluidas.
Por lo general, el valor de una expresión incluida puede ser cualquier secuencia de nodos y/o de valores atómicos. Las expresiones incluidas se pueden utilizar en el contenido de un constructor del elemento para obtener a través de la evaluación de la expresión tanto el contenido como los atributos del nodo construido.

Veamos algunos ejemplos:

```
<a>{1}</a>
```

Se construye un nodo que tiene un elemento hijo, un nodo texto, que contiene el valor "1"

```
<a>{1, 2, 3}</a>
```

El nodo construido tiene un hijo nodo texto que contiene el valor " 1 2 3"

```
<c>{1}{2}{3}</c>
```

El nodo construido tiene un nodo hijo de tipo texto que contiene el valor "123"

```
<fact>I saw 8 cats.</fact>
```

El elemento construido tiene un hijo del tipo texto que contiene el valor "I saw 8 cats". Como ya se ha comentado, podemos incluir expresiones incluidas en la generación de contenido, así, el siguiente ejemplo da como resultado el mismo elemento que el ejemplo anterior, la diferencia es que el valor 8 ha resultado de la evaluación de la expresión incluida "{5 + 3}" en lugar de utilizar el literal

```
<fact>I saw {5 + 3} cats.</fact>
```

Por último veamos un ejemplo donde el nodo construido tiene tres nodos hijos, un nodo texto conteniendo "I saw", un nodo hijo llamado <howmany> el que a su vez tiene un nodo hijo conteniendo el valor "8" y por último otro nodo hijo del tipo texto conteniendo el texto "cats".

```
<fact>I saw <howmany>{5 + 3}</howmany> cats.</fact>
```

Cuando se construye un elemento a partir de un constructor directo, pueden aparecer como parte del contenido del elemento los espacios en blanco. En otros casos, las expresiones incluidas y/o elementos jerarquizados solamente se pueden separar por espacios en blanco.
Por ejemplo, en la siguiente expresión, </title> y la etiqueta de inicio <author> están separadas por un carácter salto de carro y cuatro espacios en blanco:

```
<book isbn="isbn-0060229357">
   <title>Harold and the Purple Crayon</title>
   <author>
     <first>Crockett</first>
```

```
    <last>Johnson</last>
  </author>
</book>
```

Los espacios en blanco que aparecen en los límites entre las etiquetas y/o las expresiones incluidas, como en el ejemplo anterior, los llamaremos espacios en blanco de límite.
El prólogo contiene un declaración llamada xmlspace que controla si los espacios en blanco de límite son preservados por los constructores de los elementos. Si el xmlspace tiene el valor de "strip" (xmlspace=strip), los espacios en blanco de límite no se consideran significativos y se desprecian, en cambio si el valor de xmlspace es "preserve" (xmlspace=preserve) los espacios en blanco de límites se consideran significativos y se preservan.

Por último si queremos incluir algún comentario en el bloque de consulta, la manera de hacerlo sería :

```
(: This is an XQuery comment :)
<!-- This is an XML comment -->
```

De manera que el resultado de evaluar esta consulta XQuery daría como resultado un elemento XML que contendría solamente el comentario XML:

```
<!-- This is an XML comment -->
```

2.7.2 Constructores Programados

La segunda alternativa que tenemos para crear nodos además del uso de los constructores directos, son los constructores programados. Un constructor programado empieza con una palabra clave que indica el tipo de nodo que se está creando: element, attribute, document, text, pi (instrucción de procesamiento), comment o namespace.
Para las clases de nodos que tienen un nombre (elemento, atributo, instrucción de procesamiento y nodos namespace), la palabra que identifica el tipo de nodo que se está creando va seguida por el nombre del nodo; este nombre se puede especificar como un QName o como una expresión incluida llamada expresión de nombre (name expression) que devuelve una cadena de caracteres o un QName

El siguiente ejemplo muestra como se utilizarían los constructores programados en la creación de un elemento book. Este elemento contiene un atributo isbn y dos elementos hijos, title y author, el que contiene a su vez dos nodos elemento hijos llamados first y last.

```
element book {
        attribute isbn {"isbn-0060229357" },
        element title { "Harold and the Purple Crayon"},
        element author {
                element first { "Crockett" },
                element last {"Johnson" }
        }
}
```

Un constructor programado muchas veces se utilizará para hacer una copia modificada de un elemento existente. Por ejemplo, si la variable $e está limitado a un elemento con contenido numérico, el siguiente constructor puede ser utilizado para crear un nuevo elemento con el mismo nombre y atributos que $e pero con el contenido numérico igual al doble del valor de $e:

```
element {node-name($e)}
        {$e/@*, 2 * data($e)}
```

Si en este ejemplo, la variable $e está inicializada mediante la expresión

```
let $e := <length units=”inches”>{5}</length>
```

entonces, el resultado de la expresión anterior sobre este elemento será:

```
<length units=”inches”>10</length>
```

Una de las características más importantes de los constructores programados es permitir que el nombre del nodo que se está creando pueda ser obtenido a partir de una expresión incluida. Veamos un ejemplo que ilustre esta característica; la siguiente expresión pretende traducir el nombre del elemento que estamos creando según un idioma u otro.

Supongamos que la variable $dict contiene la secuencia de entradas en el diccionario de traducción, en este caso, por ejemplo:

```
<entry word="address">
        <variant lang="German">Adresse</variant>
        <variant lang="Italian">indirizzo</variant>
        <varinant lang=”Spanish”>Dirección</variant>
</entry>
```

Supongamos además que la variable $e tiene asignado el elemento:

```
<address>123 Roosevelt Ave. Flushing, NY 11368</address>
```

La siguiente expresión genera un nuevo elemento en el que el nombre de $e se ha traducido al italiano y el contenido de $e (incluidos sus atributos si tuviera) se han mantenido. La primera expresión incluida después de la palabra clave element genera el nombre del elemento, y la segunda expresión incluida genera el contenido y los atributos:

```
element
        {data($dict/entry[word=name($e)]/variant[lang="Italian"])}
        {$e/@*, $e/*}
```

así, el resultado obtenido será:

```
<indirizzo>123 Roosevelt Ave. Flushing, NY 11368</indirizzo>
```

Cuando se utiliza un constructor programado para atributos, no se realiza ninguna validación del atributo construido. Sin embargo, si el constructor programado del atributo está dentro de un constructor de elemento, el atributo será validado durante la validación del elemento padre.

```
attribute
        { if ($sex = "M") then "husband" else "wife" }
        { <a>Hello</a>, 1 to 3, <b>Goodbye</b> }
```

El nombre del atributo generado será husband o wife en función del valor de la variable $sex y su valor será “Hello 1 2 3 Goodbye”

Para crear nodos de tipo documento, debemos utilizar obligatoriamente constructores programados, el resultado de un constructor de nodo documento es un nuevo nodo documento, con su propia identidad del nodo.

El uso de un constructor de documento es útil cuando el resultado de una consulta sea un documento. El siguiente ejemplo ilustra una consulta que devuelve un documento XML que contiene un elemento raíz llamado author-list:

```
document
   {
     <author-list>
            {doc("bib.xml")//book/author}
     </author-list>
   }
```

Veamos un ejemplo completo de utilización de constructores programados:

```
let $ename := "altitude",
            $evalue := "10000",
            $nsURI := "http://example.org/metric-system",
            $attrname := "metric:unit",
            $attrvalue := "meter"
            return
                        element {$ename} {
                                    namespace metric {$nsURI},
                                    ttribute {$attrname} {$attrvalue},
                                    evalue
}
```

Que una vez ejecutado devolvería:

```
<altitude
          xmlns:metric = "http://example.org/metric-system"
          metric:unit = "meter">10000</altitude>
```

2.8 Expresiones FLWOR

XQuery proporciona una característica llamada expresiones FLWOR que soportan la iteración y asignación de resultados intermedios a variables. Este tipo de expresiones son muy útiles si queremos obtener resultados a partir de la combinación de dos o más documentos XML y es necesaria una reestructuración de esos datos obtenidos.
El nombre FLWOR se debe a las palabras For, Let, Where, Order by y Return.

Las cláusulas for (bucle de iteración de los lenguajes de programación estructurados) y let (asignación de variables) generan una secuencia de tuplas de variables, llamadas tuple stream (cinta de tuplas).
La cláusula Where se utiliza como filtro para conservar algunas de las tuplas y desechar otras, de la misma manera que la cláusula Where ejerce la restricción en el lenguaje SQL.
La cláusula Order by ordena la secuencia de tuplas generada por el for y el let siguiendo algún criterio de ordenación
La cláusula return construye el resultado de la expresión FLWOR, esta es evaluada una vez para cada tupla generada en la tuple stream después de filtrarla mediante la orden Where, usando la asignación de variables para cada tupla.
El resultado de la expresión FLWOR es una secuencia que contiene la concatenación de los resultados obtenidos en cada evaluación.

Veamos un ejemplo en el que se incluyen todas estas posibles cláusulas

```
for $d in doc("depts.xml")//deptno
let $e := doc("emps.xml")//emp[deptno = $d]
where count($e) >= 10
```

```
order by avg($e/salary) descending
return
    <big-dept>
        {
        $d,
        <headcount>{count($e)}</headcount>,
        <avgsal>{avg($e/salary)}</avgsal>
        }
     </big-dept>
```

La orden for produce una iteración para cada departamento del documento de entrada "depts.xml". Para cada iteración se asigna el número del departamento a la variable $d.
Para cada asignación de un numero de departamento a la variable $d la cláusula let asigna a la variable $e todos los empleados que corresponden al departamento en cuestión a partir del documento de entrada "emps.xml".
El resultado del for y el let es una secuencia de tuplas (tuple stream) en que para cada tupla tenemos un par del tipo $d $e ($d tiene la asignación del numero de departamento y $e tiene asignada los trabajadores de ese departamento).
La cláusula where filtra la secuencia de tuplas desechando aquellas tuplas en que el numero de empleados de un departamento sea menor que 10.
La cláusula order by ordena descendentemente las tuplas que cumplen la condición del where a partir del promedio de los salarios de estos empleados en el departamento. Por ultimo, la cláusula return construye un nuevo elemento <big-dept> para cada tupla conteniendo el numero de departamento, el numero de empleados (headcount) y el promedio del salario (avgsal).

2.8.1 cláusulas For y Let

Las cláusulas for y let se utilizan para producir una secuencia de tuplas en que cada tupla contiene una o más variables encadenadas.
El ejemplo más sencillo de la cláusula for contiene una variable y una expresión asociada. Se evalúa la expresión y se itera sobre los artículos de la secuencia obtenida, asignando cada elemento obtenido a la variable.

Veamos un ejemplo:

```
for $s in (<one/>, <two/>, <three/>)
return <out>{$s}</out>
```

En este ejemplo, la variable $s se itera para la expresión dad; primero para <one/>, luego para <two/> y finalmente para <three/>. Se genera una tupla para cada una de estas asignaciones y la cláusula return es invocada para cada tupla generando la siguiente salida:

```
<out>
        <one/>
</out>
<out>
        <two/>
</out>
<out>
        <three/>
</out>
```

La cláusula for también puede contener varias variables, cada una de ellas asociada a un expresión. En este caso la cláusula for itera cada variable sobre los elementos resultado de la evaluación de la expresión. En este caso el resultado obtenido contiene una secuencia de tuplas

generadas a partir del producto cartesiano de las tuplas obtenidas para cada una de las expresiones.

```
for $i in (1, 2), $j in (3, 4)
```

y la salida sería:

```
($i = 1, $j = 3)
($i = 1, $j = 4)
($i = 2, $j = 3)
($i = 2, $j = 4)
```

La cláusula let puede contener también una o mas variables, cada una de ellas asociada a un expresión. A diferencia de la cláusula for, la cláusula let asocia cada variable al resultado de su expresión asociada, sin utilizar la iteración. Las asignaciones a variables generadas a partir de la cláusula let son agregadas a las tuplas generadas por la cláusula for. Si no existe esta cláusula for, la cláusula let genera una tupla que contiene todas las variables obtenidas

```
let $s := (<one/>, <two/>, <three/>)
return <out>{$s}</out>
```

La variable $s se vincula al resultado de la expresión (<one/>, <two/>, <three/>). Como no existe la cláusula for se genera una sola tupla que contiene los valores asignados a $s, en este caso <one/>, <two/> y <three/>. La cláusula return se invoca para esta única tupla por lo que la salida que obtendríamos sería:

```
<out>
        <one/>
        <two/>
        <three/>
</out>
```

Cada variable utilizada en las cláusulas for o let, pueden tener un tipo de declaración opcional, que es uno de los tipos aceptados como Sequence Type.

Los tipos aceptados son:

xs:date	se refiere a una fecha
attribute() ?	se refiere a un atributo opcional
element()	se refiere a un elemento
node() *	se refiere a una secuencia de ningún o más nodos de cualquier tipo
item() +	se refiere a una secuencia de uno o más nodos o valores atómicos

Si el tipo definido para la variable de la cláusula no se corresponde con el tipo declarado según las reglas de las SequenceType, se producirá un error. Por ejemplo, el siguiente bloque de código genera un error ya que la variable $salary ha sido declarada con un tipo que no se corresponden con el valor asignado a esta variable:

```
let $salary as xs:decimal := "cat"
return $salary * 2
```

Cada variable utilizada en las cláusulas for y let pueden tener asociada una variable posicional. El nombre de la variable posicional está precedido por la palabra clave at. Toda variable posicional tiene siempre como tipo de definición el xs:integer. El funcionamiento de las variables posicionales es que mientras la variable itera sobre los ítems de una secuencia, la

variable posicional asociada a esa variable itera sobre los números ordinales de esas tuplas de la secuencia.
Veamos un ejemplo:

```
for $car at $i in ("Ford", "Chevy"),
$pet at $j in ("Cat", "Dog")
```

el resultado del cual sería:

```
($i = 1, $car = "Ford", $j = 1, $pet = "Cat")
($i = 1, $car = "Ford", $j = 2, $pet = "Dog")
($i = 2, $car = "Chevy", $j = 1, $pet = "Cat")
($i = 2, $car = "Chevy", $j = 2, $pet = "Dog")
```

2.8.2 La cláusula where

La cláusula opcional where se utiliza para filtrar las tuplas de variables generadas a partir de las expresiones for y let.
La cláusula where representa la restricción de tuplas a partir de los criterios de restricción.

```
avg(for $x at $i in $inputvalues
        where $i mod 100 = 0
        return $x)
```

2.8.3 Las clausulas order by y return

La cláusula order by contiene una o más especificaciones que ordenan las tuplas generadas por una expresión FLWOR, llamadas orderspecs.
Para cada tupla de la tuple stream se evalúan los orderspecs comprobando los valores asignados a las variables.

La cláusula return de una expresión FLWOR se evalúa una vez para cada tupla de la secuencia de tuplas devueltas por las cláusulas for o let. Los resultados de estas evaluaciones se concatenan para formar el resultado de la e4xpresion FLWOR.
Si la reexpresión no tiene la cláusula order by el orden de las tuplas vendrá determinado por el orden de las secuencias devueltas por la cláusula for.

Veamos un ejemplo sencillo, la expresión siguiente devuelve solamente los nombres (y no los salarios) de los empleados en orden descendente por sueldo el sueldo que cobran

```
for $e in $employees order by $e/salary return $e/name
```

Otro ejemplo sencillo sería la siguiente expresión que nos devuelve los libros cuyo precio sea menor que 100 y los ordena alfabéticamente por el titulo:

```
for $b in $books//book[price < 100]
order by $b/title
return $b
```

2.8.4 Ejemplos de expresiones FLWOR

Supongamos que tenemos un documento XML para representar los libros que tenemos en una determinada biblioteca:

```
<bib>
   <book>
         <title>TCP/IP Illustrated</title>
         <author>Stevens</author>
         <publisher>Addison-Wesley</publisher>
   </book>
    <book>
         <title>Advanced Unix Programming</title>
         <author>Stevens</author>
         <publisher>Addison-Wesley</publisher>
   </book>
   <book>
         <title>Data on the Web</title>
         <author>Abiteboul</author>
         <author>Buneman</author>
         <author>Suciu</author>
   </book>
</bib>
```

La siguiente consulta transforma el documento de entrada en una lista en que cada autor aparece solamente una vez seguido por la lista de los títulos de los libros que este autor ha escrito:

```
<authlist>
  {
    for $a in distinct-values($books)//author
    order by $a
    return
         <author>
             <name>
                   { $a/text() }
              </name>
              <books>
                   {
                    for $b in $books//book[author = $a]
                    order by $b/title
                     return $b/title
                    }
               </books>
         </author>
  }
</authlist>
```

El resultado obtenido sería:

```
<authlist>
         <author>
                  <name>Abiteboul</name>
                  <books>
                           <title>Data on the Web</title>
                  </books>
         </author>
         <author>
                  <name>Buneman</name>
                  <books>
                           <title>Data on the Web</title>
                  </books>
```

```
        </author>
        <author>
                <name>Stevens</name>
                <books>
                        <title>TCP/IP Illustrated</title>
                        <title>Advanced Unix Programming</title>
                </books>
        </author>
        <author>
                <name>Suciu</name>
                <books>
                        <title>Data on the Web</title>
                </books>
        </author>
</authlist>
```

2.9 Expresiones condicionales

XQuery soporta las expresiones condicionales basadas en las cláusulas if, then y else.
La expresión que sigue a la palabra clave if es la expresión condicional, siendo las expresiones que siguen al then y al else las then-expression y las else-expression, respectivamente.
Como es tradicional en este tipo de expresiones, si la expresión de condición es cierta se ejecutará o evaluará la expresión del then, en cambio, si la expresión de condición del if es falsa, o sea, devuelve false, entonces se evaluará la expresión del else.

Veamos algunos ejemplos sencillos:

```
if ($widget1/unit-cost < $widget2/unit-cost)
then $widget1
else $widget2
```

En el siguiente ejemplo, la condición del if está supeditada a la existencia de un atributo llamado discounted independientemente de su valor:

```
if ($part/@discounted)
then $part/wholesale
else $part/retail
```

2.10 Expresiones de cuantificación

Las expresiones de cuantificación soportan la cuantificación existencial y la universal. El valor de retorno de este tipo de expresiones es true o false.
Una expresión cuantificada empieza con un cuantificador, que es la palabra clase some (algunos) o every (cada uno), seguidos por la palabra clave satisfies (satisface) y una expresión de prueba.

Veamos algunos ejemplos:

Esta expresión es cierta si cada elemento parte tiene un atributo llamado discuonted (sin tener en cuenta el valor o valores de este atributo):

```
every $part in //part satisfies $part/@discounted
```

Esta expresión es cierta si al menos un elemento employee satisface la expresión de comparación:

```
some $emp in //employee satisfies ($emp/bonus > 0.25 * $emp/salary
```

El siguiente ejemplo, se evalúa la expresión sobre las nueve tuplas generadas por el producto cartesiano de las secuencias (1,2,3) y (2,3,4) y se comprueba si hay valores que cumplen la condición:

```
some $x in (1, 2, 3), $y in (2, 3, 4)
satisfies $x + $y = 4
```

3 Funciones y Operadores

XQuery proporciona una serie de funciones y operadores que proporcionan diversas funcionalidades sobre los tipos de datos o elementos de un documento XML. Estas funciones y operadores no son exclusivos de XQuery sino que son totalmente compatibles y pueden usarse en XPath, XSLT y otros estándares XML.

En el siguiente esquema se representa la jerarquía de tipos de datos que encontramos sobre los que se pueden aplicar estas funciones y operadores, aparecen los tipos estándar y también los tipos que se pueden generar.

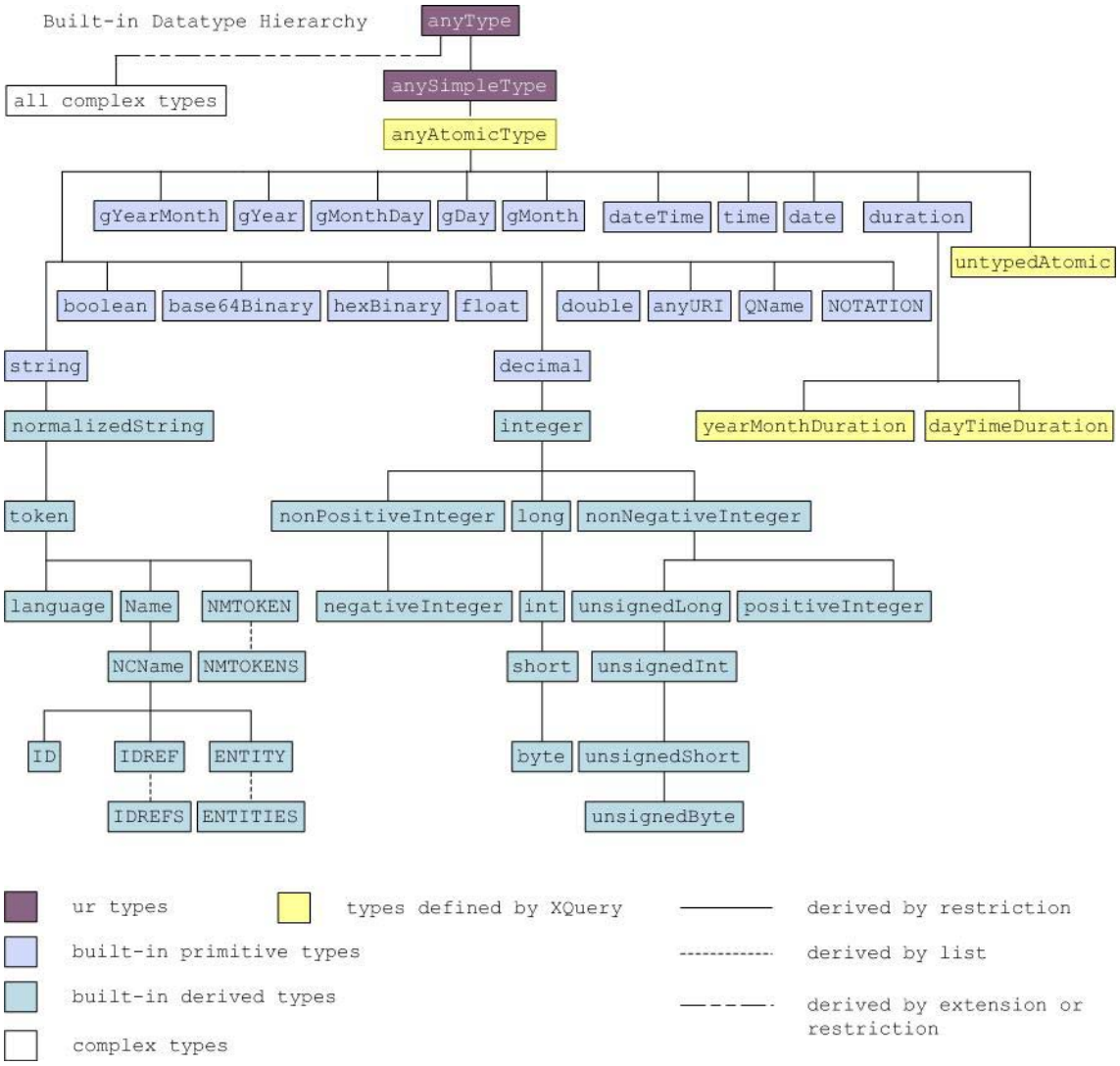


Figura 3.1. Jerarquía del tipo de datos en Xquery 1.0

3.1 Funciones de acceso

Estas funciones permiten acceder a los nodos de un arbol XML y extraer determinada información de esos nodos como pueden ser el nombre, tipo de elemento,…

Función	Parámetros	Valor Devuelto	Descripción
fn:node-kind	cualquier tipo de nodo	xs:string	Función que devuelve el tipo de nodo
fn:node-name	cualquier tipo de nodo	cero o un xs:Qname	Función que devuelve el nombre de los nodos que puedan tenerlo, en caso contrario no devuelve nada
fn:string	elemento	xs:string	Función que devuelve el elemento o item como string
fd:data	cero o mas nodos	una secuencia de valores atómicos	Función que a partir de una secuencia de nodos, devuelve el valor atómico o el tipo de nodo
fn:base-uri	Elemento, documento, instrucción de procesamiento o sin argumentos	cero o una xs:string	Devuelve el valor de la propiedad base-uri
fn:document-uri	Nodo	zero o un xs:string	Devuelve el valor de la propiedad document-uri del documento

3.2 Funciones constructoras

Las funciones constructoras permiten generar elementos de un determinado tipo de datos: string, decimal, float,…. Las siguientes funciones constructoras de tipos de datos están permitidas:

- **xs:string**($srcval as *xdt:anyAtomicType*) as *xs:string*
- **xs:boolean**($srcval as *xdt:anyAtomicType*) as *xs:boolean*
- **xs:decimal**($srcval as *xdt:anyAtomicType*) as *xs:decimal*
- **xs:float**($srcval as *xdt:anyAtomicType*) as *xs:float*
- **xs:double**($srcval as *xdt:anyAtomicType*) as *xs:double*
- **xs:duration**($srcval as *xdt:anyAtomicType*) as *xs:duration*
- **xs:dateTime**($srcval as *xdt:anyAtomicType*) as *(xs:dateTime, xdt:dayTimeDuration)*
- **xs:time**($srcval as *xdt:anyAtomicType*) as *(xs:time, xdt:dayTimeDuration)*
- **xs:date**($srcval as *xdt:anyAtomicType*) as *(xs:date, xdt:dayTimeDuration)*

- **xs:gYearMonth**($srcval as *xdt:anyAtomicType*) as *xs:gYearMonth*
- **xs:gYear**($srcval as *xdt:anyAtomicType*) as *xs:gYear*
- **xs:gMonthDay**($srcval as *xdt:anyAtomicType*) as *xs:gMonthDay*
- **xs:gDay**($srcval as *xdt:anyAtomicType*) as *xs:gDay*
- **xs:gMonth**($srcval as *xdt:anyAtomicType*) as *xs:gMonth*
- **xs:hexBinary**($srcval as *xdt:anyAtomicType*) as *xs:hexBinary*
- **xs:base64Binary**($srcval as *xdt:anyAtomicType*) as *xs:base64Binary*
- **xs:anyURI**($srcval as *xdt:anyAtomicType*) as *xs:anyURI*
- **xs:anyURI**($srcval as *xdt:anyAtomicType*) as *xs:QName*
- **xs:normalizedString**($srcval as *xdt:anyAtomicType*) as *xs:normalizedString*
- **xs:token**($srcval as *xdt:anyAtomicType*) as *xs:token*
- **xs:language**($srcval as *xdt:anyAtomicType*) as *xs:language*
- **xs:NMTOKEN**($srcval as *xdt:anyAtomicType*) as *xs:NMTOKEN*
- **xs:Name**($srcval as *xdt:anyAtomicType*) as *xs:Name*
- **xs:NCName**($srcval as *xdt:anyAtomicType*) as *xs:NCName*
- **xs:ID**($srcval as *xdt:anyAtomicType*) as *xs:ID*
- **xs:IDREF**($srcval as *xdt:anyAtomicType*) as *xs:IDREF*
- **xs:ENTITY**($srcval as *xdt:anyAtomicType*) as *xs:ENTITY*
- **xs:integer**($srcval as *xdt:anyAtomicType*) as *xs:integer*
- **xs:nonPositiveInteger**($srcval as *xdt:anyAtomicType*) as *xs:nonPositiveInteger*
- **xs:negativeInteger**($srcval as *xdt:anyAtomicType*) as *xs:negativeInteger*
- **xs:long**($srcval as *xdt:anyAtomicType*) as *xs:long*
- **xs:int**($srcval as *xdt:anyAtomicType*) as *xs:int*
- **xs:short**($srcval as *xdt:anyAtomicType*) as *xs:short*
- **xs:byte**($srcval as *xdt:anyAtomicType*) as *xs:byte*
- **xs:nonNegativeInteger**($srcval as *xdt:anyAtomicType*) as *xs:nonNegativeInteger*
- **xs:unsignedLong**($srcval as *xdt:anyAtomicType*) as *xs:unsignedLong*
- **xs:unsignedInt**($srcval as *xdt:anyAtomicType*) as *xs:unsignedInt*
- **xs:unsignedShort**($srcval as *xdt:anyAtomicType*) as *xs:unsignedShort*
- **xs:unsignedByte**($srcval as *xdt:anyAtomicType*) as *xs:unsignedByte*
- **xs:positiveInteger**($srcval as *xdt:anyAtomicType*) as *xs:positiveInteger*
- **xdt:yearMonthDuration**($srcval as *xdt:anyAtomicType*) as *xdt:yearMonthDuration*

- **xdt:dayTimeDuration**($srcval as *xdt:anyAtomicType*) as *xdt:dayTimeDuration*
- **xdt:untypedAtomic**($srcval as *xdt:anyAtomicType*) as *xdt:untypedAtomic*

3.3 Funciones y Operadores numéricos

XQuery permite el uso de operadores y funciones aritméticas. El tipo de dato devuelto por estas funciones y operadores dependerá del tipo de datos de los argumentos de las funciones ya que como en todos los lenguajes los operadores y funciones aritméticas están sobrecargadas

3.3.1 Operadores para valores numéricos

Operadores	Descripción
op:numeric-add	Suma
op:numeric-subtract	Resta
op:numeric-multiply	Multiplicación
op:numeric-divide	división
op:numeric-integer-divide	Dimisión entera
op:numeric-mod	Modulo
op:numeric-unary-plus	Signo positivo unario (+)
op:numeric-unary-minus	Signo negativo unario (-)

3.3.2 Comparacion de valores numéricos

Operadores	Descripción
op:numeric-equal	Comparación de igualdad
op:numeric-less-than	Comparación Menor que
op:numeric-greater-than	Comparación mayor que

3.3.3 Funciones para valores numéricos

Funciones	Ejemplo	Descripción
fn:floor	fn:floor(10.5) devuelve 10 fn:floor(-10.5) devuelve 11	Devuelve el numero entero más próximo al argumento
fn:ceiling	fn:ceiling(10.5) devuelve 11 fn:ceiling(-10.5) devuelve -10	Devuelve el numero entero mas pequeño mayor o igual que el argumento
fn:round	fn:round(2.5) devuelve 3 fn:round(2.49999) devuelve 2	Redondea un numero al más próximo entero
fn:round-half-to-even	fn:round-half-to-even(4.7564E-3, 2) devuelve 0.0E0. fn:round-half-to-even(35612.25, -2) devuelve to 35600.	Redondea un numero dado en función de la precisión

3.4 Funciones de Strings

3.4.1 Igualdad y comparación de Strings

Función	Descripción
fn:compare	Compara dos xs:string

La función fn:compare devuelve -1, 0 o 1 en función si el argumento que representa la primera cadena a comparar es menor, igual o mayor respectivamente que la segunda cadena.

3.4.2 Funciones para Strings

Función	Ejemplo	Descripción
fn:concat	fn:concat('abc', 'def') devuelve " abcdef " fn:concat('abc', 'def', 'ghi', 'jkl', 'mno') devuelve "abcdefghijklmno"	Concatena dos o mas cadenas
fn:string-join	fn:string-join(('Now', 'is', 'the', 'time', '...'), " ") devuelve "Now is the time ...". fn:string-join(("abra", "cadabra"), "") devuelve "abracadabra".	Devuelve una concatenación de las cadenas entradas como argumentos utilizando un separador opcional
fn:starts-with	fn:starts-with("goldenrod", "rod") devuelve false. fn:starts-with("goldenrod", "gold") devuelve true.	Comprueba si una cadena empieza con los caracteres indicados en otra cadena
fn:ends-with	fn:ends-with("goldenrod","rod") devuelve true. fn:ends-with("", "rod") devuelve false.	Comprueba si el valor de una cadena acaba con los caracteres de otra
fn:contains	fn:contains(“mi casa”, “cas”) devuelve true fn:contains(“goldenrod”,”rod”) devuelve true	Comprueba si una cadena de caracteres contiene otra
fn:substring	fn:substring("motor car", 6) devuelve " car". fn:substring("metadata", 4, 3) devuelve "ada".	Devuelve la sub-cadena de una cadena a partir de las indicaciones dadas
fn:string-length	fn:string-length("first we kill the lawyers") devuelve 25.	Devuelve la longitud de la cadena
fn:substring-before	fn:substring-before("abcdabcd","d") devuelve "abc". fn:substring-before("abcd","") devuelve "abcd"	Devuelve los caracteres de una cadena que se encuentran antes que otra sub-cadena
fn:substring-after	fn:substring-after("abcdabcd","d") devuelve "abcd". fn:substring-after("abcd","") devuleve "".	Devuelve los caracteres de una cadena que se encuentran a partir de una sub-cadena

Función	Ejemplo	Descripción
fn:normalize-space	fn:normalize-space(" hello world ") devuelve "hello world".	Devuelve la cadena de caracteres normalizada en cuanto a espacios en blanco
fn:normalize-unicode		Devuelve el valor normalizado del primer argumento en función del segundo argumento
fn:upper-case	fn:upper-case("abCd0") devuelve "ABCD0".	Devuelve la cadena de caracteres en mayúsculas
fn:lower-case	fn:lower-case("ABc!D") devuleve "abc!d".	Devuelve la cadena de caracteres en minúsculas
fn:translate	fn:translate("bar","abc","ABC") devuleve "BAr" fn:translate("abcdabc", "abc", "AB") devuelve "ABdAB"	Devuelve la cadena del primer argumento, traduciendo las ocurrencias definidas en el segundo argumento por el elemento indicado en el tercer elemento
fn:string-pad	fn:string-pad("XMLQuery", 2) devuleve "XMLQueryXMLQuery".	Devuelve una cadena formada por tantas copias de la cadena original como se ha especificado

3.4.3 Funciones de cadenas y patrones de concordancia

Función	Ejemplo	Descripción
fn:matches	fn:matches("abracadabra", "bra") devuelve true fn:matches("abracadabra", "^a.*a$") devuelve true fn:matches("abracadabra", "^bra") devuelve false	Devuelve true si el valor del primer argumento concuerda con la expresión regular indicada en el segundo argumento
fn:replace	replace("abracadabra", "bra", "*") devuelve "a*cada*" replace("abracadabra", "a.*a", "*") devuelve "*" replace("abracadabra", "a.*?a", "*") devuelve "*c*bra" replace("abracadabra", "a", "") devuelve "brcdbr"	Devuelve el valor del primer argumento con todas las sub-cadenas que se corresponden con el segundo argumento reemplazadas por el valor indicado en el tercer argumento
fn:tokenize	fn:tokenize("The cat sat on the mat", "\s+") devuelve ("The", "cat", "sat", "on", "the", "mat") fn:tokenize("Some unparsed HTML text", "\s* \s*", "i") devuelve ("Some unparsed", "HTML", "text")	Devuelve una secuencia de strings el valor de las cuales es el primer argumento separados por subcadenas que concuerdan con la expresión del segundo argumento

3.5 Funciones y operadores boleanos

Función	Descripción
fn:true	Función constructora del valor true
fn:false	Función constructora del valor false

Operador	Descripción
op:boolean-equal	Igualdad de valores booleanos
op:boolean-less-than	Comparación menor que entre operadores boleanos (false es menor que true)
op:boolean-greater-than	Comparación mayor que entre operadores boleanos (true es mayor que false)

Operador	Descripción
fn:not	Niega el argumento boleano

3.6 Funciones y Operadores de fecha y hora

3.6.1 Operadores de Comparación de Fecha/Hora

Operador	Ejemplo	Descripción
op:yearMonthDuration-equal		Comparación de igualdad entre los años y meses de dos fechas
op:yearMonthDuration-less-than		Comparación menor que entre fechas (solo mes y año)
op:yearMonthDuration-greater-than		Comparacion mayor que entre fechas (solo mes y año)
op:dayTimeDuration-equal		Comparación de igualdad entre días y horas (horas, minutos y segundos) de dos fechas
op:dayTimeDuration-less-than		Comparación menor que entre fechas (solo día y hora completa)
op:dayTimeDuration-greater-than		Comparación mayor que entre fechas (solo día y hora completa)

Operador	Ejemplo	Descripción
op:dateTime-equal	op:dateTime-equal(xs:dateTime("2002-04-02T12:00:00-01:00"), xs:dateTime("2002-04-02T17:00:00+04:00")) devuelve true. op:dateTime-equal(xs:dateTime("2002-04-02T12:00:00"), xs:dateTime("2002-04-02T23:00:00+06:00")) devuelve true.	Comparación de igualdad entre dos fechas completas (fecha completa y hora completa)
op:dateTime-less-than		Comparacion menor que entre dos fechas completas
op:dateTime-greater-than		Comparación mayor que entre dos fechas completas
op:date-equal		Comparacion de igualdad entre dos fechas (día, mes y año)
op:date-less-than		Comparación del tipo menor que entre dos fechas
op:date-greater-than		Comparación mayor que entre dos fechas
op:time-equal		Comparación de igualdad entre las horas (horas, minutos, segundos, décimas de segundos) de dos fechas
op:time-less-than	op:time-less-than(xs:time("12:00:00"), xs:time("23:00:00+06:00")) devuelve false. op:time-less-than(xs:time("11:00:00"), xs:time("17:00:00")) devuelve true.	Comparación del tipo menor que entre dos horas completas de dos fechas
op:time-greater-than		Comparación del tipo mayor que entre dos horas completas de dos fechas
op:gYearMonth-equal		Comparación de igualdad entre los años y meses de dos fechas
op:gYear-equal		Comparación de igualdad entre los años de dos fechas
op:gMonthDay-equal		Comparación de igualdad entre los meses y días de dos fechas
op:gMonth-equal		Comparación de igualdad entre los meses de dos fechas
op:gDay-equal		Comparación de igualdad entre los días de dos fechas

3.6.2 Funciones de extracción de componentes

Las siguientes funciones nos permiten extraer determinados componentes de fechas, horas y campos que indiquen duración

Función	Ejemplo	Descripción
fn:get-years-from-yearMonthDuration	fn:get-years-from-yearMonthDuration(xdt:yearMonthDuration("P20Y15M")) devuelve 21.	Devuelve el año de un valor del tipo `xdt:yearMonthDuration`
fn:get-months-from-yearMonthDuration	fn:get-months-from-yearMonthDuration(xdt:yearMonthDuration("P20Y15M")) devuelve 3.	Devuelve el componente mes de un valor `xdt:yearMonthDuration`
fn:get-days-from-dayTimeDuration	fn:get-days-from-dayTimeDuration(xdt:dayTimeDuration("P3DT10H")) devuelve 3. fn:get-days-from-dayTimeDuration(xdt:dayTimeDuration("P3DT55H")) devuelve 5.	Devuelve el componente días de un valor del tipo `xdt:dayTimeDuration`
fn:get-hours-from-dayTimeDuration	fn:get-hours-from-dayTimeDuration(xdt:dayTimeDuration("P3DT12H32M12S")) devuelve 12.	Devuelve las horas de un valor `xdt:dayTimeDuration`
fn:get-minutes-from-dayTimeDuration	fn:get-minutes-from-dayTimeDuration(xdt:dayTimeDuration("-P5DT12H30M")) devuelve -30.	Devuelve los minutos de un valor `xdt:dayTimeDuration`
fn:get-seconds-from-dayTimeDuration	fn:get-seconds-from-dayTimeDuration(xdt:dayTimeDuration("P3DT10H12.5S")) devuelve 12.5.	Devuelve los segundos de un valor `xdt:dayTimeDuration`
fn:get-year-from-dateTime	fn:get-year-from-dateTime(xs:dateTime("1999-05-31T13:20:00-05:00")) devuelve 1999. fn:get-year-from-dateTime(xs:dateTime("1999-12-31T19:20:00-05:00")) devuelve 2000	Devuelve el año de un valor `xs:dateTime`
fn:get-month-from-dateTime	fn:get-month-from-dateTime(xs:dateTime("1999-05-31T13:20:00-05:00")) devuelve 5. fn:get-month-from-dateTime(xs:dateTime("1999-12-31T19:20:00-05:00")) devuelve 1.	Devuelve el mes de un valor `xs:dateTime`
fn:get-day-from-dateTime	fn:get-day-from-dateTime(xs:dateTime("1999-05-31T13:20:00-05:00")) devuelve 31. fn:get-day-from-dateTime(xs:dateTime("1999-12-31T20:00:00-05:00")) devuelve 1.	Devuelve el día de un valor `xs:dateTime`
fn:get-hours-from-dateTime	fn:get-hours-from-dateTime(xs:dateTime("1999-05-31T08:20:00-05:00")) devuelve 13. fn:get-hours-from-dateTime(xs:dateTime("1999-12-31T21:20:00-05:00")) devuelve 2.	Devuelve las horas de un valor `xs:dateTime`
fn:get-minutes-from-dateTime	fn:get-minutes-from-dateTime(xs:dateTime("1999-05-31T13:20:00-05:00")) devuelve 20 . fn:get-minutes-from-dateTime(xs:dateTime("1999-05-31T13:30:00+05:30")) devuelve 0 .	Devuelve los minutos de un valor `xs:dateTime`

Función	Ejemplo	Descripción
fn:get-seconds-from-dateTime	fn:get-seconds-from-dateTime(xs:dateTime("1999-05-31T13:20:00-05:00")) devuelve 0.	Devuelve los segundos de un valor `xs:dateTime`
fn:get-timezone-from-dateTime	fn:get-timezone-from-dateTime(xs:dateTime("1999-05-31T13:20:00-05:00")) devuelve the xdt:dayTimeDuration whose value is -PT5H. fn:get-timezone-from-dateTime(xs:dateTime("2000-06-12T13:20:00Z")) devuelve the xdt:dayTimeDuration whose value is -PT0H.	devuelve la zona horaria de un valor `xs:dateTime`
fn:get-year-from-date	fn:get-year-from-date(xs:date("1999-05-31")) devuelve 1999. fn:get-year-from-date(xs:date("2000-01-01+05:00")) devuelve 1999.	Devuelve el año de un valor `xs:date`
fn:get-month-from-date	fn:get-month-from-date(xs:date("1999-05-31-05:00")) devuelve 5 . fn:get-month-from-date(xs:date("2000-01-01+05:00")) devuelve 12 .	Devuelve el mes de un valor `xs:date`
fn:get-day-from-date	fn:get-day-from-date(xs:date("1999-05-31-05:00")) devuelve 31. fn:get-day-from-date(xs:date("2000-01-01+05:00")) devuelve 31.	Devuelve el día de un valor `xs:date`
fn:get-timezone-from-date	fn:get-timezone-from-date(xs:date("1999-05-31-05:00")) devuelve the xdt:dayTimeDuration whose value is -PT5H. fn:get-timezone-from-date(xs:date("2000-06-12Z")) devuelve the xdt:dayTimeDuration with value PT0H.	Devuelve la zona horaria de un valor `xs:date`
fn:get-hours-from-time	fn:get-hours-from-time(xs:time("11:23:00")) devuelve 16. fn:get-hours-from-time(xs:time("21:23:00")) devuelve 2.	Devuelve las horas de un valor `xs:time`
fn:get-minutes-from-time	fn:get-minutes-from-time(xs:time("13:00:00Z")) devuelve 0 .	Devuelve los minutos de un valor `xs:time`
fn:get-seconds-from-time	fn:get-seconds-from-time(xs:time("13:20:10.5")) devuelve 10.5.	Devuelve los segundos de un valor `xs:time`
fn:get-timezone-from-time	fn:get-timezone-from-time(xs:time("13:20:00-05:00")) devuelve xdt:dayTimeDuration whose value is -PT5H.	Devuelve la zona horaria de un valor `xs:time`

3.6.3 Funciones aritméticas de fechas y horas

Función	Ejemplo	Descripción
fn:subtract-dateTimes-yielding-yearMonthDuration	fn:subtract-dateTimes-yielding-yearMonthDuration(xs:dateTime("2000-10-30T11:12:00"), xs:dateTime("1999-11-28T09:00:00")) devuelve a xdt:yearMonthDuration value corresponding to 11 months.	Devuelve la diferencia entre dos fechas indicada en meses
fn:subtract-dateTimes-yielding-dayTimeDuration	fn:subtract-dateTimes-yielding-dayTimeDuration(xs:dateTime("2000-10-30T06:12:00"), xs:dateTime("1999-11-28T09:00:00Z")) devuelve an xdt:dayTimeDuration value corresponding to 337 days, 2 hours and 12 minutes.	Devuelve la diferencia entre dos fechas expresada en días
op:subtract-dates	op:subtract-dates(xs:date("2000-10-30"), xs:date("1999-11-28")) devuelve an xdt:dayTimeDuration value corresponding to 337 days.	Devuelve la diferencia entre dos fechas expresada en días
op:subtract-times	op:subtract-times(xs:time("11:12:00Z"), xs:time("04:00:00")) devuelve an xdt:dayTimeDuration value corresponding to 2 hours and 12 minutes.	Devuelve la diferencia entre dos horas expresada en días
op:add-yearMonthDuration-to-dateTime	op:add-yearMonthDuration-to-dateTime(xs:dateTime("2000-10-30T11:12:00"), xdt:yearMonthDuration("P1Y2M")) devuelve an xs:dateTime value corresponding to the lexical representation "2001-12-30T11:12:00".	Devuelve el final de un periodo de tiempo añadiendo los meses indicados a la fecha inicial
op:add-dayTimeDuration-to-dateTime	op:add-dayTimeDuration-to-dateTime(xs:dateTime("2000-10-30T11:12:00"), xdt:dayTimeDuration("P3DT1H15M")) devuelve an xs:dateTime value corresponding to the lexical representation "2000-11-02T12:27:00".	Devuelve el final de un periodo de tiempo añadiendo un numero de días a la fecha inicial
op:subtract-yearMonthDuration-from-dateTime	op:subtract-yearMonthDuration-from-dateTime(xs:dateTime("2000-10-30T11:12:00"), xdt:yearMonthDuration("P1Y2M")) devuelve an xs:dateTime value corresponding to the lexical representation "1999-08-30T11:12:00".	Devuelve el principio de un periodo de tiempo resultado de restar a la fecha en que termina el periodo un determinado numero de meses
op:subtract-dayTimeDuration-from-dateTime	op:subtract-dayTimeDuration-from-dateTime(xs:dateTime("2000-10-30T11:12:00"), xdt:dayTimeDuration("P3DT1H15M")) devuelve an xs:dateTime value corresponding to the lexical representation "2000-10-27T09:57:00".	Devuelve el principio de un periodo de tiempo resultado de restar a la fecha en que termina el periodo un determinado numero de días

Función	Ejemplo	Descripción
op:add-yearMonthDuration-to-date	op:add-yearMonthDuration-to-date(xs:date("2000-10-30"), xdt:yearMonthDuration("P1Y2M")) devuelve the xs:date whose normalized value is December 30, 2001.	Devuleve el final de un periodo resultado de sumar un numero determinado de meses a la fecha en que comienza el periodo
op:add-dayTimeDuration-to-date	op:add-dayTimeDuration-to-date(xs:date("2000-10-30"), xdt:dayTimeDuration("P2DT2H30M0S")) devuelve the xs:date whose normalized value is November 1, 2000".	Devuleve el final de un periodo resultado de sumar un numero determinado de días a la fecha en que comienza el periodo
op:subtract-yearMonthDuration-from-date	op:subtract-yearMonthDuration-from-date(xs:date("2000-10-30"), xdt:yearMonthDuration("P1Y2M")) devuelve the xs:date whose normalized value is August 30, 1999.	Devuelve el la fecha de inicio de un periodo de tiempo resultado de restar a la fecha final un determinado numero de meses
op:subtract-dayTimeDuration-from-date	op:subtract-dayTimeDuration-from-date(xs:date("2000-10-30"), xdt:dayTimeDuration("P3DT1H15M")) devuelve a xs:date whose normalized value is October 26, 2000.	Devuelve el la fecha de inicio de un periodo de tiempo resultado de restar a la fecha final un determinado numero de días
op:add-dayTimeDuration-to-time	op:add-dayTimeDuration-to-time(xs:time("11:12:00"), xdt:dayTimeDuration("P3DT1H15M")) devuelve a normalized xs:time value corresponding to the lexical representation "12:27:00".	Añade a un elemento fecha un determinado numero de horas, minutos y segundos
op:subtract-dayTimeDuration-from-time	op:subtract-dayTimeDuration-from-time(xs:time("11:12:00"), xdt:dayTimeDuration("P3DT1H15M")) devuelve a normalized xs:time value corresponding to the lexical representation "09:57:00".	Resta a un elemento fe4cha una hora completa (horas, minutos y segundos)

3.7 Funciones y Operadores de Nodos

Función	Descripción
fn:name	Devuelve el nombre del nodo de contexto o el indicado como argumento
fn:namespace-uri	devuelve el namespace del nodo de contexto o el nodo indicado como argumento
fn:number	Devuelve el valor del nodo de contexto o el nodo especificado convertido a double
op:node-equal	Devuelve cierto si los dos nodos pasados como argumentos son iguales
op:node-before	Indica si un nodo aparece antes que otro en el documento
op:node-after	Indica si un nodo aparece después que otro en el documento
fn:root	Devuelve el nodo raíz del arbol al que el nodo indicado pertenece

3.8 Funciones y Operadores de Secuencias

3.8.1 Funciones y operadores de secuencias

Función	Descripción
fn:zero-or-one	Devuelve la secuencia de entrada si contiene como máximo un elemento
fn:one-or-more	Devuelve la secuencia de entrada si contiene como mínimo un elemento
fn:exactly-one	Devuelve la secuencia de entrada si contiene exactamente un elemento
fn:boolean	Convierte una cadena en boleano
op:concatenate	Concatena dos secuencias
fn:item-at	Devuelve el elemento de la secuencia indicado por el índice pasado como argumento
fn:index-of	Devuelve una secuencia de enteros, siendo estos los índices de posición donde el elemento pasado como primer argumento se encuentra
fn:empty	Indica si una secuencia está vacía
fn:exists	Indica si una secuencia tiene algún elemento como mínimo
fn:distinct-nodes	Devuelve una secuencia de elementos eliminando los nodos duplicados
fn:distinct-values	devuelve una secuencia de elementos eliminando los que tienen valores duplicados
fn:insert-before	Inserta un elemento o secuencia dentro de otra secuencia en la posición indicada
fn:remove	Borra el elemento de una secuencia indicando su posición
fn:subsequence	devuelve la sub-secuencia a partir de la indicación de su localización
fn:unordered	Indica si la secuencia obtenida se debe devolver en un determinado orden

3.8.2 Funciones de union, intersección y diferencia de secuencias

Función	Descripción
fn:deep-equal	Devuelve cierto si los dos argumentos dados tiene el mismo elemento en sus posiciones correspondientes
fn:sequence-node-identical	Devuelve cierto si los dos argumentos tienen nodos idénticos en las posiciones indicadas
op:union	devuelve la union de dos secuencias, pasadas como argumentos, eliminando los duplicados
op:intersect	Devuelve la intersección de dos secuencias eliminando los duplicados
op:except	Devuelve la diferencia de dos secuencias eliminando los duplicados

3.8.3 Funciones de agregados

Función	Descripción
fn:count	Devuelve el numero de elementos de una secuencia
fn:avg	Devuelve el promedio de los valores de los elementos de una secuencia
fn:max	Devuelve el elemento con el valor máximo de una secuencia
fn:min	Devuelve el elemento de una secuencia con el mínimo valor
fn:sum	Devuelve la suma de los valores de una secuencia

Tratamiento de XML sobre Oracle 9i

1 Introducción

Cada vez mas negocios se están implantando a través de Internet, y el requisito para el intercambio de datos sin la intervención humana se convierte en uno de los requisitos primordiales de este tipo de negocios.
La integración de XML con las bases de datos de las empresas y las aplicaciones servidoras permiten dar a este tipo de negocios la infraestructura para satisfacer la importante demanda que actualmente se está dando para el acceso e intercambio de esta información.

Los negocios actuales dirigidos a ofrecer servicios electrónicos a través de la web necesitan compartir información de la propia empresa con las demás empresas con las que se relaciona sin tener que modificar los sistemas corporativos actuales con los que la empresa trabaja.

Ya se ha comentado anteriormente los problemas principales que se dan a la hora de conectar dos sistemas diferentes permitiendo la integración de los datos; entre otros encontramos que debe darse la misma infraestructura en ambas empresas, es de gran complejidad el proceso de intercambio de información y hay una enorme dificultad de usar los protocolos de Internet.

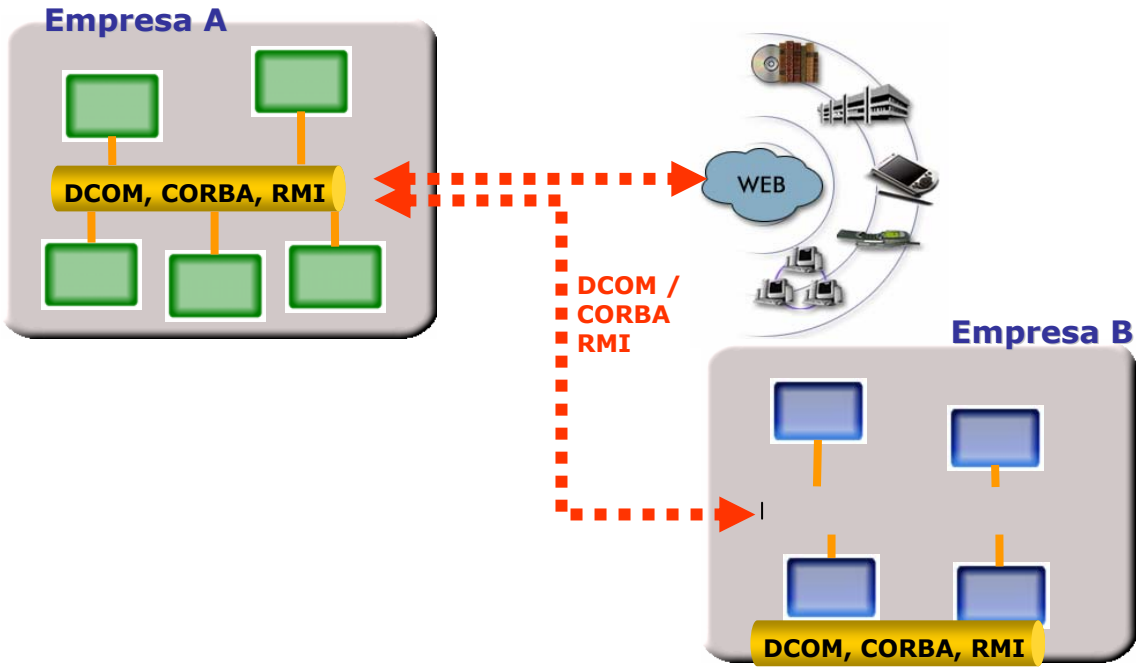


Figura 4.1. Intercambio de información vía web

En definitiva, muchas veces ocurre que en aplicaciones Business to Business (B2B) son las propias personas de las empresas las que integran la información a intercambiar en los propios sistemas.
Los sistemas de gestión de contenidos basados en el lenguaje XML permiten satisfacer las necesidades comerciales de este tipo de empresas. XML es en la actualidad el estándar más aceptado para el intercambio de datos. XML es el estándar idóneo para este tipo de negocios ya que solventa los problemas descritos anteriormente: las propias aplicaciones entienden y pueden decocidifcar el lenguaje XML, no se necesita cambiar de sistemas de origen en las respectivas empresas y sobretodo, por que soporta los protocolos de Internet.

XML ha emergido como el estándar de la industria para describir los datos de negocio para aplicaciones de integración de datos Business to Business (B2B) en Internet ya que es la forma más simple, estándar y no propietaria (la misma información puede ser usada a través de múltiple dispositivos como navegadores, teléfonos móviles, PDA's,…) de compartir información

Es por estos motivos que el número de bases de datos, tanto relacionales como nativas que soportan XML como fuente de datos es cada vez mayor.

En los últimos años las bases de datos han sido el principal elemento de apoyo en los negocios de cualquier tipo y el teclado el elemento primordial de entrada de datos. Principalmente esto se debía al hecho que los sistemas que se relacionaban para efectuar las diferentes transacciones eran homogéneos entre sí y el intercambio de datos entre ellos era más o menos sencillo. En los últimos años, XML juntamente con Internet y la aparición de multitud de sistemas heterogéneos han sido los elementos que han revolucionado este sistema de negocio. Actualmente no se puede esperar que las empresas que realizan negocios a través de la red dispongan de la misma infraestructura.

2 Oracle 9i y XML

Aunque Oracle no formaba parte inicialmente de los grupos de trabajo XML del W3C, la gran aceptación que empezó a tener este estándar hizo que Oracle tuviese que implicarse en los organismos de normalización de XML así como se involucró en tareas para impulsar el estándar. Actualmente Oracle forma parte de los siguientes grupos de trabajo del W3C:

- Grupo de trabajo XML CORE responsable de especificar las tecnologías XML
- Grupo de trabajo XLS responsable de definir el lenguaje ampliado de hojas de estilo en sus dos implementaciones actuales XSLT y XSLFO
- Grupo de trabajos de esquemas XML responsable de definir la sintaxis y utilización de tipos de datos dentro de los documentos XML
- Grupo de trabajo de enlaces XML responsable de definir como se referencian los recursos XML desde dentro de los documentos XML
- Grupo de trabajo DOM responsable de la API DOM como método de acceso a documentos XML
- Grupo de trabajo de consultas XML responsable de la especificación de la gramática y sintaxis de los lenguajes destinados a realizar consultas sobre documentos XML

Oracle 9i cuenta con un paquete de herramientas de desarrollo XML, este paquete es conocido por XDK o XML Developer's Kit. El XDK de Oracle utiliza aplicaciones Java, aplicaciones desarrolladas en C y módulos desarrollados en PL/SQL.
El XDK de Oracle soporta los siguientes estándares:

- XML 1.0
- XML Schema 1.0
- XML Namespace 1.0
- XSLT 1.0
- XPath 1.0
- DOM 1.0 y DOM 2.0
- SAX 1.0 y SAX 2.0

Por su parte el XDK para Java cuenta con los siguientes componentes:

- XML Parser for Java V2
- XML Schema Processor for Java
- XML Class Generator for Java
- XSQL Servlet
- XML SQL Utility

2.1 Analizador XML de Oracle

Cualquier producto que trata con documentos XML debe poder determinar si el documento XML es válido o no, lo que significa si concuerda con el archivo DTD que define ese documento y si el documento es válido o no. (Véase El lenguaje de marcas XML. 3 DTD)
Estas dos operaciones se llevan a cabo a través de un analizador XML.
El analizador XML de Oracle, juntamente con el procesador incorporado XSLT está codificado en Java, C/C++ y PL/SQL. Este analizador se ajusta a la especificación XML 1.0 y puede ejecutarse tanto en aplicaciones externas a la base de datos como en aplicaciones dentro de la base de datos utilizando la Java Virtual Machina de Oracle 9i.

2.1.1 Instalación del analizador XML de Oracle

El analizador XML de Oracle es una colección de clases Java contenidas en un archivo .jar llamado xmlparserv2.jar que se encuentran en la carpeta lib de la instalación (C:\ORACLE\ORA92\LIB).
Para instalarse tenemos dos formas; el primer método es incluir el archivo xmlparserv2.jar en la variable classpath para que el compilador de java que estemos utilizando pueda utilizar las clases incluidas en este archivo. El segundo método consiste en utilizar el comando loadjava (C:\ORACLE\ORA92\BIN) para recuperar el paquete xmlparserv2.jar y que éste se aloje en la base de datos como parte de la base de datos Oracle 9i funcionando mediante la Java VM de Oracle.

La sintaxis de la instrucción loadjava es:

```
% loadjava -user scott/tiger -r -v xmlparserv2.jar
```

```
% loadjava -user scott/tiger -r -v xmlplsql.jar
```

En nuestro caso concreto y teniendo en cuenta que hemos creado un usuario llamado xml y con contraseña xml para la realización de los ejemplos prácticos sobre Oracle, tenemos que ejecutamos la siguiente instrucción:

```
c:\oracle\ora92\bin> loadjava     -user xml/xml    -r –v     xmlparserv2.jar
```

Con lo que cargamos el analizador XML

y

```
c:\oracle\ora92\bin> loadjava     -user xml/xml    -r –v     xmlplsql.jar
```

Con lo que cargamos en la base de datos el entorno del analizador XML para PL/SQL.

2.1.2 Ejecución del analizador XML de Oracle fuera de la base de datos

El analizador XML de Oracle, como ya se ha comentado, está implementado en lenguaje Java, con lo que puede utilizarse fuera de la base de datos. Además, a la hora de utilizar el analizador podemos escoger entre usar la API DOM y la API SAX para analizar un documento XML.

Para comprobar el funcionamiento del analizador XML de Oracle y todos las diferentes pruebas posteriores que se ira desarrollando vamos a trabajar sobre un archivo llamado bib.xml que representa un documento xml en el que se almacenan diferentes libros. El contenido de este documento es:

```
<bib>
   <book year="1994">
      <title>TCP/IP Illustrated</title>
      <isbn>123-1345-44</isbn>
      <author><last>Stevens</last><first>W.</first></author>
      <publisher>Addison-Wesley</publisher>
      <price> 65.95</price>
   </book>

   <book year="1992">
      <title>Advanced Programming in the Unix environment</title>
      <isbn>12-1345-XX</isbn>
      <author><last>Stevens</last><first>W.</first></author>
      <publisher>Addison-Wesley</publisher>
      <price>65.95</price>
   </book>

   <book year="2000">
      <title>Data on the Web</title>
      <isbn>34-13445-00</isbn>
      <author><last>Abiteboul</last><first>Serge</first></author>
      <author><last>Buneman</last><first>Peter</first></author>
      <author><last>Suciu</last><first>Dan</first></author>
      <publisher>Morgan Kaufmann Publishers</publisher>
      <price>39.95</price>
   </book>

   <book year="1999">
      <title>The Economics of Technology and Content for Digital TV</title>
      <isbn>01-00001-01</isbn>
      <editor>
            <last>Gerbarg</last><first>Darcy</first>
             <affiliation>CITI</affiliation>
      </editor>
         <publisher>Kluwer Academic Publishers</publisher>
      <price>129.95</price>
   </book>

  <book year="2002">
      <title>Oracle 9i.Desarrollo Web</title>
      <isbn>84-415-1406-2</isbn>
      <author><last>Brown</last><first>Bradley</first></author>
      <publisher>Anaya Multimedia</publisher>
      <price>54.00</price>
   </book>


</bib>
```

Observamos que este documento XML contiene información sobre libros, los que tienen como atributo el año de publicación y como componentes del elemento libro, el titulo, el isbn, el autor o autores del libro o bien el editor del libro, también se incluye la información referente a la editorial así como el precio.
El archivo DTD que corresponde a este documento XML será bib.dtd:

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

<!DOCTYPE bib >
<!ELEMENT bib  (book* )>
<!ELEMENT book  (title,  isbn, (author+ | editor+ ), publisher, price )>
<!ATTLIST book  year CDATA  #REQUIRED >
<!ELEMENT isbn  (#PCDATA )>
<!ELEMENT author  (last, first )>
<!ELEMENT editor  (last, first, affiliation )>
<!ELEMENT title  (#PCDATA )>
<!ELEMENT last  (#PCDATA )>
<!ELEMENT first  (#PCDATA )>
<!ELEMENT affiliation  (#PCDATA )>
<!ELEMENT publisher  (#PCDATA )>
<!ELEMENT price  (#PCDATA )>
```

Analizador XML utilizando DOM

La siguiente aplicación Java lee el documento bib.xml externo (que se recoge como parámetro) y muestra sus elementos, atributos y valores utilizando la API DOM

```
// DOMDemoTraverse.java
import java.io.*;
import org.w3c.dom.*;
import org.w3c.dom.Node;
import oracle.xml.parser.v2.DOMParser;
import oracle.xml.parser.v2.XMLElement;
import oracle.xml.parser.v2.XMLConstants;
public class DOMDemoTraverse {
  //Document Object Model Parser
   public static void main(String[] args){
      try{
        if (args.length != 1){
          // Verify that an XML file was passed
          System.err.println("XML source file expected.") ;
          System.exit(1);
        }
        // Get an instance of the parser
        DOMParser parser = new DOMParser();
        // Create an input stream from the filename.
       FileInputStream  xmlInputStream
              = new FileInputStream(new File(args[0]));
        // Set parser options: validation on,
        // warnings shown, error stream set to stderr.
        parser.setErrorStream(System.err);
        parser.setValidationMode(XMLConstants.NONVALIDATING);
        parser.showWarnings(true);
        // Parse the document.
        parser.parse(xmlInputStream);
        // Get the document.
        Document doc = parser.getDocument();
        // Display document elements
        System.out.println("Document Element, Attributes, " +
                           "and values are: ");
        printElements(doc);
```

```
      // Display document element attributes
      System.out.println("Parsing Complete. ");
    }catch (Exception e){
      System.out.println("Error: "+e.toString());
    }
  }
  public static void printElements(Document doc){
    try{
     //Get Root Element
     displayElements(doc.getDocumentElement());
    }catch(Exception e){
     System.err.println("Error printElements: "+e.getMessage());
    }
  }
   public static void displayElements(Node node){
    Node          childNode;
    NodeList      nodeChildren;
    if(node.getNodeType() == Node.ELEMENT_NODE){
      System.out.println("Element Name= "+node.getNodeName());
      //if (node.hasAttributes()){
      if (node.getAttributes().getLength() > 0){
        displayAttributes(node.getAttributes());
      }
      if (node.hasChildNodes()){
        if (node.getFirstChild().getNodeType()==Node.TEXT_NODE){
          System.out.println("Element value= "+
          node.getFirstChild().getNodeValue());
          System.out.println("");
        }
        nodeChildren=node.getChildNodes();
        for (int i=0; i<nodeChildren.getLength();i++){
          //Recursive call to displayElements method
          displayElements(nodeChildren.item(i));
         }
       }
     }
   }
   public static void displayAttributes(NamedNodeMap attNodes){
     Node node;
     System.out.println("Attribute List");
     for (int i=0; i < attNodes.getLength(); i++){
       node = attNodes.item(i);
       System.out.println("Attibute  Name  = "+node.getNodeName());
       System.out.println("Attribute Value = "+node.getNodeValue());
     }
     System.out.println("");
   }
}
```

El resultado del programa anterior después de ejecutarlo sobre bib.xml será el siguiente:

```
Document Element, Attributes, and values are:
Element Name= bib
Element Name= book
Attribute List
Attibute  Name  = year
Attribute Value = 1994

Element Name= title
Element value= TCP/IP Illustrated

Element Name= isbn
Element value= 123-1345-44

Element Name= author
Element Name= last
Element value= Stevens

Element Name= first
Element value= W.

Element Name= publisher
```

```
Element value= Addison-Wesley

Element Name= price
Element value=  65.95

Element Name= book
Attribute List
Attibute  Name  = year
Attribute Value = 1992

Element Name= title
Element value= Advanced Programming in the Unix environment

Element Name= isbn
Element value= 12-1345-XX

Element Name= author
Element Name= last
Element value= Stevens

Element Name= first
Element value= W.

Element Name= publisher
Element value= Addison-Wesley

Element Name= price
Element value= 65.95

Element Name= book
Attribute List
Attibute  Name  = year
Attribute Value = 2000

Element Name= title
Element value= Data on the Web

Element Name= isbn
Element value= 34-13445-00

Element Name= author
Element Name= last
Element value= Abiteboul

Element Name= first
Element value= Serge

Element Name= author
Element Name= last
Element value= Buneman

Element Name= first
Element value= Peter

Element Name= author
Element Name= last
Element value= Suciu

Element Name= first
Element value= Dan

Element Name= publisher
Element value= Morgan Kaufmann Publishers

Element Name= price
Element value= 39.95

Element Name= book
Attribute List
Attibute  Name  = year
Attribute Value = 1999

Element Name= title
Element value= The Economics of Technology and Content for Digital TV

Element Name= isbn
Element value= 01-00001-01

Element Name= editor
Element Name= last
Element value= Gerbarg

Element Name= first
Element value= Darcy

Element Name= affiliation
Element value= CITI

Element Name= publisher
Element value= Kluwer Academic Publishers

Element Name= price
Element value= 129.95

Element Name= book
Attribute List
Attibute  Name  = year
Attribute Value = 2002

Element Name= title
Element value= Oracle 9i.Desarrollo Web

Element Name= isbn
Element value= 84-415-1406-2

Element Name= author
Element Name= last
Element value= Brown

Element Name= first
Element value= Bradley

Element Name= publisher
Element value= Anaya Multimedia

Element Name= price
Element value= 54.00

Parsing Complete.
```

Analizador XML utilizando SAX

La siguiente aplicación Java usa la especificación SAX 1.0 utilizando la clase HandlerBase que es el controlador de contenido por defecto de SAX 1.0. Al igual que el ejemplo anterior esta aplicación lee el archivo bib.xml recogido como argumento de la aplicación y devuelve los elementos y sus atributos y valores:

```
// SAXDemoTraverse.java
import org.xml.sax.*;
import java.io.*;
import java.net.*;
import oracle.xml.parser.v2.*;

public class SAXDemoTraverse extends HandlerBase{
  // Store the locator
  Locator locator;
  public static void main(String[] args){
    try{
      if (args.length != 1){
        // Make sure name of XML file was entered.
        System.err.println("Usage: saxXMLParser filename");
        System.exit(1);
      }
      // Create a new handler for the parser
      SAXDemoTraverse saxHandler = new SAXDemoTraverse();
      // Get an instance of the parser
      Parser saxparser = new SAXParser();
      // Set Handlers in the parser
      saxparser.setDocumentHandler(saxHandler);
      saxparser.setEntityResolver(saxHandler);
      saxparser.setErrorHandler(saxHandler);
      try{
        // Create an input stream from the filename.
        FileInputStream  xmlInputStream =
            new FileInputStream(new File(args[0]));
        saxparser.parse(new InputSource(xmlInputStream));
      }catch (SAXParseException e){
        System.out.println(e.getMessage());
      }catch (SAXException e){
        System.out.println(e.getMessage());
      }
    }catch (Exception e){
      System.out.println(e.toString());
    }
  }
  // Implementation of DocumentHandler interface.
  public void setDocumentLocator (Locator locator){
    System.out.println("Set Document Locator:");
    this.locator = locator;
  }
  public void startDocument(){
    System.out.println("Start Document");
  }
  public void endDocument() throws SAXException{
    System.out.println("End Document");
  }
  public void startElement(String name, AttributeList
      attributes) throws SAXException{
    System.out.println("Start Element:"+name);
    for (int i=0;i<attributes.getLength();i++){
      String attrname = attributes.getName(i);
      String type     = attributes.getType(i);
      String value    = attributes.getValue(i);
```

```
        System.out.println("");
        System.out.println("Attribute: "+attrname+"="+value);
      }
      System.out.println("");
    }
    public void endElement(String name) throws SAXException{
      System.out.println("");
      System.out.println("End Element:"+name);
      System.out.println("");
    }
    public void characters(char[] cbuf, int start, int len){
      System.out.print("Characters: ");
      System.out.println(new String(cbuf,start,len));
    }
    public void processingInstruction(String target, String data)
            throws SAXException{
      System.out.println("ProcessingInstruction:"+target+" "+data);
    }
    // Implementation of the EntityResolver interface.
    public InputSource resolveEntity (String publicId, String
       systemId)throws SAXException{
      System.out.println("ResolveEntity:"+publicId+" "+systemId);
      System.out.println("Locator:"+locator.getPublicId()+" "+
                locator.getSystemId()+
                " "+locator.getLineNumber()+
                " "+locator.getColumnNumber());
      return null;
    }
    // Implementation of the DTDHandler interface.
    public void notationDecl (String name, String publicId, String
      systemId){
      System.out.println("NotationDecl:"+name+" "+publicId+
      " "+systemId);
    }
    public void unparsedEntityDecl (String name, String publicId,
                          String systemId, String notationName){
      System.out.println("UnparsedEntityDecl:"+name +
      " "+publicId+" "+systemId+" "+notationName);
    }
    // Implementation of the ErrorHandler interface.
    public void warning (SAXParseException e)
            throws SAXException{
      System.out.println("Warning:"+e.getMessage());
    }
    public void error (SAXParseException e)
            throws SAXException{
      throw new SAXException(e.getMessage());
    }
    public void fatalError (SAXParseException e)
            throws SAXException{
      System.out.println("Fatal error");
      throw new SAXException(e.getMessage());
    }
}
```

El resultado obtenido será:

```
Set Document Locator:
Start Document
Start Element:bib

Start Element:book
```

```
Attribute: year=1994

Start Element:title

Characters: TCP/IP Illustrated
```

```
End Element:title
Start Element:isbn
Characters: 123-1345-44
End Element:isbn
Start Element:author
Start Element:last
Characters: Stevens
End Element:last
Start Element:first
Characters: W.
End Element:first

End Element:author
Start Element:publisher
Characters: Addison-Wesley
End Element:publisher
Start Element:price
Characters:
Characters: 65.95
End Element:price

End Element:book
Start Element:book
Attribute: year=1992
Start Element:title
Characters: Advanced Programming in the
Unix environment
End Element:title
Start Element:isbn
Characters: 12-1345-XX
End Element:isbn
Start Element:author
Start Element:last
Characters: Stevens
```

```
End Element:last
Start Element:first
Characters: W.
End Element:first

End Element:author
Start Element:publisher
Characters: Addison-Wesley
End Element:publisher
Start Element:price
Characters: 65.95
End Element:price

End Element:book
Start Element:book
Attribute: year=2000
Start Element:title
Characters: Data on the Web
End Element:title
Start Element:isbn
Characters: 34-13445-00
End Element:isbn
Start Element:author
Start Element:last
Characters: Abiteboul
End Element:last
Start Element:first
Characters: Serge
End Element:first

End Element:author
Start Element:author
Start Element:last
Characters: Buneman
```

End Element:last

Start Element:first

Characters: Peter

End Element:first

End Element:author

Start Element:author

Start Element:last

Characters: Suciu

End Element:last

Start Element:first

Characters: Dan

End Element:first

End Element:author

Start Element:publisher

Characters: Morgan Kaufmann Publishers

End Element:publisher

Start Element:price

Characters: 39.95

End Element:price

End Element:book

Start Element:book

Attribute: year=1999

Start Element:title

Characters: The Economics of Technology and Content for Digital TV

End Element:title

Start Element:isbn

Characters: 01-00001-01

End Element:isbn

Start Element:editor

Start Element:last

Characters: Gerbarg

End Element:last

Start Element:first

Characters: Darcy

End Element:first

Start Element:affiliation

Characters: CITI

End Element:affiliation

End Element:editor

Start Element:publisher

Characters: Kluwer Academic Publishers

End Element:publisher

Start Element:price

Characters: 129.95

End Element:price

End Element:book

Start Element:book

Attribute: year=2002

Start Element:title

Characters: Oracle 9i.Desarrollo Web

End Element:title

Start Element:isbn

Characters: 84-415-1406-2

End Element:isbn

Start Element:author

Start Element:last

Characters: Brown

End Element:last

Start Element:first

Characters: Bradley

End Element:first

```
End Element:author
Start Element:publisher
Characters: Anaya Multimedia
End Element:publisher
Start Element:price
Characters: 54.00
End Element:price
End Element:book
End Element:bib
End Document
```

Veamos ahora un ejemplo más práctico utilizando el analizador SAX 2.0 y comprobando como este analizador implementa la gestión de eventos y algunos callbacks. La aplicación consiste en que a partir de unos datos dados en formato XML (se dan por comodidad dentro de una variable String pero se sobreentiende que pueden haber sido recogidos de cualquier forma) se insertan dos registros en la mítica tabla emp dentro del perfil del usuario xml.

```
// InsertSAXEmp.java
import oracle.xml.parser.v2.SAXParser;
import org.xml.sax.*;
import org.xml.sax.helpers.*;
import java.io.*;
import java.sql.*;
public class InsertSAXEmp extends DefaultHandler{
   private Locator locator;
   private static Connection conn;
   private String vElementData  = null;
   private String vEMPNO        = new String();
   private String vENAME        = new String();
   private String vJOB          = new String();
   private String vMGR          = new String();
   private String vHIREDATE     = new String();
   private String vSAL          = new String();
   private String vDEPTNO       = new String();
   public static String xmlData = new String("<ROWSET>"+
                                 "<ROW num='1'>"+
                                 "<EMPNO>1000</EMPNO>"+
                                 "<ENAME>MARIN</ENAME>"+
                                 "<JOB>MANAGER</JOB>"+
                                 "<MGR>7902</MGR>"+
                                 "<HIREDATE>12-04-2001</HIREDATE>"+
                                 "<SAL>8000</SAL>"+
                                 "<DEPTNO>20</DEPTNO>"+
                                 "</ROW>"+
                                  "<ROW num='2'>"+
                                 "<EMPNO>9367</EMPNO>"+
                                 "<ENAME>SMITH</ENAME>"+
                                 "<JOB>CLERK</JOB>"+
                                 "<MGR>7902</MGR>"+
                                 "<HIREDATE>11-04-2001</HIREDATE>"+
                                 "<SAL>800</SAL>"+
                                 "<DEPTNO>20</DEPTNO>"+
                                 "</ROW>"+
                                 "</ROWSET>");
   public InsertSAXEmp()
        {
           super();
        }
   public static void main(String arg[]){
     try{
```

```
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
       conn = DriverManager.getConnection("jdbc:oracle:oci8:@","xml","xml");
      // Create a new default handler for the parser
      DefaultHandler saxEmpInsert = new InsertSAXEmp();
      // Get an instance of the parser
      SAXParser parser = new SAXParser();
      // Set Handlers in the parser
      ((SAXParser)parser).setContentHandler(saxEmpInsert);
      ((SAXParser)parser).setEntityResolver(saxEmpInsert);
      ((SAXParser)parser).setErrorHandler(saxEmpInsert);
      try
      {
        StringReader is = new StringReader(xmlData);
        InputSource xmlDataIS = new InputSource(is);
        ((SAXParser)parser).parse(xmlDataIS);
      }
      catch (SAXParseException e)
      {
         System.out.println("Parse Error "+e.getMessage());
      }
      catch (SAXException e)
      {
         System.out.println(e.getMessage());
      }
    }
    catch (Exception e)
    {
      System.out.println(e.toString());
    }
}
public void setDocumentLocator(Locator locator){
  this.locator = locator;
}
public void endElement(String namespaceURI,
                       String name,
                       String rawName) throws SAXException
{
    if (name.equals("EMPNO")){
        vEMPNO    = vElementData;
    }else if(name.equals("ENAME")){
        vENAME    = vElementData;
    }else if (name.equals("JOB")){
        vJOB      = vElementData;
    }else if (name.equals("MGR")){
        vMGR      = vElementData;
    }else if (name.equals("HIREDATE")){
        vHIREDATE = vElementData;
    }else if (name.equals("SAL")){
        vSAL      = vElementData;
    }else if (name.equals("DEPTNO")){
        vDEPTNO   = vElementData;
    }else if (name.equals("ROW")){
        insertEmpRecord();
    }
    vElementData = null;
}
public void characters(char[] cbuf, int start, int len)
{
    vElementData = new String(cbuf,start,len);
}
public void insertEmpRecord(){
 try{
    PreparedStatement stmt = conn.prepareStatement(
```

```
      "insert                 into                 EMP(EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,DEPTNO)
VALUES(?,?,?,?,?,?,?,?)");
     stmt.setInt(1,Integer.parseInt(vEMPNO));
     stmt.setString(2,vENAME);
     stmt.setString(3,vJOB);
     stmt.setString(4,vMGR);
     stmt.setString(5,vHIREDATE);
     stmt.setFloat(6,Float.parseFloat(vSAL));
     stmt.setInt(7,Integer.parseInt(vDEPTNO));
     stmt.executeUpdate();
     conn.commit();
   }catch(Exception e){
     System.out.println(e.getMessage());
   }
  }
}
```

Si dentro del SQLPlus de Oracle 9i realizamos una consulta sobre la tabla emp, tenemos el siguiente resultado:

```
SQL> select * from emp;

    EMPNO ENAME      JOB              MGR HIREDATE        SAL      COMM     DEPTNO
---------- ---------- --------- ---------- -------- ---------- ---------- ----------
     7369 SMITH      CLERK          7902 17/12/80        800                    20
     7499 ALLEN      SALESMAN       7698 20/02/81       1600        300         30
     7521 WARD       SALESMAN       7698 22/02/81       1250        500         30
     7566 JONES      MANAGER        7839 02/04/81       2975                    20
     7654 MARTIN     SALESMAN       7698 28/09/81       1250       1400         30
     7698 BLAKE      MANAGER        7839 01/05/81       2850                    30
     7782 CLARK      MANAGER        7839 09/06/81       2450                    10
     7788 SCOTT      ANALYST        7566 19/04/87       3000                    20
     7839 KING       PRESIDENT           17/11/81       5000                    10
     7844 TURNER     SALESMAN       7698 08/09/81       1500          0         30
     7876 ADAMS      CLERK          7788 23/05/87       1100                    20
     7900 JAMES      CLERK          7698 03/12/81        950                    30
     7902 FORD       ANALYST        7566 03/12/81       3000                    20
     7934 MILLER     CLERK          7782 23/01/82       1300                    10
     1000 MARIN      MANAGER        7902 12/04/01       8000                    20
     9367 SMITH      CLERK          7902 11/04/01        800                    20


17 filas seleccionadas.
```

2.1.3 Ejecución del analizador XML de Oracle desde el interior de la base de datos

Como ya se ha comentado anteriormente el analizador XML de Oracle puede ejecutarse desde el interior de la base de datos usando la máquina virtual Java de Oracle 9i. Los métodos del analizador deben estar desarrollados siguiendo el entorno de "programación" PL/SQL de Oracle; actualmente Oracle ofrece 16 paquetes PL/SQL llamados Oracle XML Parser for PL/SQL, cuyos procedimientos y funciones asocian los métodos Java del analizador XML.
El siguiente ejemplo muestra como se utiliza el analizador XML mediante procedimientos y funciones PL/SQL, al igual que los dos ejemplos anteriores, este procedimiento PL/SQL lee un documento XML externo desde el sistema operativo, lo analiza y muestra los elementos y atributos del documento analizado.

```
CREATE OR REPLACE PROCEDURE PLSQLXML(directory VARCHAR2,
                                inputFile VARCHAR2,
                                errorFile VARCHAR2) is
  parser    xmlparser.Parser;
  document  xmldom.DOMDocument;

  PROCEDURE printElements(document xmldom.DOMDocument) is
  nodeList   xmldom.DOMNodeList;
  listLength NUMBER;
  node       xmldom.DOMNode;
  BEGIN
    nodeList   := xmldom.getElementsByTagName(document, '*');
    listLength := xmldom.getLength(nodeList);
    FOR elements IN 0..listLength-1 LOOP
      node := xmldom.item(nodeList, elements);
      DBMS_OUTPUT.PUT(xmldom.getNodeName(node) || ' ');
    END LOOP;
    DBMS_OUTPUT.PUT_LINE('');
  END;
  PROCEDURE printElementAttributeValues(document xmldom.DOMDocument) IS
    nodeList        xmldom.DOMNodeList;
    lengthList      NUMBER;
    lengthChild     NUMBER;
    lengthAttr      NUMBER;
    node            xmldom.DOMNode;
    childNode       xmldom.DOMNode;
    childNodeList   xmldom.DOMNodeList;
    nodeElement     xmldom.DOMElement;
    data            xmldom.DOMCharacterData;
    namedNodeMap    xmldom.DOMNamedNodeMap;
    attrname        VARCHAR2(100);
    attrval         VARCHAR2(100);
  BEGIN
     nodeList   := xmldom.getElementsByTagName(document, '*');
     lengthList := xmldom.getLength(nodelist);
     FOR element IN 0..lengthList-1 LOOP
       node         := xmldom.item(nodeList, element);
       nodeElement  := xmldom.makeElement(node);
       dbms_output.put(xmldom.getTagName(nodeElement) || ':
           ');
       childNodeList :=
       xmldom.getChildNodes(xmldom.makeNode(nodeElement));
      IF(xmldom.isNull(childNodeList)=FALSE) THEN
        lengthChild := xmldom.getLength(childNodeList);
        FOR childElement IN 0..lengthChild-1 LOOP
          childNode :=
            xmldom.item(childNodeList,childElement);
          IF xmldom.getNodeValue(childNode) IS NOT NULL THEN
            dbms_output.put_line(xmldom.getNodeValue(childNode));
          END IF;
        END LOOP;
       END IF;
       namedNodeMap := xmldom.getAttributes(node);
      IF (xmldom.isNull(namedNodeMap) = FALSE) THEN
        lengthAttr := xmldom.getLength(namedNodeMap);
        FOR attribute in 0..lengthAttr-1 LOOP
          node := xmldom.item(namedNodeMap, attribute);
          attrname := xmldom.getNodeName(node);
          attrval := xmldom.getNodeValue(node);
          dbms_output.put(' ' || attrname || ' = ' || attrval);
        END LOOP;
        dbms_output.put_line('');
      END IF;
     END LOOP;
```

```
  END;
  BEGIN
    parser := xmlparser.newParser;
    xmlparser.setValidationMode(parser, FALSE);
    xmlparser.setErrorLog(parser, directory || '\' || errorfile);
    xmlparser.setBaseDir(parser, directory);
    xmlparser.parse(parser, directory || '\' || inputFile);
    document := xmlparser.getDocument(parser);
    dbms_output.put('The elements are: ');
    printElements(document);
    dbms_output.put_line('The attributes of each element are:');
    printElementAttributeValues(document);
  EXCEPTION
    WHEN xmldom.INDEX_SIZE_ERR THEN
      raise_application_error(-20120, 'Index Size error');
    WHEN xmldom.DOMSTRING_SIZE_ERR THEN
      raise_application_error(-20120, 'String Size error');
    WHEN xmldom.HIERARCHY_REQUEST_ERR THEN
      raise_application_error(-20120, 'Hierarchy request error');
    WHEN xmldom.WRONG_DOCUMENT_ERR THEN
      raise_application_error(-20120, 'Wrong doc error');
    WHEN xmldom.INVALID_CHARACTER_ERR THEN
      raise_application_error(-20120, 'Invalid Char error');
    WHEN xmldom.NO_DATA_ALLOWED_ERR THEN
     raise_application_error(-20120, 'No data allowed error');
    WHEN xmldom.NO_MODIFICATION_ALLOWED_ERR THEN
     raise_application_error(-20120, 'No mod allowed error');
    WHEN xmldom.NOT_FOUND_ERR THEN
     raise_application_error(-20120, 'Not found error');
    WHEN xmldom.NOT_SUPPORTED_ERR THEN
     raise_application_error(-20120, 'Not supported error');
    WHEN xmldom.INUSE_ATTRIBUTE_ERR THEN
     raise_application_error(-20120, 'In use attr error');
  END;
```

Para ejecutar el procedimiento deberemos ejecutar el siguiente comando en el editor SQLPlus de Oracle 9i:

```
SQL> exec plsqlxml ('c:','bib.xml','error.txt');
```

2.2 El generador de clases XML de Oracle

Oracle contiene el Oracle XML Class Generator que genera un conjunto de archivos Java sobre un DTD existente. Los archivos Java obtenidos a partir del generador de clases XML se pueden utilizar para construir, validar e imprimir documentos XML que se ajuste al DTD dado.

A diferencia de lo que la documentación de Oracle describe, para poder utilizar el generador de clases debemos disponer del archivo XML del que queremos obtener las clases Java incluyendo en el archivo, al principio del documento XML la sintaxis del DTD que genera ese documento. Para comprobar el funcionamiento del generador de clases sobre nuestro ejemplo bib.xml, primero hemos tenido que incluir el contenido del archivo DTD quedando como se muestra:

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

<!DOCTYPE bib[
<!ELEMENT bib  (book* )>
<!ELEMENT book  (title,  isbn, (author+ | editor+ ), publisher, price )>
<!ATTLIST book  year CDATA  #REQUIRED >
<!ELEMENT isbn  (#PCDATA )>
```

```
<!ELEMENT author  (last, first )>
<!ELEMENT editor  (last, first, affiliation )>
<!ELEMENT title  (#PCDATA )>
<!ELEMENT last  (#PCDATA )>
<!ELEMENT first  (#PCDATA )>
<!ELEMENT affiliation  (#PCDATA )>
<!ELEMENT publisher  (#PCDATA )>
<!ELEMENT price  (#PCDATA )>
]>



<bib>
   <book year="1994">
      <title>TCP/IP Illustrated</title>
      <isbn>123-1345-44</isbn>
      <author><last>Stevens</last><first>W.</first></author>
      <publisher>Addison-Wesley</publisher>
      <price> 65.95</price>
   </book>

   <book year="1992">
      <title>Advanced Programming in the Unix environment</title>
      <isbn>12-1345-XX</isbn>
      <author><last>Stevens</last><first>W.</first></author>
      <publisher>Addison-Wesley</publisher>
      <price>65.95</price>
   </book>

   <book year="2000">
      <title>Data on the Web</title>
      <isbn>34-13445-00</isbn>
      <author><last>Abiteboul</last><first>Serge</first></author>
      <author><last>Buneman</last><first>Peter</first></author>
      <author><last>Suciu</last><first>Dan</first></author>
      <publisher>Morgan Kaufmann Publishers</publisher>
      <price>39.95</price>
   </book>

   <book year="1999">
      <title>The Economics of Technology and Content for Digital TV</title>
      <isbn>01-00001-01</isbn>
      <editor>
           <last>Gerbarg</last><first>Darcy</first>
            <affiliation>CITI</affiliation>
      </editor>
         <publisher>Kluwer Academic Publishers</publisher>
      <price>129.95</price>
   </book>

  <book year="2002">
      <title>Oracle 9i.Desarrollo Web</title>
      <isbn>84-415-1406-2</isbn>
      <author><last>Brown</last><first>Bradley</first></author>
      <publisher>Anaya Multimedia</publisher>
      <price>54.00</price>
   </book>


</bib>
```

Sobre este nuevo documento ejecutamos la siguiente aplicación Java que recogerá el archivo como argumento y generará un archivo java por cada uno de los elementos del documento XML:

```
// XmlClassGen.java
import java.io.File;
import java.net.URL;
import oracle.xml.parser.v2.XMLConstants;
import oracle.xml.parser.v2.DOMParser;
import oracle.xml.parser.v2.DTD;
import oracle.xml.parser.v2.XMLDocument;
import oracle.xml.classgen.DTDClassGenerator;
public class XmlClassGen{
  public static void main(String[] args){
       // validate arguments and display usage statement if incorrect
     if (args.length < 1){
       System.out.println("Usage: java SampleMain "+
                          "[-root <rootName>] <fileName>");
       System.out.println("fileName  Input file, XML documentor"
        + "external DTD file");
       System.out.println("-root <rootName>   Name of the root " +
                          "Element " + "(required if the input file " +
                          "is an external DTD)");
       return ;
     }
     try{
       // Try to open the XML File/ External DTD File
       // instantiate the parser
       DOMParser parser = new DOMParser();
       if (args.length == 3)
         parser.parseDTD(fileToURL(args[2]), args[1]);
       else
         parser.parse(fileToURL(args[0]));
       XMLDocument doc = parser.getDocument();
       DTD dtd = (DTD)doc.getDoctype();
       String doctype_name = null;
       if (args.length == 3)
         doctype_name = args[1];
       else
         /* get the Root Element name from the XMLDocument*/
         doctype_name = doc.getDocumentElement().getTagName();
       // generate the Java files...
       DTDClassGenerator generator = new DTDClassGenerator();
       // set generate comments to true
       generator.setGenerateComments(true);
       // set output directory
       generator.setOutputDirectory(".");
       // set validating mode to true
       generator.setValidationMode(true);
       // generate java src
       generator.generate(dtd, doctype_name);
     }catch(Exception e){
       System.out.println ("XML Class Generator: Error " +
        e.toString());
        e.printStackTrace();
     }
  }
   static public URL fileToURL(String sfile){
     File file = new File(sfile);
     String path = file.getAbsolutePath();
     String fSep = System.getProperty("file.separator");
     if (fSep != null && fSep.length() == 1){
       path = path.replace(fSep.charAt(0), '/');
```

```
    }
    if (path.length() > 0 && path.charAt(0) != '/'){
      path = '/' + path;
    }
    try{
      return new URL("file", null, path);
    }catch (java.net.MalformedURLException e){
      /* According to the spec this could only happen if the */
      /* file protocol were not recognized. */
      throw new Error("unexpected MalformedURLException");
    }
  }
}
```

El resultado obtenido serán, como se ha comentado un archivo java por cada elemento del documento para que se pueda generar a partir de estas clases un nuevo documento XML validado con el archivo DTD dado.

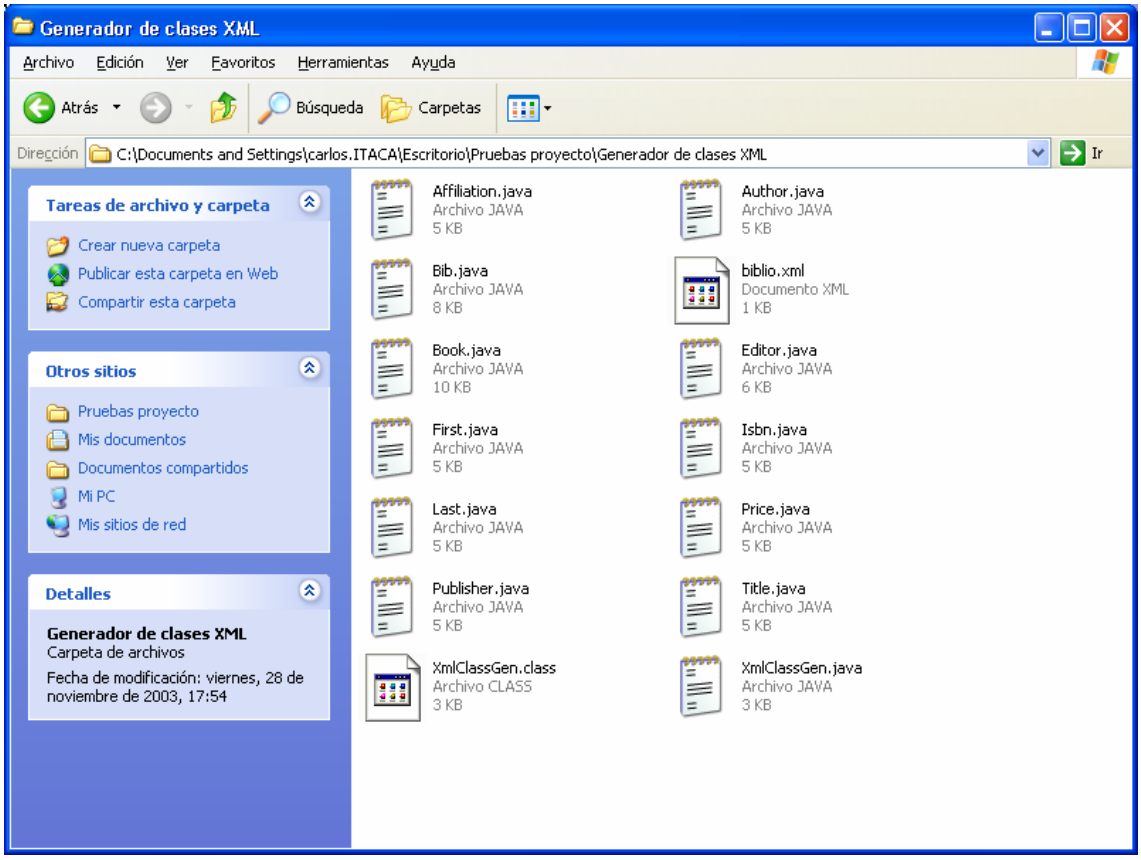


Figura 4.2. Resultado de clases java a partir del generador de clases XML de Oracle

Así, por ejemplo, el código del archivo bib.java contendrá:

```
/* DO NOT EDIT THIS FILE - it is machine-generated */
/* File: Bib.java - generated by XML Class Generator version 2.0.0 at Fri Nov 28 10:22:36 CET 2003
*/

import oracle.xml.parser.v2.DTD;
import oracle.xml.parser.v2.XMLDocument;
import oracle.xml.parser.v2.XMLElement;
import oracle.xml.parser.v2.XMLNode;
```

```
import oracle.xml.parser.v2.DOMParser;
import oracle.xml.io.XMLObjectInput;
import oracle.xml.io.XMLObjectOutput;
import oracle.xml.comp.CXMLContext;
import oracle.xml.classgen.CGNode;
import oracle.xml.classgen.CGDocument;
import oracle.xml.classgen.InvalidContentException;
import oracle.xml.classgen.UnmarshalException;

import java.io.InputStream;
import java.io.OutputStream;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.ObjectInput;
import java.io.ObjectOutput;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.Externalizable;
import java.io.IOException;
import java.net.URL;

/**
 * The Document Class (for the root element Bib)
 */
public class Bib extends CGDocument implements Externalizable
{

   public static DTD globalDTD = null;
   static DOMParser parser = new DOMParser();

   static
   {
      String dtdFile = "Bib_dtd.txt";
      URL url = null;

      try
      {
         url = ClassLoader.getSystemResource(dtdFile);
         parser.parseDTD(url, "bib");
         globalDTD = (DTD)parser.getDoctype();
      }
      catch (Exception e)
      {
         System.out.println("Unexpected error opening DTD file");
      }
   }


   /**
    * Default Constructor
    */
   public Bib()
   {
      super("bib", globalDTD);
      isValidating = true;
   }

   /**
    * Add <code>Book</code> to <code>Bib</code>
    * @param B Node of type <code>Book</code>
    * @exception InvalidContentException  if node cannot be added as per
    *            the Content model of the element.
    * @deprecated Use the setBookmethod instead.
    */
```

```
public void addNode(Book B) throws InvalidContentException
{
   super.addNode(B);
   B.setDocument(this);
}

/**
 * Add the child node to the element Bib
 * @param B Node of type Book
 * @exception InvalidContentException if node cannot be added as per
 *            the Content model of the elemnt.
 */
public void setBook(Book  B)throws InvalidContentException
{
   super.addNode(B);
   B.setDocument(this);
}

/**
 * Get the child node of the element Bib
 * @param book Node of type book
 * @exception InvalidContentException if the node cannot be found
 */
public Book getBook() throws InvalidContentException
{
   return (Book)super.getNode("book");
}

/**
 * Prints the document to the specified OutputStream
 * @param out Java outputstream.
 * @exception InvalidContentException  if the document is not valid
 */
public void print(OutputStream out) throws InvalidContentException
{
   super.print(out);
}

/**
 * Prints the document to the specified OutputStream
 * in the given encoding
 * @param out Java outputstream.
 * @exception InvalidContentException  if the document is not valid
 */
public void print(OutputStream out, String enc) throws InvalidContentException
{
   super.print(out, enc);
}

/**
 * Returns the Bib DTD.
 * @return the DTD
 * @see oracle.xml.parser.v2.DTD
 */
public DTD getDTDNode()
{
   return globalDTD;
}

/**
 * Validate contents of element <code>Bib</code>
 * @return true if valid contents, else false
 * @exception InvalidContentException  if the document is not valid
 */
```

```
public void validateContent() throws InvalidContentException
{
   super.validateContent();
}


/**
 * Unmarshals the input  URL  to create an object.
 * @params url The URL
 * @returns Bib an instance object
 * @exception IOException if there there is an IO error
 * @exception UnmarshalException if there error in unmarshalling
 */
public static Bib unmarshal(URL url)
      throws UnmarshalException
{
   try
   {
      parser.parse(url);
   }
   catch (Exception e)
   {
      throw new UnmarshalException(e.toString());
   }
   return unmarshal();
}

/**
 * Unmarshals the input XML stream to create an object.
 * @params in The XML InputStream
 * @returns Bib an instance object
 * @exception IOException if there there is an IO error
 * @exception UnmarshalException if there error in unmarshalling
 */
public static Bib unmarshal(InputStream in)
      throws UnmarshalException
{
   try
   {
      parser.parse(in);
   }
   catch (Exception e)
   {
      throw new UnmarshalException(e.toString());
   }
   return unmarshal();
}
/**
 * Unmarshals the input file to create an instance of object.
 * @returns Bib an instance object
 * @exception IOException if there there is an IO error
 * @exception UnmarshalException if there error in unmarshalling
 */
private static Bib unmarshal()
      throws UnmarshalException
{
   XMLDocument doc = null;
   XMLElement elem = null;
   try
   {
      doc = parser.getDocument();
      OutputStream os = new FileOutputStream("xml.ser");
      ObjectOutputStream oos = new ObjectOutputStream(os);
      elem = (XMLElement)doc.getDocumentElement();
```

```
            elem.writeExternal(oos);
            oos.flush();
            oos.close();
        }
        catch (Exception e)
        {
            throw new UnmarshalException(e.toString());
        }
        Bib theBib=  new Bib();
        try
        {
            InputStream is = new FileInputStream("xml.ser");
            ObjectInputStream ois = new ObjectInputStream(is);
            elem = theBib.getElementNode();
            elem.readExternal(ois);
            theBib.setElementNode(elem);
            ois.close();
        }
        catch (Exception e)
        {
            throw new UnmarshalException(e.toString());
        }
        return theBib;
    }
    /**
     * Saves the state of the object by creating a binary compressed
     * stream with information about the object.
     * @param out The ObjectOutput stream used to write the  compressed stream.
     * @exception IOException is thrown when there is an exception
                  in writing the compressed stream.
     */
    public void writeExternal(ObjectOutput out)
                throws IOException
    {
        XMLObjectOutput xout = new XMLObjectOutput(out);
        CXMLContext cxmlContext = new CXMLContext();
        super.writeExternal(xout, cxmlContext);
    }

    /**
     * Reads the information written in the compressed stream and  restores the object.
     * @param in the ObjectInput stream used for reading the  compressed stream
     * @exception IOException is thrown when there is an error in
                  reading the input stream.
     * @exception ClassNotFoundException is thrown when the class is not found
     */
    public void readExternal(ObjectInput in)
              throws IOException, ClassNotFoundException
    {
        XMLObjectInput xin = new XMLObjectInput(in);
        CXMLContext cxmlContext = new CXMLContext();
        super.readExternal(xin, cxmlContext);
    }

}
```

El siguiente ejemplo es el código generado para el elemento miembro book (archivo book.java) que contendrá la información necesaria para generar un elemento book

```
/* DO NOT EDIT THIS FILE - it is machine-generated */
/* File: Book.java - generated by XML Class Generator version 2.0.0 at Fri Nov 28 10:22:36 CET 2003 */
```

```
import oracle.xml.parser.v2.DTD;
import oracle.xml.io.XMLObjectInput;
import oracle.xml.io.XMLObjectOutput;
import oracle.xml.comp.CXMLContext;
import oracle.xml.classgen.CGNode;
import oracle.xml.classgen.InvalidContentException;
import java.io.ObjectInput;
import java.io.ObjectOutput;
import java.io.IOException;
import java.io.Externalizable;
/**
 * The Node Class (for the root element Book)
 */
public class Book extends CGNode implements Externalizable
{

   /**
    * Default Constructor
    */
   public Book()
   {
      super("book");
      isValidating = true;
   }

   /**
    * Constructor with the required Attributes
    * @param varYear for the Attribute year
    * @exception InvalidContentException if invalid value is specified for an attribute
    */
   public Book(String varYear) throws InvalidContentException
   {
      super("book");
      try
      {
         setYear(varYear);
      }
      catch (IllegalArgumentException e) {
         throw new InvalidContentException("Invalid Attribute value specified"); }
      isValidating = true;
   }

   /**
    * Add <code>Title</code> to <code>Book</code>
    * @param T Node of type <code>Title</code>
    * @exception InvalidContentException  if node cannot be added as per
    *            the Content model of the element.
    * @deprecated Use the setTitlemethod instead.
    */
   public void addNode(Title T) throws InvalidContentException
   {
      super.addNode(T);
   }

   /**
    * Add the child node to the element Book
    * @param T Node of type Title
    * @exception InvalidContentException if node cannot be added as per
    *            the Content model of the elemnt.
    */
   public void setTitle(Title  T)throws InvalidContentException
   {
      super.addNode(T);
```

```
}

/**
 * Get the child node of the element Book
 * @param title Node of type title
 * @exception InvalidContentException if the node cannot be found
 */
public Title getTitle() throws InvalidContentException
{
   return (Title)super.getNode("title");
}

/**
 * Add <code>Isbn</code> to <code>Book</code>
 * @param I Node of type <code>Isbn</code>
 * @exception InvalidContentException  if node cannot be added as per
 *            the Content model of the element.
 * @deprecated Use the setIsbnmethod instead.
 */
public void addNode(Isbn I) throws InvalidContentException
{
   super.addNode(I);
}

/**
 * Add the child node to the element Book
 * @param I Node of type Isbn
 * @exception InvalidContentException if node cannot be added as per
 *            the Content model of the elemnt.
 */
public void setIsbn(Isbn  I)throws InvalidContentException
{
   super.addNode(I);
}

/**
 * Get the child node of the element Book
 * @param isbn Node of type isbn
 * @exception InvalidContentException if the node cannot be found
 */
public Isbn getIsbn() throws InvalidContentException
{
   return (Isbn)super.getNode("isbn");
}

/**
 * Add <code>Author</code> to <code>Book</code>
 * @param A Node of type <code>Author</code>
 * @exception InvalidContentException  if node cannot be added as per
 *            the Content model of the element.
 * @deprecated Use the setAuthormethod instead.
 */
public void addNode(Author A) throws InvalidContentException
{
   super.addNode(A);
}

/**
 * Add the child node to the element Book
 * @param A Node of type Author
 * @exception InvalidContentException if node cannot be added as per
 *            the Content model of the elemnt.
 */
public void setAuthor(Author  A)throws InvalidContentException
```

```
{
   super.addNode(A);
}

/**
 * Get the child node of the element Book
 * @param author Node of type author
 * @exception InvalidContentException if the node cannot be found
 */
public Author getAuthor() throws InvalidContentException
{
   return (Author)super.getNode("author");
}

/**
 * Add <code>Editor</code> to <code>Book</code>
 * @param E Node of type <code>Editor</code>
 * @exception InvalidContentException  if node cannot be added as per
 *            the Content model of the element.
 * @deprecated Use the setEditormethod instead.
 */
public void addNode(Editor E) throws InvalidContentException
{
   super.addNode(E);
}

/**
 * Add the child node to the element Book
 * @param E Node of type Editor
 * @exception InvalidContentException if node cannot be added as per
 *            the Content model of the elemnt.
 */
public void setEditor(Editor  E)throws InvalidContentException
{
   super.addNode(E);
}

/**
 * Get the child node of the element Book
 * @param editor Node of type editor
 * @exception InvalidContentException if the node cannot be found
 */
public Editor getEditor() throws InvalidContentException
{
   return (Editor)super.getNode("editor");
}

/**
 * Add <code>Publisher</code> to <code>Book</code>
 * @param P Node of type <code>Publisher</code>
 * @exception InvalidContentException  if node cannot be added as per
 *            the Content model of the element.
 * @deprecated Use the setPublishermethod instead.
 */
public void addNode(Publisher P) throws InvalidContentException
{
   super.addNode(P);
}

/**
 * Add the child node to the element Book
 * @param P Node of type Publisher
 * @exception InvalidContentException if node cannot be added as per
 *            the Content model of the elemnt.
```

```
 */
public void setPublisher(Publisher  P)throws InvalidContentException
{
    super.addNode(P);
}

/**
 * Get the child node of the element Book
 * @param publisher Node of type publisher
 * @exception InvalidContentException if the node cannot be found
 */
public Publisher getPublisher() throws InvalidContentException
{
    return (Publisher)super.getNode("publisher");
}

/**
 * Add <code>Price</code> to <code>Book</code>
 * @param P Node of type <code>Price</code>
 * @exception InvalidContentException  if node cannot be added as per
 *            the Content model of the element.
 * @deprecated Use the setPricemethod instead.
 */
public void addNode(Price P) throws InvalidContentException
{
    super.addNode(P);
}

/**
 * Add the child node to the element Book
 * @param P Node of type Price
 * @exception InvalidContentException if node cannot be added as per
 *            the Content model of the elemnt.
 */
public void setPrice(Price  P)throws InvalidContentException
{
    super.addNode(P);
}

/**
 * Get the child node of the element Book
 * @param price Node of type price
 * @exception InvalidContentException if the node cannot be found
 */
public Price getPrice() throws InvalidContentException
{
    return (Price)super.getNode("price");
}

/**
 * Sets the value of attribute <code>year</code>
 * @param theData value of the attribute
 */
public void setYear(String theData)

{
    setAttribute("year", theData);
}

/**
 * Gets the value of attribute <code>year</code>
 * @param theData value of the attribute
 */
public String getYear()
```

```
  {
     return super.getAttribute("year");
  }

  /**
   * Returns the Bib DTD.
   * @return the DTD
   * @see oracle.xml.parser.v2.DTD
   */
  public DTD getDTDNode()
  {
     return Bib.globalDTD;
  }

  /**
   * Validate contents of element <code>Book</code>
   * @return true if valid contents, else false
   * @exception InvalidContentException  if the document is not valid
   */
  public void validateContent() throws InvalidContentException
  {
     super.validateContent();
  }

  /**
   * Get Document Class <code>Bib</code>
   * @return <code>Bib</code> class
   */
  public Bib getDocument()
  {
     return (Bib)super.getCGDocument();
  }

  /**
   * Saves the state of the object by creating a binary compressed
   * stream with information about the object.
   * @param out The ObjectOutput stream used to write the  compressed stream.
   * @exception IOException is thrown when there is an exception
                in writing the compressed stream.
   */
  public void writeExternal(ObjectOutput out)
            throws IOException
  {
     XMLObjectOutput xout = new XMLObjectOutput(out);
     CXMLContext cxmlContext = new CXMLContext();
     super.writeExternal(xout, cxmlContext);
  }

  /**
   * Reads the information written in the compressed stream and  restores the object.
   * @param in the ObjectInput stream used for reading the  compressed stream
   * @exception IOException is thrown when there is an error in
                reading the input stream.
   * @exception ClassNotFoundException is thrown when the class is not found
   */
  public void readExternal(ObjectInput in)
           throws IOException, ClassNotFoundException
  {
     XMLObjectInput xin = new XMLObjectInput(in);
     CXMLContext cxmlContext = new CXMLContext();
     super.readExternal(xin, cxmlContext);
  }
}
```

Por último, y como muestra del código generador de un elemento del nodo book del documento bib.xml tenemos que el código generado para el elemento autor sería (author.java):

```
/* DO NOT EDIT THIS FILE - it is machine-generated */
/* File: Author.java - generated by XML Class Generator version 2.0.0 at Fri Nov 28 10:22:36 CET 2003 */

import oracle.xml.parser.v2.DTD;
import oracle.xml.io.XMLObjectInput;
import oracle.xml.io.XMLObjectOutput;
import oracle.xml.comp.CXMLContext;
import oracle.xml.classgen.CGNode;
import oracle.xml.classgen.InvalidContentException;
import java.io.ObjectInput;
import java.io.ObjectOutput;
import java.io.IOException;
import java.io.Externalizable;
/**
 * The Node Class (for the root element Author)
 */
public class Author extends CGNode implements Externalizable
{

   /**
    * Default Constructor
    */
   public Author()
   {
      super("author");
      isValidating = true;
   }

   /**
    * Add <code>Last</code> to <code>Author</code>
    * @param L Node of type <code>Last</code>
    * @exception InvalidContentException  if node cannot be added as per
    *           the Content model of the element.
    * @deprecated Use the setLastmethod instead.
    */
   public void addNode(Last L) throws InvalidContentException
   {
      super.addNode(L);
   }

   /**
    * Add the child node to the element Author
    * @param L Node of type Last
    * @exception InvalidContentException if node cannot be added as per
    *           the Content model of the elemnt.
    */
   public void setLast(Last  L)throws InvalidContentException
   {
      super.addNode(L);
   }

   /**
    * Get the child node of the element Author
    * @param last Node of type last
    * @exception InvalidContentException if the node cannot be found
    */
   public Last getLast() throws InvalidContentException
   {
      return (Last)super.getNode("last");
```

```
}

/**
 * Add <code>First</code> to <code>Author</code>
 * @param F Node of type <code>First</code>
 * @exception InvalidContentException  if node cannot be added as per
 *            the Content model of the element.
 * @deprecated Use the setFirstmethod instead.
 */
public void addNode(First F) throws InvalidContentException
{
   super.addNode(F);
}

/**
 * Add the child node to the element Author
 * @param F Node of type First
 * @exception InvalidContentException if node cannot be added as per
 *            the Content model of the elemnt.
 */
public void setFirst(First  F)throws InvalidContentException
{
   super.addNode(F);
}

/**
 * Get the child node of the element Author
 * @param first Node of type first
 * @exception InvalidContentException if the node cannot be found
 */
public First getFirst() throws InvalidContentException
{
   return (First)super.getNode("first");
}

/**
 * Returns the Bib DTD.
 * @return the DTD
 * @see oracle.xml.parser.v2.DTD
 */
public DTD getDTDNode()
{
   return Bib.globalDTD;
}

/**
 * Validate contents of element <code>Author</code>
 * @return true if valid contents, else false
 * @exception InvalidContentException  if the document is not valid
 */
public void validateContent() throws InvalidContentException
{
   super.validateContent();
}

/**
 * Get Document Class <code>Bib</code>
 * @return <code>Bib</code> class
 */
public Bib getDocument()
{
   return (Bib)super.getCGDocument();
}
```

```
   /**
    * Saves the state of the object by creating a binary compressed
    * stream with information about the object.
    * @param out The ObjectOutput stream used to write the  compressed stream.
    * @exception IOException is thrown when there is an exception
                in writing the compressed stream.
    */
   public void writeExternal(ObjectOutput out)
              throws IOException
   {
      XMLObjectOutput xout = new XMLObjectOutput(out);
      CXMLContext cxmlContext = new CXMLContext();
      super.writeExternal(xout, cxmlContext);
   }

   /**
    * Reads the information written in the compressed stream and  restores the object.
    * @param in the ObjectInput stream used for reading the  compressed stream
    * @exception IOException is thrown when there is an error in
                reading the input stream.
    * @exception ClassNotFoundException is thrown when the class is not found
    */
   public void readExternal(ObjectInput in)
            throws IOException, ClassNotFoundException
   {
      XMLObjectInput xin = new XMLObjectInput(in);
      CXMLContext cxmlContext = new CXMLContext();
      super.readExternal(xin, cxmlContext);
   }

}
```

A partir de estas clases generadas se puede crear dos nuevos registros book. Los atributos necesarios se especifican añadiéndolos como parámetros al constructor. También se crean los elementos que conforman el objeto XML, en este caso title, isbn, author, publisher y price. Una vez creado el elemento book se añade entonces como un nodo al elemento raíz del documento bib.
Veamos como se añadiría un nuevo libro a nuestro catálogo de libros:

```
public class CreateBook
{
        public static void main(String args[])
        {
                try
                {
                        Bib biblioList = new Bib();
                        //Nuevo libro
                        Book book1= new Book("2003");  //crea book1 y define year
                        book1.setYear("2003");
                        Title title1= new Title("Manual de Oracle XML");
                        Isbn isbn1= new Isbn("84-481-3034-0");
                        Author author1= new Author();
                        Last last1= new Last("Chang");
                        author1.setLast(last1);
                        First first1= new First("Ben");
                        author1.setFirst(first1);
                        Publisher publisher1= new Publisher("Osborne Mac Graw Hill");
                        Price price1= new Price("35.00");

                        // añadimos los elementos de book1 como nodos del arbol
                        book1.addNode(title1);
```

```
                        book1.addNode(isbn1);
                        author1.addNode(last1);
                        author1.addNode(first1);
                        book1.addNode(author1);
                        book1.addNode(publisher1);
                        book1.addNode(price1);

                        // añadimos book1 como nodo del arbol al documento  raíz biblioList
                        biblioList.addNode(book1);

                        biblioList.validateContent();
                        biblioList.print(System.out);

                }
                catch (Exception e)
                {
                        System.out.println(e.toString());
                        e.printStackTrace();
                }
        }
}
```

Notemos la importancia que tiene el generador de clases XML ya que la idea es que la entrada de la aplicación anterior puede recibirse a través de varias fuentes: a partir de un formulario web, un analizador SAX o una aplicación JDBC y generar un documento XML bien formado y en consonancia con el archivo DTD.

La salida de la aplicación anterior nos genera el siguiente documento XML:

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

<bib>
   <book year="2003">
      <title> Manual de Oracle XML </title>
      <isbn>84-481-3034-0</isbn>
      <author><last> Chang </last><first>Ben.</first></author>
      <publisher> Osborne Mac Graw Hill </publisher>
      <price> 35.00</price>
   </book>
</bib>
```

2.3 La SQL XML Utility de Oracle

La utilidad SQL XML son un conjunto de clases en Java que permiten:

- Generar documentos XML a partir de un consulta SQL o de los datos obtenidos a través del JDBC
- Recuperar datos de un documento XML almacenándolos en una tabla de la base de datos

La SQL XML Utility de Oracle consta de cinco componentes, cuatro de ellos son clases java: OracleXML, OracleXMLStore, OracleXMLSave y OracleXMLQuery y el quinto componente es un paquete para PL/SQL muy similar al OracleXMLStore llamado xmlgen.

El modo de ejecución del SQL XML Utility en una base de datos es como se muestra en el gráfico siguiente:

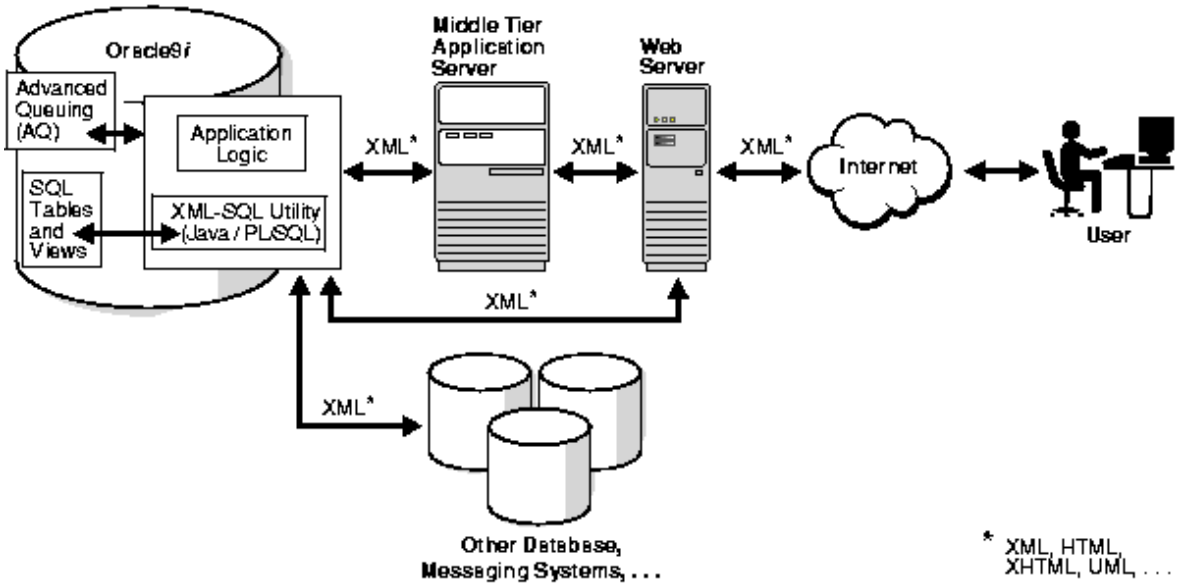


Figura 4.3. Esquema de funcionamiento del XML SQL Utility de Oracle

El componente OracleXML a partir de los dos componentes getXML y putXML que contiene, permite mediante el uso de ordenes introducidas por línea de comandos extraer o insertar información de una tabla en formato XML o bien insertar datos a partir de información XML.
El uso, por ejemplo de getXML sería:

```
c:\oracle\ora92\bin\>java OracleXML .user xml/xml “select * from emp”
```

Lo que nos produciría el resultado de la consulta SQL en formato XML

Por su parte, el funcionamiento de putXML seria similar.

OracleXMLQuery, muy similar a getXML de OracleXML, es una API que permite generar documentos XML a partir de una consulta o los resultados provenientes de JDBC. Veamos como funcionaría este componente de la Utilidad SQL XML de Oracle:

```
// QueryXML.java
import oracle.xml.sql.query.OracleXMLQuery;
import oracle.jdbc.driver.*;
import oracle.sql.*;
import java.sql.*;

public class QueryXML{
 public static void main(String args[]){
   try{
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
    Connection conn =
           DriverManager.getConnection("jdbc:oracle:oci8:@","xml","xml");
    OracleXMLQuery query = new OracleXMLQuery(conn,
              "select * from emp where deptno='20'");
    System.out.println(query.getXMLString());
    query.close();
```

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

El resultado de esta aplicación sería:

```
<?xml version = '1.0'?>
<ROWSET>
  <ROW num="1">
    <EMPNO>7369</EMPNO>
    <ENAME>SMITH</ENAME>
    <JOB>CLERK</JOB>
    <MGR>7902</MGR>
    <HIREDATE>12/17/1980
0:0:0</HIREDATE>
    <SAL>800</SAL>
    <DEPTNO>20</DEPTNO>
  </ROW>
  <ROW num="2">
    <EMPNO>7566</EMPNO>
    <ENAME>JONES</ENAME>
    <JOB>MANAGER</JOB>
    <MGR>7839</MGR>
    <HIREDATE>4/2/1981
0:0:0</HIREDATE>
    <SAL>2975</SAL>
    <DEPTNO>20</DEPTNO>
  </ROW>
  <ROW num="3">
    <EMPNO>7788</EMPNO>
    <ENAME>SCOTT</ENAME>
    <JOB>ANALYST</JOB>
    <MGR>7566</MGR>
    <HIREDATE>4/19/1987
0:0:0</HIREDATE>
    <SAL>3000</SAL>
    <DEPTNO>20</DEPTNO>
  </ROW>
  <ROW num="4">
    <EMPNO>7876</EMPNO>
    <ENAME>ADAMS</ENAME>
    <JOB>CLERK</JOB>
    <MGR>7788</MGR>
    <HIREDATE>5/23/1987
0:0:0</HIREDATE>
    <SAL>1100</SAL>
    <DEPTNO>20</DEPTNO>
  </ROW>
  <ROW num="5">
    <EMPNO>7902</EMPNO>
    <ENAME>FORD</ENAME>
    <JOB>ANALYST</JOB>
    <MGR>7566</MGR>
    <HIREDATE>12/3/1981
0:0:0</HIREDATE>
    <SAL>3000</SAL>
    <DEPTNO>20</DEPTNO>
  </ROW>
  <ROW num="6">
    <EMPNO>1000</EMPNO>
    <ENAME>MARIN</ENAME>
    <JOB>MANAGER</JOB>
    <MGR>7902</MGR>
    <HIREDATE>4/12/2001
0:0:0</HIREDATE>
    <SAL>8000</SAL>
    <DEPTNO>20</DEPTNO>
  </ROW>
  <ROW num="7">
    <EMPNO>1000</EMPNO>
    <ENAME>MARIN</ENAME>
    <JOB>MANAGER</JOB>
    <MGR>7902</MGR>
    <HIREDATE>4/12/2001
0:0:0</HIREDATE>
    <SAL>8000</SAL>
    <DEPTNO>20</DEPTNO>
  </ROW>
  <ROW num="8">
    <EMPNO>9367</EMPNO>
    <ENAME>SMITH</ENAME>
    <JOB>CLERK</JOB>
    <MGR>7902</MGR>
    <HIREDATE>4/11/2001
0:0:0</HIREDATE>
    <SAL>800</SAL>
    <DEPTNO>20</DEPTNO>
  </ROW>
</ROWSET>
```

Este componente tiene más utilidad que getXML del componente OracleXML ya que puede incluirse en aplicaciones Java a diferencia de getXML que solamente puede ejecutarse mediante línea de comandos.

Por su parte, el componente OracleXMLSave es también una API cuya utilidad radica en almacenar los datos de un documento XML en una base de datos. Esta utilidad mediante la asociación de los nombres de etiquetas con los nombres de las columnas de la tabla de la base de datos, permite almacenar los datos por columnas.
Veamos un ejemplo sencillo en que insertamos en la tabla emp del usuario xml un nuevo empleado:

```
// InsertXMLEmp.java
import oracle.xml.sql.dml.OracleXMLSave;
import oracle.jdbc.driver.*;
import oracle.sql.*;
import java.sql.*;

public class InsertXMLEmp{
   // Recommended this be passed in, not hardcoded, this is for
   // demonstration purposes only.
   public static String xmlData = new String(
            "<?xml version = '1.0'?>"+
                                "<ROWSET>"+
                                "<ROW num='1'>"+
                                "<EMPNO>5555</EMPNO>"+
                                "<ENAME>MARIN</ENAME>"+
                                "<JOB>BOSS</JOB>"+
                                "<MGR>5555</MGR>"+
                                "<HIREDATE>29/11/2003 0:0:0</HIREDATE>"+
                                "<SAL>8000</SAL>"+
                                "<DEPTNO>30</DEPTNO>"+
                                "</ROW>"+
                                "</ROWSET>");
 public static void main(String args[]){
   try{
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
    Connection conn = DriverManager.getConnection("jdbc:oracle:oci8:@","xml","xml");
    OracleXMLSave sav = new OracleXMLSave(conn,"EMP");
    sav.insertXML(xmlData);
    sav.close();
   }catch(Exception e){
       System.out.println("Error:"+e.getMessage());
   }
 }
}
```

El resultado de esta aplicación es la inserción del nuevo registro en la tabla emp. Observemos que el documento XML a insertar, en este caso un solo elemento, se ha definido por simplicidad como un String pero obviamente, este archivo XML podría recogerse desde cualquier origen

El resultado en la tabla emp desde SQLPlus será:

```
SQL> select * from emp
  2  where empno=5555;

    EMPNO ENAME      JOB              MGR HIREDATE        SAL       COMM    DEPTNO
---------- ---------- --------- ---------- -------- ---------- ----------
     5555 MARIN      BOSS            5555 11/05/05       8000         30
```

Por último, tenemos el paquete PL/SQL xmlgen. Este paquete es idéntico a la clase OracleXMLStore pero integrado dentro del entorno PL/SQL. Para poder utilizar este paquete se debe ejecutar el script SQL xmlgen.sql. Este paquete permite obtener en formato XML el resultado de una consulta SQL.

Veamos un ejemplo sencillo sobre la tabla emp:

```
SQL> select xmlgen.getXML('select * from emp where deptno=10')
  2  from dual;
```

El resultado obtenido será:

```
XMLGEN.GETXML('SELECT*FROMEMPWH
EREDEPTNO=10')
-----------------------------------------------------------
------------------
<?xml version = '1.0'?>
<ROWSET>
  <ROW num="1">
    <EMPNO>7369</EMPNO>
    <ENAME>SMITH</ENAME>
    <JOB>CLERK</JOB>
    <MGR>7902</MGR>
    <HIREDATE>12/17/1980
0:0:0</HIREDATE>
    <SAL>800</SAL>
    <DEPTNO>20</DEPTNO>
  </ROW>
  <ROW num="2">
    <EMPNO>7566</EMPNO>
    <ENAME>JONES</ENAME>
    <JOB>MANAGER</JOB>
    <MGR>7839</MGR>
    <HIREDATE>4/2/1981
0:0:0</HIREDATE>
    <SAL>2975</SAL>
    <DEPTNO>20</DEPTNO>
  </ROW>
  <ROW num="3">
    <EMPNO>7788</EMPNO>
    <ENAME>SCOTT</ENAME>
    <JOB>ANALYST</JOB>
    <MGR>7566</MGR>
    <HIREDATE>4/19/1987
0:0:0</HIREDATE>
    <SAL>3000</SAL>
    <DEPTNO>20</DEPTNO>
  </ROW>
  <ROW num="4">
    <EMPNO>7876</EMPNO>
    <ENAME>ADAMS</ENAME>
    <JOB>CLERK</JOB>
    <MGR>7788</MGR>
    <HIREDATE>5/23/1987
0:0:0</HIREDATE>
    <SAL>1100</SAL>
    <DEPTNO>20</DEPTNO>
  </ROW>
  <ROW num="5">
    <EMPNO>7902</EMPNO>
    <ENAME>FORD</ENAME>
    <JOB>ANALYST</JOB>
    <MGR>7566</MGR>
    <HIREDATE>12/3/1981
0:0:0</HIREDATE>
    <SAL>3000</SAL>
    <DEPTNO>20</DEPTNO>
  </ROW>
  <ROW num="6">
    <EMPNO>1000</EMPNO>
    <ENAME>MARIN</ENAME>
    <JOB>MANAGER</JOB>
    <MGR>7902</MGR>
    <HIREDATE>4/12/2001
0:0:0</HIREDATE>
    <SAL>8000</SAL>
    <DEPTNO>20</DEPTNO>
  </ROW>
  <ROW num="7">
    <EMPNO>1000</EMPNO>
    <ENAME>MARIN</ENAME>
    <JOB>MANAGER</JOB>
    <MGR>7902</MGR>
    <HIREDATE>4/12/2001
0:0:0</HIREDATE>
    <SAL>8000</SAL>
    <DEPTNO>20</DEPTNO>
  </ROW>
  <ROW num="8">
    <EMPNO>9367</EMPNO>
    <ENAME>SMITH</ENAME>
    <JOB>CLERK</JOB>
    <MGR>7902</MGR>
    <HIREDATE>4/11/2001
0:0:0</HIREDATE>
    <SAL>800</SAL>
    <DEPTNO>20</DEPTNO>
  </ROW>
</ROWSET>

<?xml version = '1.0'?>
<ROWSET>
  <ROW num="1">
    <EMPNO>7782</EMPNO>
    <ENAME>CLARK</ENAME>
    <JOB>MANAGER</JOB>
    <MGR>7839</MGR>
    <HIREDATE>6/9/1981
0:0:0</HIREDATE>
    <SAL>2450</SAL>
    <DEPTNO>10</DEPTNO>
  </ROW>
  <ROW num="2">
    <EMPNO>7839</EMPNO>
    <ENAME>KING</ENAME>
    <JOB>PRESIDENT</JOB>
    <HIREDATE>11/17/1981
0:0:0</HIREDATE>
    <SAL>5000</SAL>
    <DEPTNO>10</DEPTNO>
  </ROW>
  <ROW num="3">
    <EMPNO>7934</EMPNO>
    <ENAME>MILLER</ENAME>
    <JOB>CLERK</JOB>
    <MGR>7782</MGR>
    <HIREDATE>1/23/1982
0:0:0</HIREDATE>
    <SAL>1300</SAL>
    <DEPTNO>10</DEPTNO>
  </ROW>
</ROWSET>
```

2.4 Oracle 9i y bases de datos para XML

Hasta ahora hemos visto como Oracle 9i a partir de funcionalidades implementadas en Java permite generar o extraer documentos XML a partir de consultas sobre la base de datos, como a partir de API's podemos extraer información de tablas en formato XML o como a partir de un documento XML con su correspondiente DTD podemos extraer todas las clases necesarias en Java para poder generar documentos basados en éste a partir de estas clases.

Pero Oracle 9i también permite almacenar documentos XML en tablas. Cuando necesitamos almacenar en la base de datos un documento o bien información XML podemos hacerlo de tres formas:

- Asignando el documento completo como un único objeto intacto y almacenándolo en una tabla como un CLOB (Character Large Object)
- Asignando elementos XML a tablas y columnas relacionales dentro de la base de datos
- Asignando fragmentos de documentos XML como CLOB y el resto como tablas

En principio dependerá de la estructura del documento XML y de las operaciones que debamos realizar lo que nos hará decantarnos por la elección de una de estas tres formas posibles.

Para poder implementar estas tres posibles estrategias, Oracle9i soporta un nuevo tipo de objeto definido, el XMLType con unas funciones incorporadas que ofrecen mecanismos para crear, extraer e indexar datos XML. El usuario, además, puede generar documentos XML mediante funciones SQL, funciones de SYS_XMLGEN y SYS_XMLAGG y mediante el paquete DBMS_XMLGEN de PL/SQL.

XMLType es un nuevo tipo de dato del servidor que se puede utilizar tanto en columnas, como en tablas como en vistas. Las variables de tipo XMLType se pueden utilizar en procedimientos almacenados PL/SQL tanto como parámetros como valores de retorno

XMLType incluye funciones miembro útiles que pueden operar sobre el contenido XML, así por ejemplo la función extract extrae un nodo específico de una instancia XMLType
Utilizar el tipo XMLType en una base de datos aporta las siguientes ventajas:

- une los dos mundos de XML y SQL y posibilita ejecutar operaciones SQL sobre XML y operaciones XML sobre SQL
- aporta funciones de creación, indexación, ayuda, navegación,…

Los métodos que incorpora el XMLType son los que se muestran a continuación:

Función XMLType	Sintaxis	Descripción
createXML()	STATIC FUNCTION createXML(xmlval IN varchar2) RETURN sys.XMLType deterministic	Función estática que crea un tipo XMLType a partir de una cadena de caracteres. También chequea que el nuevo tipo de dato corresponda a un contenido XML bien formado
createXML()	STATIC FUNCTION createXML(xmlval IN clob) RETURN sys.XMLType deterministic	Función que genera un XMLType a partir de un dato CLOB. También chequea que el nuevo valor corresponda a un XML bien formado
existsNode()	MEMBER FUNCTION existsNode(xpath IN varchar2) RETURN number deterministic	Dada una expresión XPath chequea si esa expresión se puede aplicar sobre el documento y puede devolver algún nodo

Función XMLType	Sintaxis	Descripción
extract()	MEMBER FUNCTION extract(xpath IN varchar2) RETURN sys.XMLType deterministic	Función que dada una expresión XPath, la aplica al documento y devuelve el fragmento sobre el que accedemos como tipo de dato XMLType
isFragment()	MEMBER FUNCTION isFragment() RETURN number	Comprueba si el documento es un fragmento. Tendremos un fragmento cuando hayamos aplicado la función extract o cualquier otra operación que nos haya retornado una secuencia de nodos
getClobVal()	MEMBER FUNCTION getClobVal() RETURN clob deterministic	Recoge el XMLType como un CLOB
getStringVal()	MEMBER FUNCTION getStringVal() RETURN varchar2 deterministic	Recoge el XML como una cadena de caracteres
getNumberVal()	MEMBER FUNCTION getNumberVal() RETURN number deterministic	Obtiene el valor numérico apuntado por el XMLType como dato numérico

Vayamos ahora a comprobar como Oracle9i permite a partir del tipo de datos XMLType almacenar documentos XML y como se puede después consultar, actualizar o borrar información de estos documentos XML.

2.4.1 Almacenamiento de documentos XML como CLOB

Partimos para realizar estos ejemplos del mismo documento bib.xml con su correspondiente archivo DTD (bib.dtd) y el usuario xml creado para tal fin.

Lo primero que vamos a hacer es crear una tabla que permita albergar todo el documento bib.xml; para ello creamos la tabla bib desde el editor SQLPlus:

```
SQL> create table bib
  2  (
  3  contenido sys.XMLType
  4  )
  5  XMLType column contenido store as CLOB
  6  (
  7  storage (initial 12000 next 12000)
  8  chunk 12000 cache
  9  )
 10  ;

Tabla creada.
```

Observemos que hemos creado una tabla llamada bib con una sola columna llamada contenido que hemos definido como CLOB y le hemos asignado diferentes parámetros de almacenamiento

Una vez hecho esto, lo que debemos hacer ahora es extraer el contenido del documento XML bib.xml y almacenarlo dentro de la columna contenido de la tabla bib que acabamos de crear:

```
SQL> create directory XMLFILES as 'C:\Documents and Settings\carlos.ITACA\Escritorio\Pruebas
proyecto' ;

SQL>  declare
  2   CONTENT CLOB := ' ';
  3   SOURCE bfile := bfilename('XMLFILES','bib.xml');
  4   begin
  5   DBMS_LOB.fileOpen(SOURCE,DBMS_LOB.file_readonly);
  6   DBMS_LOB.loadFromFile(CONTENT,SOURCE,DBMS_LOB.getLength(SOURCE),1,1);
  7   DBMS_LOB.fileClose(SOURCE);
  8   insert into bib (contenido)
  9   values(sys.xmltype.createXML(CONTENT));
 10   commit;
 11   end;
 12  /
```

Comprobemos ahora el contenido de la tabla bib mediante la función getClobVal(), para ver el contenido completo debemos modificar la variable de entorno long y si no queremos ver la cabecera de la consulta también modificaremos la variable pagesize:

```
SQL> set long 10000
SQL> set pagesize 0
SQL> select b.contenido.getclobval()
  2  from bib b;
```

En este caso el resultado no variará si para comprobar el contenido de la tabla bib ejecutáramos sencillamente un 'select * from bib'

El resultado será:

```
B.CONTENIDO.GETCLOBVAL()
--------------------------------------------------------------
------------------
<bib>

   <book year="1994">

      <title>TCP/IP Illustrated</title>

      <isbn>123-1345-44</isbn>


<author><last>Stevens</last><first>W.</first>
</author>

      <publisher>Addison-Wesley</publisher>

      <price> 65.95</price>

   </book>



   <book year="1992">

      <title>Advanced Programming in the
Unix environment</title>

      <isbn>12-1345-XX</isbn>


<author><last>Stevens</last><first>W.</first>
</author>

      <publisher>Addison-Wesley</publisher>

      <price>65.95</price>

   </book>



   <book year="2000">

      <title>Data on the Web</title>

      <isbn>34-13445-00</isbn>


<author><last>Abiteboul</last><first>Serge</f
irst></author>


<author><last>Buneman</last><first>Peter</f
irst></author>


<author><last>Suciu</last><first>Dan</first><
/author>

      <publisher>Morgan Kaufmann
Publishers</publisher>
```

```
      <price>39.95</price>

   </book>



   <book year="1999">

      <title>The Economics of Technology and
Content for Digital TV</title>

      <isbn>01-00001-01</isbn>

      <editor>


<last>Gerbarg</last><first>Darcy</first>

            <affiliation>CITI</affiliation>

      </editor>

         <publisher>Kluwer              Academic
Publishers</publisher>

      <price>129.95</price>
```

```
   </book>



  <book year="2002">

      <title>Oracle 9i.Desarrollo Web</title>

      <isbn>84-415-1406-2</isbn>


<author><last>Brown</last><first>Bradley</fir
st></author>

      <publisher>Anaya
Multimedia</publisher>

      <price>54.00</price>

   </book>




</bib>
```

Veamos que ocurre si guardásemos la información del documento XML bib.xml de otra forma, lo guardaremos en un solo campo de una tabla, a la que llamaremos biblio y que contendrá:

```
SQL> create table biblio(
  2  biblio_id number(3),
  3  biblio_dat sys.xmltype);
```

Pasamos ahora el contenido del documento XML bib.xml al campo biblio_dat de esta tabla a través del mismo procedimiento que hicimos en el caso anterior

```
SQL>  declare
  2   CONTENT CLOB := ' ';
  3   SOURCE bfile := bfilename('XMLFILES','bib.xml');
  4   begin
  5   DBMS_LOB.fileOpen(SOURCE,DBMS_LOB.file_readonly);
  6   DBMS_LOB.loadFromFile(CONTENT,SOURCE,DBMS_LOB.getLength(SOURCE),1,1);
  7   DBMS_LOB.fileClose(SOURCE);
  8   insert into biblio (biblio_dat)
  9   values(sys.xmltype.createXML(CONTENT));
 10   commit;
 11   end;
 12  /
```

Siendo el resultado obtenido el siguiente:

```
SQL> select * from biblio;

<bib>
  <book year="1994">
    <title>TCP/IP Illustrated</title>
    <isbn>123-1345-44</isbn>
    <author>
```

```
      <last>Stevens</last>
      <first>W.</first>
    </author>
    <publisher>Addison-Wesley</publisher>
    <price>65.95</price>
  </book>
  <book year="1992">
```

```
   <title>Advanced Programming in the Unix
environment</title>
   <isbn>12-1345-XX</isbn>
   <author>
    <last>Stevens</last>
    <first>W.</first>
   </author>
   <publisher>Addison-Wesley</publisher>
   <price>65.95</price>
  </book>
  <book year="2000">
   <title>Data on the Web</title>
   <isbn>34-13445-00</isbn>
   <author>
    <last>Abiteboul</last>
    <first>Serge</first>
   </author>
   <author>
    <last>Buneman</last>
    <first>Peter</first>
   </author>
   <author>
    <last>Suciu</last>
    <first>Dan</first>
   </author>
   <publisher>Morgan                Kaufmann
Publishers</publisher>
   <price>39.95</price>
  </book>
  <book year="1999">
   <title>The Economics of Technology and
Content for Digital TV</title>
   <isbn>01-00001-01</isbn>
   <editor>
    <last>Gerbarg</last>
    <first>Darcy</first>
    <affiliation>CITI</affiliation>
   </editor>
   <publisher>Kluwer                Academic
Publishers</publisher>
   <price>129.95</price>
  </book>
  <book year="2002">
   <title>Oracle 9i.Desarrollo Web</title>
   <isbn>84-415-1406-2</isbn>
   <author>
    <last>Brown</last>
    <first>Bradley</first>
   </author>
   <publisher>Anaya Multimedia</publisher>
   <price>54.00</price>
  </book>
</bib>
```

2.4.2 Búsqueda dentro de un documento y a través de varios documentos

El estándar del W3C para navegar por un documento XML es el XPath. Oracle 9i permite que las expresiones XPath sean utilizadas para navegar sobre una instancia XMLType y para la búsqueda sobre múltiples instancias de XMLType. XMLTYpe soporta los métodos extract() y existsNode() como método de acceso a los nodos de un documento XML.

Es necesario comentar, que por motivos que desconozco, para que todas las funcionalidad sobre navegación y búsqueda dentro de documentos XML cuando están almacenado como XMLTypes, se debe hacer una pequeña modificación para que estos funcionen, y es que cada libro (etiqueta book) debe estar anidada dentro de una etiqueta bib; esto es, habrán tantos elementos bib como libros haya en el catálogo. Este problema es debido al hecho que la tabla bib que hemos generado no contiene más columnas que la del tipo XMLType.

Veamos dos ejemplos en que utilizando el método extract() podemos obtener información sobre la información de bib.xml almacenada ya como XMLType.

```
SQL> select extract(b.contenido,
  2  '/bib/book/title/text()').getStringVal()
  3  "Titulo libro"
  4  from bib b
```

Y el resultado obtenido será la lista de todos los títulos de los libros:

```
TCP/IP Illustrated
Advanced Programming in the Unix environment
Data on the Web
The Economics of Technology and Content for Digital TV
Oracle 9i.Desarrollo Web
```

Veamos ahora otro ejemplo para ver todos los autores de los libros de nuestro catalogo:

```
SQL> select extract(b.contenido,'/bib/book/author').getClobVal()
  2  "Autores"
  3  from bib b
```

El resultado es:

```
<author>
  <last>Stevens</last>
  <first>W.</first>
</author>
<author>
  <last>Stevens</last>
  <first>W.</first>
</author>
<author>
  <last>Abiteboul</last>
  <first>Serge</first>
</author>
```

```
<author>
  <last>Buneman</last>
  <first>Peter</first>
</author>
<author>
  <last>Suciu</last>
  <first>Dan</first>
</author>
<author>
  <last>Brown</last>
  <first>Bradley</first>
</author>
```

Veamos ahora como funcionaría una consulta con la cláusula where:

```
select extract(b.contenido,'/bib/book').getClobVal()
 "Informacion de libros"
 from bib b
 where extract(b.contenido,'/bib/book/title/text()').getStringVal()='TCP/IP Illustrated';
```

Siendo el resultado:

```
<book year="1994">
  <title>TCP/IP Illustrated</title>
  <isbn>123-1345-44</isbn>
  <author>
    <last>Stevens</last>
    <first>W.</first>
  </author>
  <publisher>Addison-Wesley</publisher>
  <price>65.95</price>
</book>
```

2.4.3 Manipulación de datos XML en columnas XMLType

XMLType permite, además de la posibilidad de invocar una serie de funciones como las detalladas anteriormente, el uso de las operaciones de manipulación de datos como son la inserción, modificación y borrado de datos XML sobre el tipo de datos XMLType.

Inserción de datos XML

Existen dos posibilidad a la hora de insertar datos XML en una tabla, dependiendo si queremos almacenar todo el documento XML o bien insertar un nuevo elemento a la tabla.

En el primer caso, debemos usar un módulo PL/SQL como el utilizado al inicio de este capitulo para recuperar a partir de un archivo XML todo el contenido.

Si lo que queremos es insertar uno o pocos solo elementos XML sobre una tabla, este tipo de inserción de datos XML se realiza mediante el comando SQL INSERT INTO en combinación con la función del XMLTYpe createXML.
Veamos un ejemplo de inserción de datos XML sobre la tabla biblio.

```
SQL> insert into biblio (biblio_id, biblio_dat)
  2  values (2, sys.XMLType.createXML(
  3  '<bib><book year="2003">
  4          <title>Desarrollo de aplicacion con PL/SQL</title>
  5          <isbn>84-456-98-93</isbn>
  6          <author><last>Scardina</last><first>Mark</first></author>
  7          <publisher>Oracle Education</publisher>
  8          <price> 39.50</price>
  9       </book></bib>'));
```

Observemos, tal y como hemos creado la tabla biblio, que lo que estamos haciendo no es insertar un elemento en el documento XML original bib.xml, sino estamos creando un segundo registro en el que hemos incluido en el campo biblio_dat el nuevo "documento" XML que en este caso contiene una sola referencia a un libro.
Observemos cual es el resultado después de hacer una consulta de selección sobre la tabla biblio.

```
<bib>
  <book year="1994">
    <title>TCP/IP Illustrated</title>
    <isbn>123-1345-44</isbn>
    <author>
      <last>Stevens</last>
      <first>W.</first>
    </author>
    <publisher>Addison-Wesley</publisher>
    <price>65.95</price>
  </book>
  <book year="1992">
    <title>Advanced Programming in the Unix
environment</title>
    <isbn>12-1345-XX</isbn>
    <author>
      <last>Stevens</last>
      <first>W.</first>
    </author>
    <publisher>Addison-Wesley</publisher>
    <price>65.95</price>
  </book>
  <book year="2000">
    <title>Data on the Web</title>
    <isbn>34-13445-00</isbn>
    <author>
      <last>Abiteboul</last>
      <first>Serge</first>
    </author>
    <author>
      <last>Buneman</last>
      <first>Peter</first>
    </author>
    <author>
      <last>Suciu</last>
      <first>Dan</first>
    </author>
    <publisher>Morgan Kaufmann
Publishers</publisher>
    <price>39.95</price>
  </book>
  <book year="1999">
    <title>The Economics of Technology and
Content for Digital TV</title>
    <isbn>01-00001-01</isbn>
    <editor>
      <last>Gerbarg</last>
      <first>Darcy</first>
      <affiliation>CITI</affiliation>
    </editor>
    <publisher>Kluwer Academic
Publishers</publisher>
    <price>129.95</price>
  </book>
  <book year="2002">
    <title>Oracle 9i.Desarrollo Web</title>
    <isbn>84-415-1406-2</isbn>
    <author>
      <last>Brown</last>
      <first>Bradley</first>
    </author>
    <publisher>Anaya Multimedia</publisher>
    <price>54.00</price>
  </book>
</bib>

       2
<bib>
  <book year="2003">
    <title>Desarrollo de aplicacion con
PL/SQL</title>
    <isbn>84-456-98-93</isbn>
    <author>
      <last>Scardina</last>
      <first>Mark</first>
    </author>
    <publisher>Oracle Education</publisher>
    <price>39.50</price>
  </book>
</bib>
```

Modificación de datos XML

La modificación de datos XML se realiza mediante la ejecución del comando UPDATE de SQL y utilizando la misma sintaxis que UPDATE permite.

Veamos un ejemplo modificando datos de la tabla inicial bib creada a partir de una sola columna de tipo CLOB y donde hemos almacenado todo el contenido del documento XML bib.xml

```
SQL> update bib b
  2  set b.contenido=sys.XMLType.createXML(
  3  '<book year="2001">
  4          <title>Camino de la Atlantida</title>
  5          <isbn>84-1400-00-87</isbn>
  6          <author><last>King</last><first>Stephen</first></author>
  7          <publisher>Plaza Janes</publisher>
  8          <price>19.00</price>
  9       </book>')
 10  where extract(b.contenido,'/bib/book/title/text()').getStringVal()='TCP/IP Illustrated';
```

Siendo el resultado obtenido el documento modificado:

```
<book year="2001">
  <title>Camino de la Atlantida</title>
  <isbn>84-1400-00-87</isbn>
  <author>
    <last>King</last>
    <first>Stephen</first>
  </author>
  <publisher>Plaza Janes</publisher>
  <price>19.00</price>
</book>

<bib>
  <book year="1992">
    <title>Advanced Programming in the Unix
environment</title>
    <isbn>12-1345-XX</isbn>
    <author>
      <last>Stevens</last>
      <first>W.</first>
    </author>
    <publisher>Addison-Wesley</publisher>
    <price>65.95</price>
  </book>
</bib>

<bib>
  <book year="2000">
    <title>Data on the Web</title>
    <isbn>34-13445-00</isbn>
    <author>
      <last>Abiteboul</last>
      <first>Serge</first>
    </author>
    <author>
      <last>Buneman</last>
      <first>Peter</first>
    </author>
    <author>
      <last>Suciu</last>
      <first>Dan</first>
    </author>
    <publisher>Morgan Kaufmann
Publishers</publisher>
    <price>39.95</price>
  </book>
</bib>

<bib>
  <book year="1999">
    <title>The Economics of Technology and
Content for Digital TV</title>
    <isbn>01-00001-01</isbn>
    <editor>
      <last>Gerbarg</last>
      <first>Darcy</first>
      <affiliation>CITI</affiliation>
    </editor>
    <publisher>Kluwer Academic
Publishers</publisher>
    <price>129.95</price>
  </book>
</bib>

<bib>
  <book year="2002">
    <title>Oracle 9i.Desarrollo Web</title>
    <isbn>84-415-1406-2</isbn>
    <author>
      <last>Brown</last>
      <first>Bradley</first>
    </author>
    <publisher>Anaya Multimedia</publisher>
    <price>54.00</price>
  </book>
</bib>
```

Borrado de datos XML

Para borrar datos XML de una columna del tipo XMLType utilizaremos el comando SQL DELETE. Vamos a eliminar de nuestro catalogo de libros el libro de Stephen King que acabamos de dar de alta:

```
SQL>  delete from bib b
  2   where extract(b.contenido,'/bib/book/author/last/text()').getStringVal()='King';
```

Tratamiento de XML sobre Tamino

1 Introducción

Como ya se ha ido comentando a lo largo de este trabajo, en los últimos años Internet se ha convertido en el estándar de comunicación universal con mayor influencia en todos los campos de la vida cotidiana.
Actualmente Internet también se ha convertido en una parte muy importante de los negocios con la aparición imparable de muchos negocios electrónicos con gran volumen de datos transmitidos y que se deben intercambiar entre empresas. La aparición del estándar XML como lenguaje de intercambio y transmisión universal ha contribuido que cada vez más empresas adopten este tipo de negocios dentro de su estrategia de mercado.

Los datos deben almacenarse ya que son el potencial más grande que toda empresa posee. Ya hemos visto que las bases de datos relacionales han adoptado parches en otros lenguajes de programación para poder trabajar con XML o bien han optado por crear nuevos tipos de datos pero que no ofrecen todas las funcionalidades que otros tipos de datos de este tipo de Bases de Datos tienen. Además el lenguaje de consultas SQL que este tipo de Bases de Datos incorporan no estuvieron diseñados para tratar con XML con lo que se han hecho ciertas adaptaciones sobre el lenguaje para conseguir un resultado más o menos adecuado.
La opción para este tipo de empresas con necesidades de almacenamiento e intercambio de información en el estándar XML son las Bases de Datos Nativas, bases de datos en que su origen, diseño e implementación estuvo desde siempre enfocada al trabajo sobre XML como datos fuente.
Este tipo de bases de datos no solamente sirven para almacenar y manipular datos XML, sino que aportan un paquete integrado de funcionalidades que permiten el acceso a datos y aplicaciones existentes de manera que permite generar aplicaciones e-commerce en que todos estos elementos están integrados.

Figura 5.1. Integración en Tamino de diferentes tipos de datos

El objeto de estudio en este trabajo será la base de datos Tamino de AG Software, servidor de información para datos XML y basado en los estándares abiertos XML; TCP/IP y HTTP.

Tamino permite el uso de distintos tipos de información como texto, tablas, archivos multimedia,… y su integración como documentos XML que podrán ser almacenados y manipulados.

Las principales ventajas que encontramos cuando trabajamos con un sistema de bases de datos nativo son:

Rapidez: como servidor nativo de XML, Tamino excede por mucho el rendimiento de los sistemas de bases de datos que tienen que trabajar con un convertidor de XML.

Extremadamente robusto: Tamino fue diseñado para ejecutar aplicaciones de misión crítica.

Costo eficiente: Tamino está diseñado para minimizar el costo total de propiedad. Todo el sistema se puede administrar desde un solo punto, mediante un "Web Browser" o mediante un cliente "GUI".

Altamente escalable: Tamino puede evolucionar de acuerdo con el número de usuarios y volúmenes de datos.

Seguro: Tamino soporta conceptos flexibles de seguridad en distintos niveles. Combina sistemas estándar de autenticación y criptografiado. También soporta SSL en el nivel de transporte.

Internacional: Tamino soporta el estándar Unicode y por lo tanto provee la base necesaria para aplicaciones internacionales.

2 Arquitectura general de Tamino

La arquitectura del servidor de información Tamino incluye los siguientes elementos:

"X-Machine" es el robusto núcleo de alto rendimiento para el almacenamiento de objetos XML. "X-Machine" administra todas las estructuras de datos bien estén en formato XML o SQL, imágenes, archivos de audio o vídeo o data recuperada a través de "X-Node".

"X-Node" permite el acceso a bases de datos existentes con estructuras de datos tradicionales como Adabas y DB2. "X-Node" proyecta esta data en estructuras XML. Eso le permite a Tamino actuar como un servidor para bases de datos existentes para la RED y para aplicaciones orientadas a la RED. También proyecta la data "tradicional" a estructuras XML, esto le permite a aplicaciones existentes continuar usando esa data.

"SQL Engine" permite el almacenamiento de data SQL en Tamino. Esos datos pueden ser bien parte de documentos XML o utilizados por aplicaciones que solo trabajan con data SQL, pero que ejecutan en el mismo servidor que Tamino.

"X-Manager" permite al administrador de Tamino proyectar todos los objetos XML a estructuras internas y administrar el sistema completo.

"Data Map" contiene meta data como los DTD ("Document Type Definitions"), hojas de estilos, esquemas relacionales, etc. "Data Map" también determina como serán proyectados los documentos XML en las estructuras físicas de datos. Las herramientas de administración de Tamino tienen una sencilla interfaz gráfica para esta proyección, con "Data Map", las bases de datos existentes pueden adecuarse a la tecnología XML y a la RED.

3 Tamino XML Server

3.1 Introducción y concepto

Tamino XML Server es el producto del paquete de software AG que ofrece almacenamiento, mantenimiento e intercambio de documentos XML.
Tamino XML Server es la plataforma de más alto rendimiento de manipulación de datos basada en el estándar XML y ofrece:

- Almacenamiento eficiente de documentos XML nativos, esto es, en su formato original
- Permite la manipulación y publicación de datos provenientes de varias fuentes externas de XML o de documentos no-XML
- Permite búsquedas eficientes sobre la información a la que Tamino tiene acceso.

La filosofía del Tamino XML Server se basa en una ecuación sencilla:
Tamino XML server = Servicios de la Base + Servicios Permitidos + Soluciones

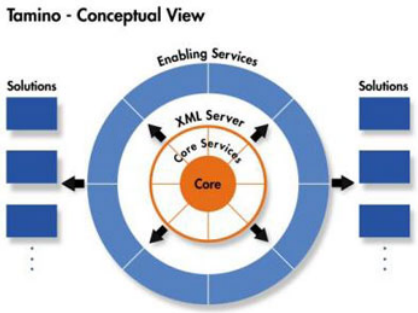


Figura 5.2. Visión conceptual del Tamino XML Server

3.1.1 Servicios de la base (Core Services)

El corazón del Tamino XML server es una base de alto rendimiento que ofrece de mecanismos y funcionalidades para:

- Configurar uno o más sistemas Tamino
- Gestión y mantenimiento de documentos de datos
- Almacenamiento de documentos en formato XML nativo o formatos no XML
- Consultas sobre los datos internos o externos
- Posibilidad de modificación de las funcionalidades dependiendo de los requisitos particulares

3.1.2 Servicios permitidos (Enabling services)

Los enabling services permiten una amplia gama de herramientas y componentes necesarios para el desarrollo de soluciones basadas en XML dirigidas a la Web. Permiten además publicar e intercambiar documentos electrónicos de una manera eficiente.

Estos servicios además permiten el acceso a la información interna o externa de las empresas sin comprometer la seguridad e incluyen las herramientas para el desarrollo jumpstarting de aplicaciones basadas en XML que permiten utilizar el Tamino XML Server como si fuera un sistema de ficheros escribible de la Web (WebDAV)

3.1.3 Soluciones

La capa externa de aplicaciones específicas son las soluciones, esta capa es cada vez versátil y rápida.
Puesto que el Tamino XML Server se basa en estándares abiertos, proporciona, también una arquitectura abierta en la que es fácil agregar nuevos servicios que permiten construir soluciones de gran alcance y a medida para cada situación.

3.2 Arquitectura general de Tamino XML Server

El siguiente grafico muestra como funciona la arquitectura del XML Server. Esencialmente, Tamino consta de dos componentes, el servidor Tamino y los componentes propios del producto.

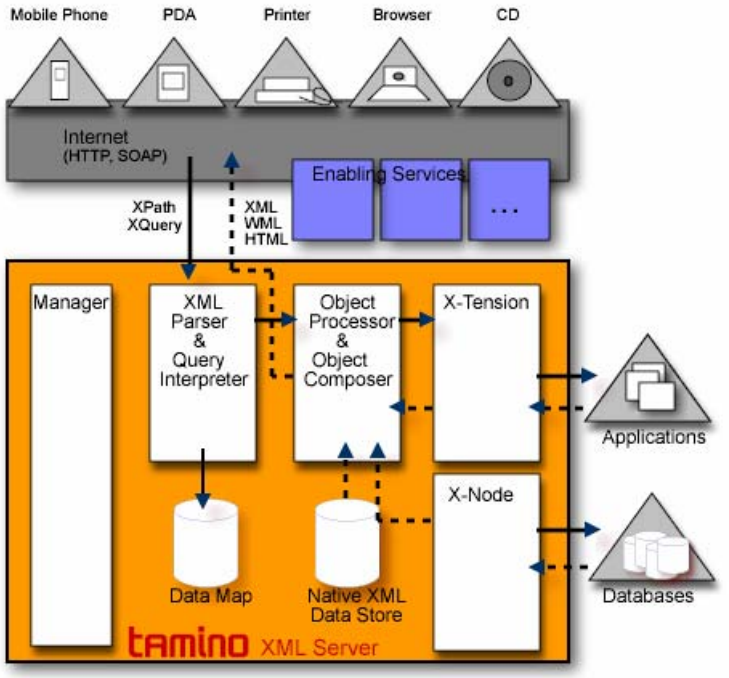


Figura 5.3. Arquitectura general de Tamino XML Server

Los componentes del Tamino XML Server son cinco:

- Tamino Manager
- Data Map
- Native XML Data Store
- X_Tension
- X-Node

3.2.1 Native XML Data Store

Es juntamente con el Motor XML la parte principal de la arquitectura de Tamino, permite el almacenamiento y recuperación de los datos XML, así como la funcionalidad de consultas basada en el XQuery del W3C

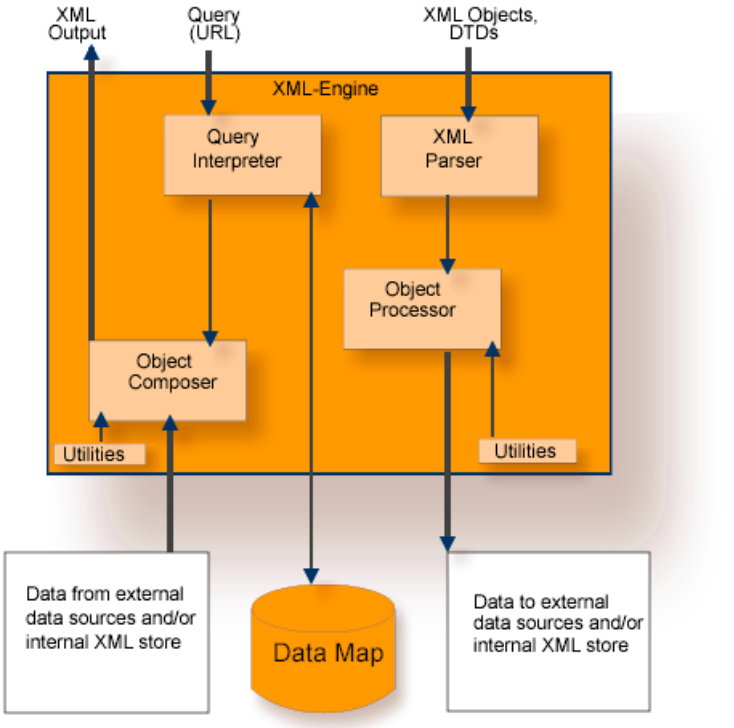


Figura 5.4. Arquitectura y funcionamiento del Native Data Store de Tamino

XML Parser: Los objetos XML que se almacenan a través de la X-Machine o motor XML son descritos por su esquema almacenado en el Data Map. El Parser XML chequea sintácticamente los esquemas comprobando si el documento XML asociado al esquema está bien formado

Object Processor: El procesador de objetos se utiliza cuando se almacenan objetos en XML nativo.

Query Interpreter: Tamino soporta dos lenguajes de consultas sobre datos XML, el Tamino X-Query basado en el estándar XPath y el lenguaje de consultas XQuery recomendado por el W3C. El interprete de consultas de Tamino está formado por el compìlador de Consultas y el motor de ejecución de consultas. El interprete de consultas de Tamino optimiza al máximo las consultas comprobando si los índices dados a partir del esquema en la ejecución de la consulta pueden ser capaces de acelerar la ejecución

Object Composer: El compositor de objetos se utiliza cuando la información XML necesita ser compuesta. Utilizando la información almacenada y las reglas definidas en el Data Map el Object Composer obtiene la información de los diferentes objetos y los devuelve como un documento XML.

3.2.2 Data Map

El Data Map es la base de conocimiento de la base del Tamino XML Server. En el se almacenan metadatos: los esquemas de los documentos XML que permiten decidir si un documento está bien formado o no. Además en el Data Map se almacena información referente a aspectos como el formato de almacenamiento de los datos (si se almacenan de forma nativa o externa), que estructura física se utilizará, aspectos como si se deben generar índices para acelerar las consultas,…

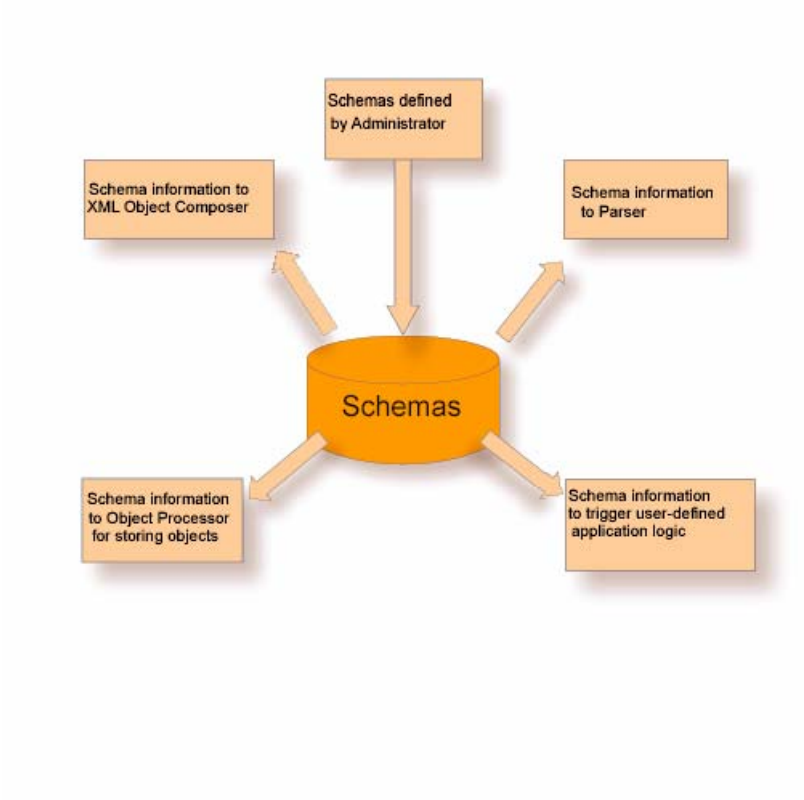


Figura 5.5. Funcionamiento del Data Map de Tamino

El Data Map contiene información necesaria para las funciones siguientes:

- Validación de documentos XML contra su esquema lógico
- Almacenamiento e indexación de las direcciones de los objetos XML dentro de Tamino

La definición de los esquemas en el Data Map se realiza a través de la herramienta gráfica Tamino Schema Editor que permite la creación de objetos XML sin errores sintácticos.

El Tamino XML Server soporta el esquema del W3C dándole gran flexibilidad tanto en la creación como almacenamiento de datos XML permitiendo la creación y almacenamiento de datos XML bien formados (sin una definición explícita del esquema del documento) y de XML válido (que incorpora un esquema)

3.2.3 X-Node

El X-Node es el componente de Tamino que permite la integración de datos almacenados externamente

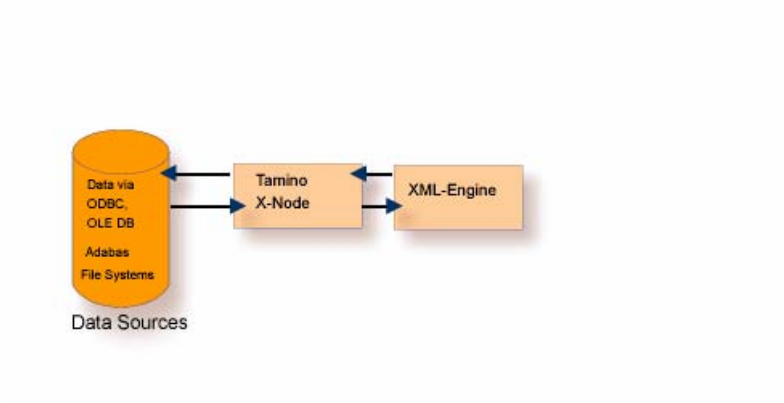


Figura 5.6. Funcionamiento del X-Node

El X-Node permite el acceso a bases de datos heterogéneas sin importarle el tipo de la base de datos o su localización, además el X-Node permite presentar información obtenida de bases de datos o fuentes diferentes y presentarla como si hubiese sido obtenida a partir de una sola base de datos.

3.2.4 X-Tension

El componente X-Tensión de Tamino permite realizar llamadas a las funciones definidas por el usuario llamadas extensiones del servidor (server extensions)

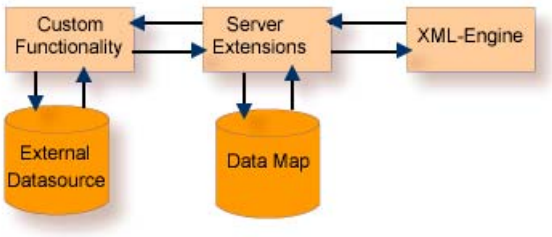


Figura 5.7. Funcionamiento del X-Tension

Las Server Extensions permiten el acceso a aplicaciones externas y permite la creación de funcionalidades soportadas por Tamino XML Server.

Estos plug-ins definidos por el usuario se pueden escribir en Java, C, C++ o en cualquier lenguaje COM-enabled

3.2.5 Tamino Manager

El Tamino Manager es la herramienta de administración de Tamino
Está implementado como una aplicación cliente-servidor e integrada en el System Management Hub, entorno multi-plataforma para la gestión unificada de los productos software de AG.

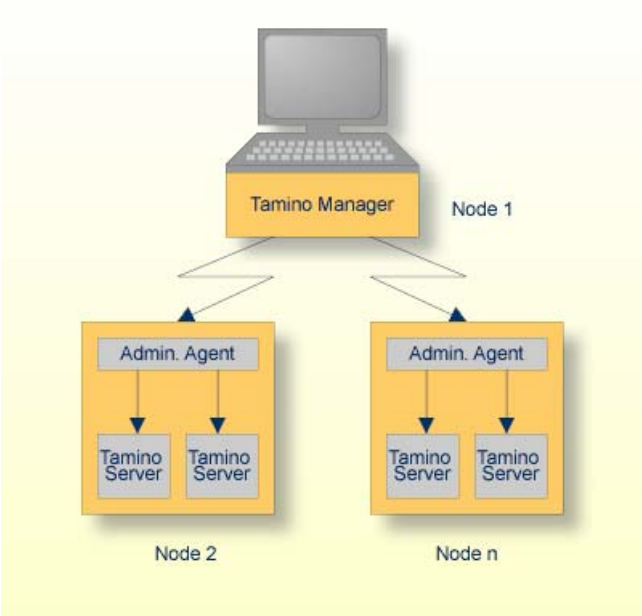


Figura 5.8. Arquitectura y funcionamiento del Tamino Manager

3.3 Componentes y productos de Tamino

Mediante Tamino XML Server y todos los componentes que tiene integrados ya se tiene todos los elementos necesarios para poder ejecutar y configurar completamente el sistema Tamino como servidor.
De todas formas Tamino ofrece una serie de productos independientes que pueden ser descargados desde la página de la Tamino Community Web http://developer.softwareag.com/tamino/default.htm, e instalados en el servidor Tamino para ofrecer y mejorar algunas funcionalidades del sistema.
Destacamos los siguientes productos:

3.3.1 Tamino Schema Editor

El Tamino Schema Editor permite la creación de esquemas. Mediante el interfaz grafico que ofrece evita tener que mecanografiar la sintaxis de los esquemas haciendo que la creación de los esquemas sea mucho mas rápida y menos propensa a errores.

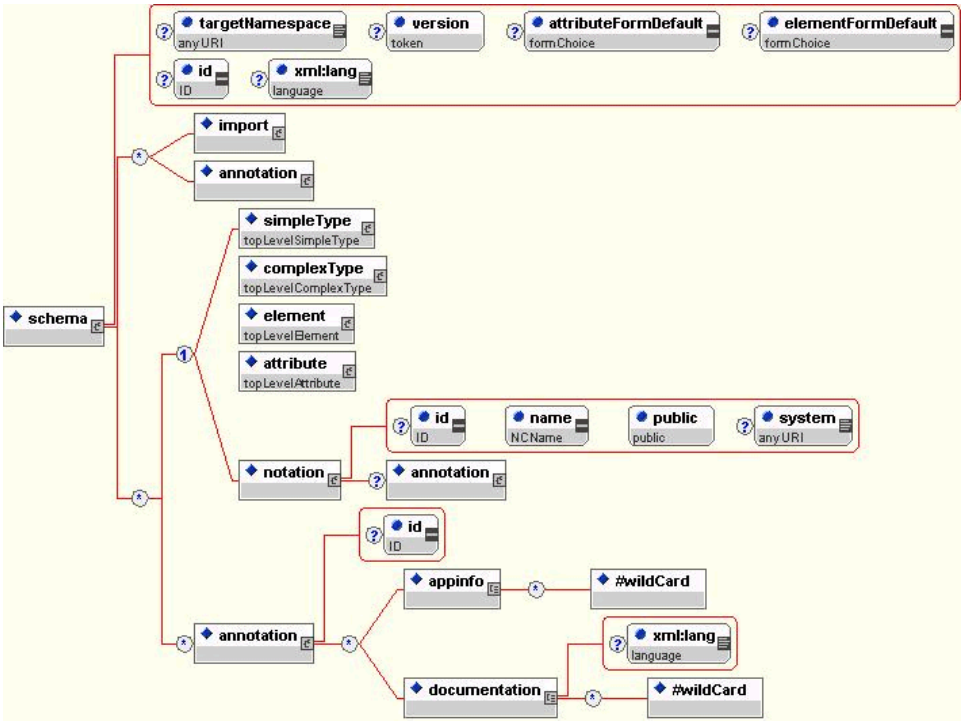


Figura 5.9. Ejemplo de uso del Tamino Schema Editor

3.3.2 Tamino Interactive Interface

El Tamino Interactive Interface es una herramienta que permite definir colecciones dentro de un esquema, cargar instancias XML de un esquema dentro de una base de datos, realizar consultas sobre la base de datos utilizando el lenguaje de consultas estándar del W3C XQuery y el propio lenguaje de consultas Tamino X-Query; permite además el borrado de instancias XML, esquemas y colecciones de la base de datos.

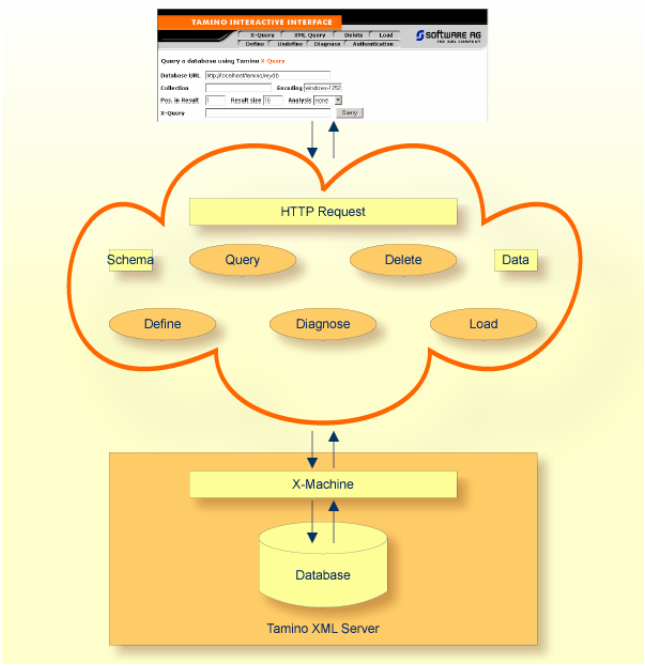


Figura 5.10. Funcionamiento del Tamino Interactive Interface

3.3.3 Tamino X-Plorer

El Tamino X-Plorer a través de su interfaz gráfico permite mostrar el contenido de las bases de datos del Tamino XML Server a través de una presentación en árbol de navegación, permitiendo la exploración de la base de datos así como su manipulación.

Mediante el Tamino X-Plorer se pueden realizar las siguientes funciones:

- Explorar una base de datos de Tamino XML Server
- Realizar consultas sobre una base de datos del servidor Tamino
- Realizar el mantenimiento de la estructura y contenidos del servidor Tamino XML Server
- Mostrar, crear y editar objetos dentro del servidor Tamino
- Ejecutar aplicaciones externas que ayudan en el desarrollo de aplicaciones Tamino (como por ejemplo el Tamino X-Application Generador)
- Se pueden tomar los datos externos como datos de seguridad, extensiones del servidor o como colecciones permitidas WebDAV

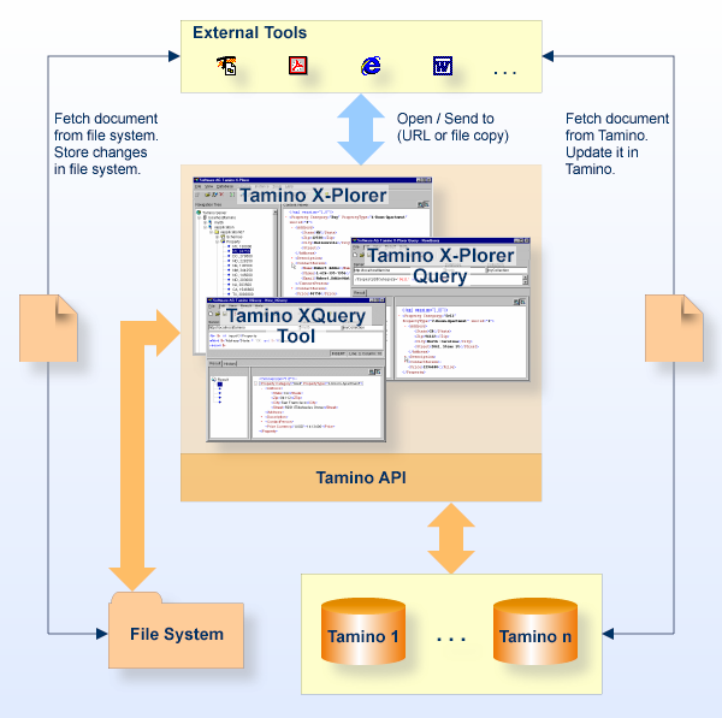


Figura 5.11. Funcionamiento del Tamino X-Plorer

3.3.4 Tamino WebDAV Server

Basado en el protocolo HTTP y WebDAV el Tamino WebDAV Server permite que usuarios de paquetes ofimáticos como Microsoft Office 2000, usando las carpetas de la Web de Microsoft y las herramientas estándares del Office, puedan editar recursos Web con la misma facilidad como si estuvieran trabajando en un entorno local. La principal ventaja que aporta el Tamino WebDAV Server es que los recursos web que son editados por los usuarios son inmediatamente accesibles por los demás usuarios vía Intranet, extranet o Internet.
El estándar WebDAV permite:

- Publicación instantánea en la web
- Trabajo y gestión de grupos de usuarios (Workgroups)
- Gestión y administración de contenidos
- Gestión y administración completa de ficheros

A diferencia que las demás aplicaciones que aquí se están comentando, el Tamino WebDAV Server se ofrece en la instalación completa del Tamino y no es necesario descargarla de la página de desarrollo de Software AG

3.3.5 Interfaces de Programación de Aplicaciones (API's)

Cuando instalamos el Tamino XML Server se instalan las API's (Application programming interfaces) que permiten mediante lenguajes de programación convencionales acceder a algunas de las funcionalidades de Tamino o el acceso a los datos de las bases de datos del Servidor Tamino.
Encontramos las siguientes API's:

- **Tamino API para .NET**
  Es la API de Tamino para lenguajes .NET proporciona un interfaz de programación orientado a objetos para el Tamino XML Server y las aplicaciones .NET. La API está codificada en lenguaje C# y soporta las clases del .NET XML para acceder y manipular documentos XML de una base de datos Tamino.

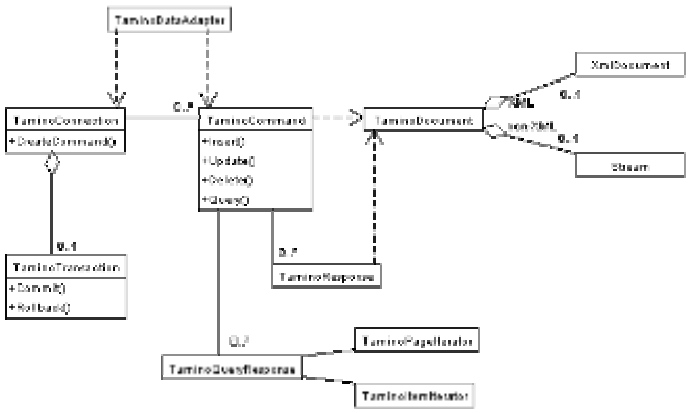


Figura 5.12. Esquema UML de la API para .NET

- **Tamino API para Java**
  La API de Java ofrece el acceso a datos almacenados en Tamino utilizando diferentes modelos de objetos (DOM, SAX, JDOM) o acceso basado en streams. Se incluye además la API Tamino EJB (Enterprise Java Bean) que permite la creación de aplicaciones de negocio utilizando Tamino XML Server como un recurso de gestión en el entorno EJB clásico

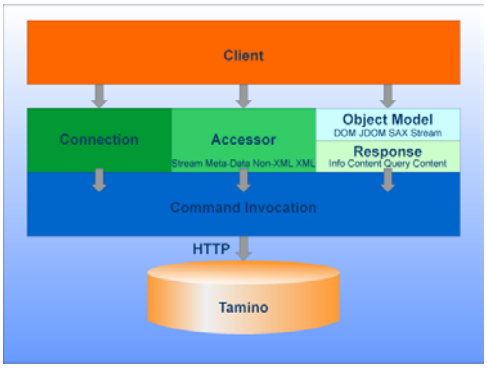


Figura 5.13. Arquitectura de la API para Java

- **API HTTP Cliente para ActiveX**
  API para las plataformas Windows que usando el modelo de objetos DOM permite acceder y manipular los datos almacenados en el servidor Tamino. Se puede utilizar para aplicaciones escritas en lenguajes como C++ o VisualBasic

- **API HTTP Cliente para Java**
  API para Java que utilizando el DOM permite acceder y manipular los datos almacenados en el servidor Tamino

- **API HTTP Cliente para JScript**
  API que utiliza el modelo de objetos DOM para acceder y manipular los datos almacenados en el Servidor Tamino. Se suele utilizar en aplicaciones para navegadores.

3.4 Creación de una base de datos en Tamino

Vamos ahora analizar a la práctica como Tamino realiza la gestión con datos XML, para ello, vamos a retomar nuestro ejemplo basado en el documento XML bib.xml con su correspondiente archivo DTD bib.dtd que ya utilizamos en el capítulo anterior para analizar las herramientas XML de que Oracle dispone.

Lo primero que debemos hacer cuando queremos trabajar con Tamino es crear la base de datos. Esto se hace desde el Tamino Manager, un interfaz gráfico de usuario con el cual podemos realizar funciones de administración como arrancar, detener, renombrar, borrar o restaurar bases de datos.
Durante la creación de la base de datos se debe definir el nombre de la base de datos, la localización y aspectos como el tamaño o las propiedades.

Vamos a ver los pasos necesarios para crear la base de datos para poder realizar las pruebas prácticas sobre Tamino.

Lo primero que debemos hacer al ejecutar el Tamino Manager es autenticarnos mediante un login y un password; este login y password debe tener privilegios de administrador en el sistema operativo ya que la validación siempre se hace a través del sistema operativo.

Una vez autenticados nos encontraremos con una pantalla como la que se muestra:

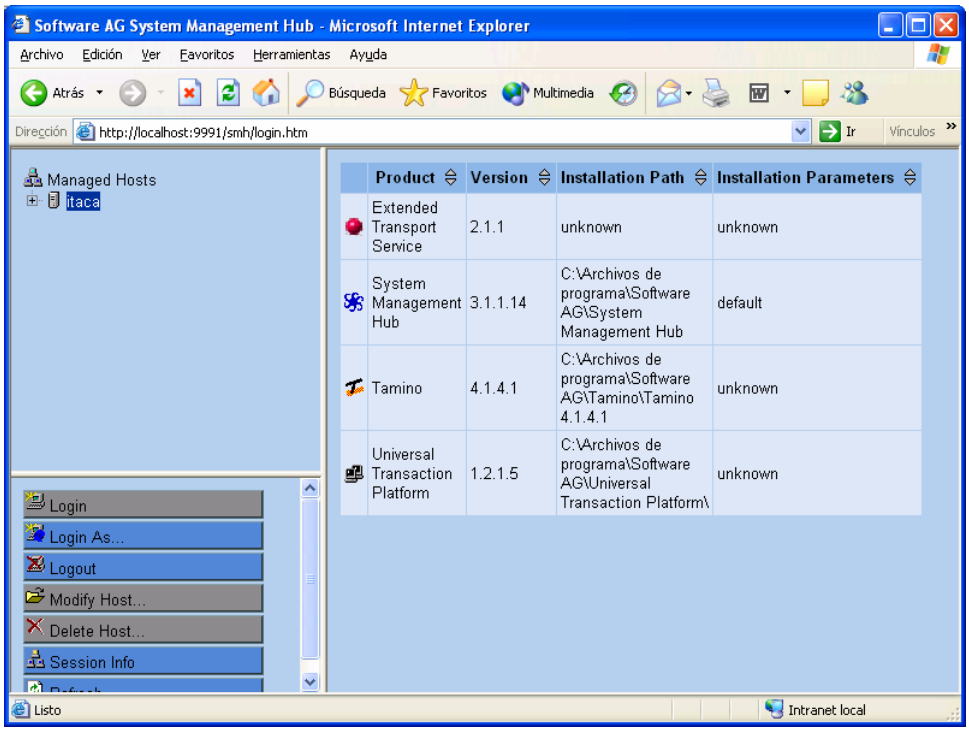


Figura 5.14. Pantalla principal del Tamino Manager (creación de la base de datos I)

Podemos desplegar la línea que muestra el nombre del host que estamos utilizando y, en este caso, dominio de nuestra máquina y tendremos:

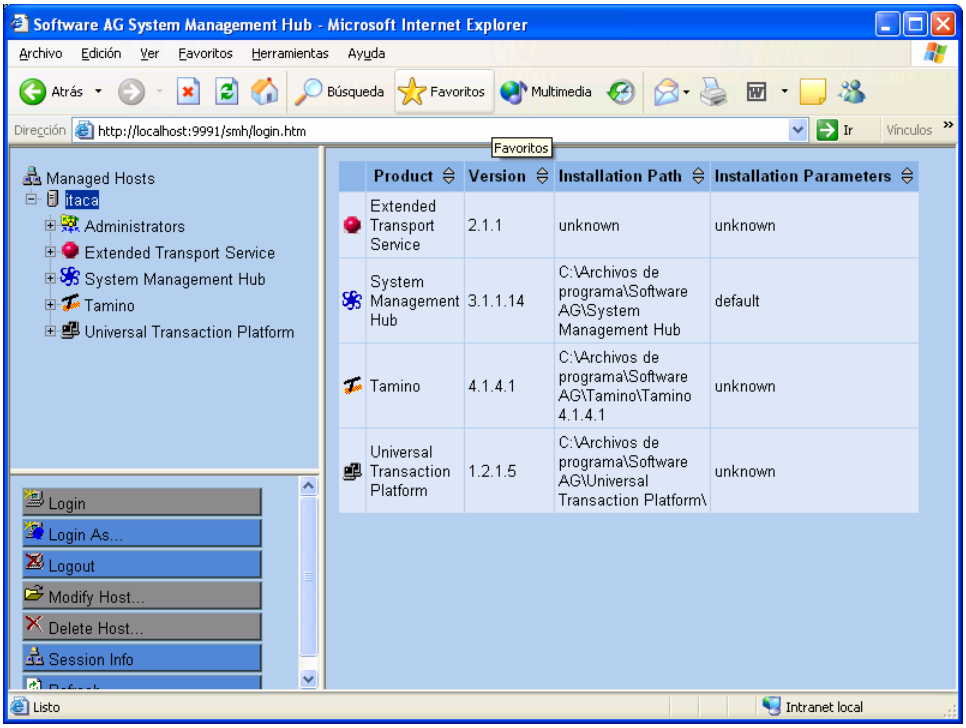


Figura 5.15. Pantalla principal del Tamino Manager (creación de la base de datos II)

Si desplegamos del árbol de navegación la opción Tamino, obtendremos:

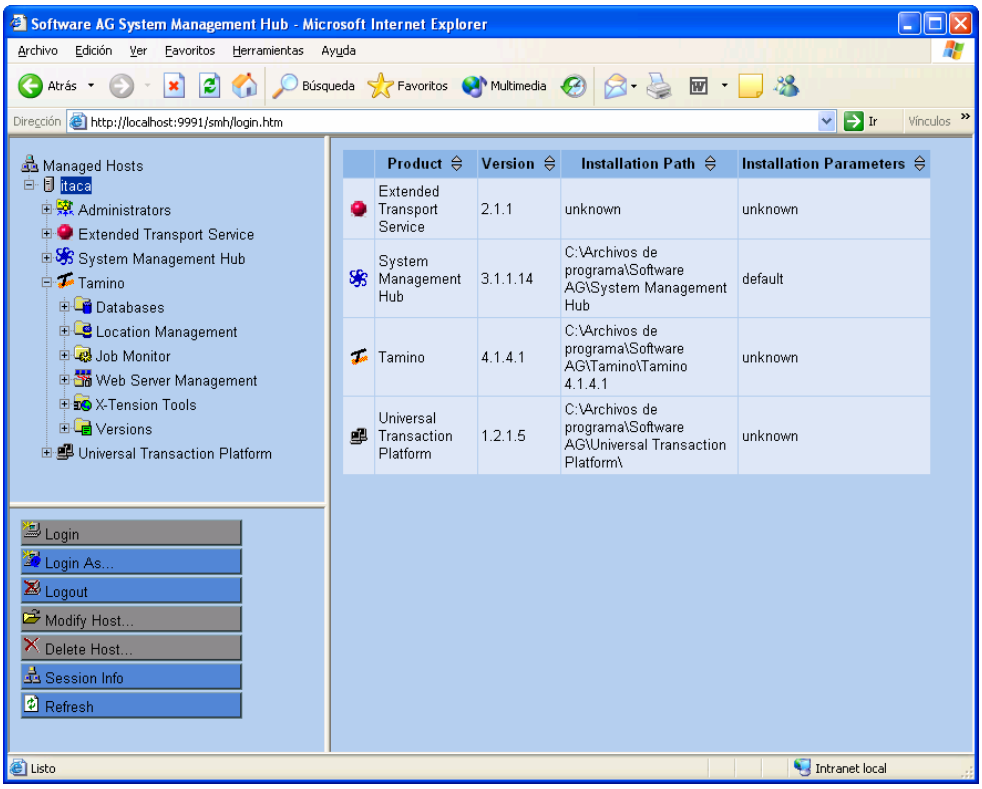


Figura 5.16. Pantalla principal del Tamino Manager (creación de la base de datos III)

Seleccionamos sobre Databases y mediante la opción Create Database creamos la base de datos, en este caso, y para evitar problemas se han seleccionado las opciones por defecto:

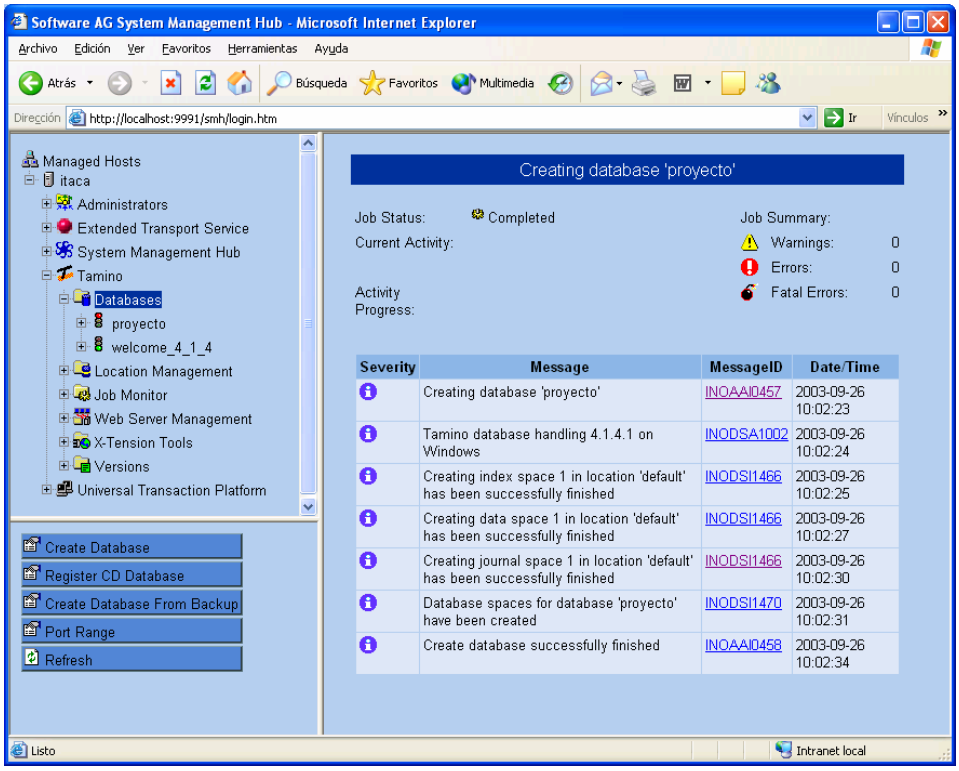


Figura 5.17. Pantalla principal del Tamino Manager (creación de la base de datos IV)

Observemos que tenemos ahora dos bases de datos, la que viene por defecto con la instalación del Tamino y la que acabamos de crear, pero una esta en ejecución y la otra parada. Mediante la selección de la base de datos, aparece en la pantalla inferior izquierda las acciones que podemos realizar sobre la base de datos; así podemos arrancarla, pararla, renombrarla, borrarla,… Seleccionamos Start Database para arrancar nuestra base de datos

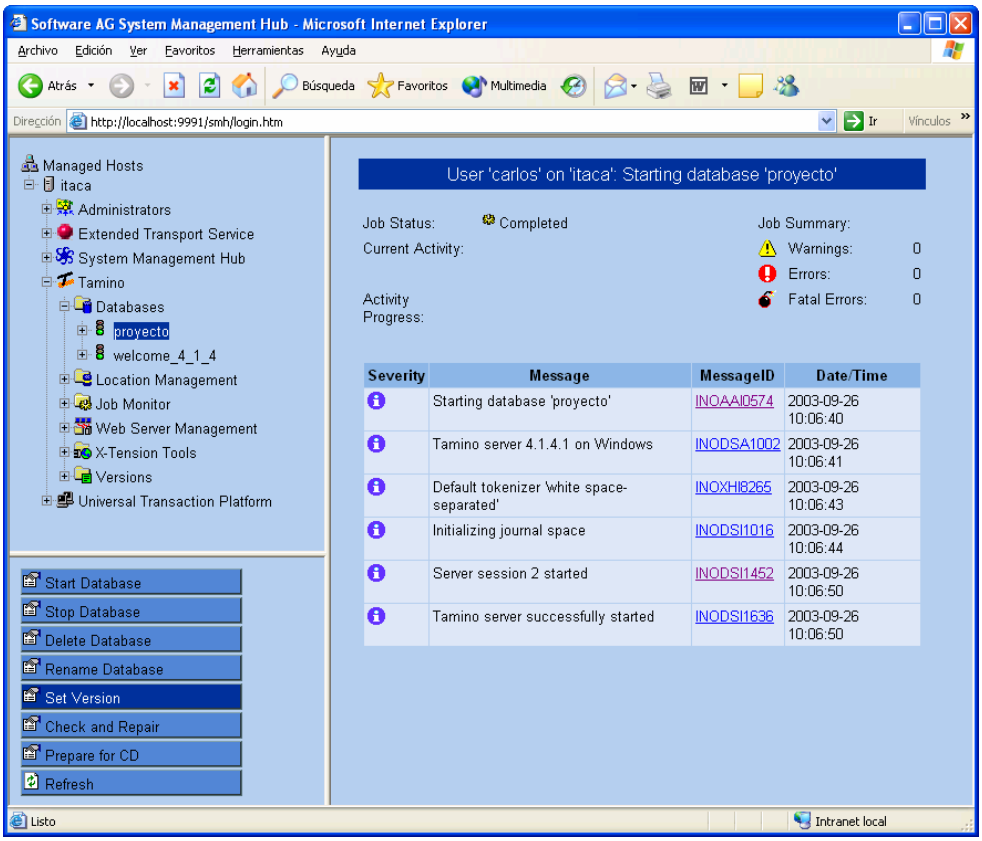


Figura 5.18. Pantalla principal del Tamino Manager (arranque de la base de datos)

3.5 Localización de objetos XML

Lo siguiente que debemos hacer es agregar a nuestra base de datos los objetos XML que queremos incluir en esta base de datos. Sabemos por lo que hemos visto que Tamino se basa en los esquemas de los documentos XML, el principal problema que tenemos es que para el caso práctico que nos ocupa, el documento XML llamado bib.xml, no tenemos definido el esquema y sí tenemos definido el archivo DTD.
Mediante el Tamino Schema Editor, herramienta para crear esquemas, podemos crear un esquema para nuestro ejemplo práctico a partir del documento DTD del que disponemos.
Cuando ejecutamos la aplicación Schema Editor nos encontramos con la pantalla siguiente:

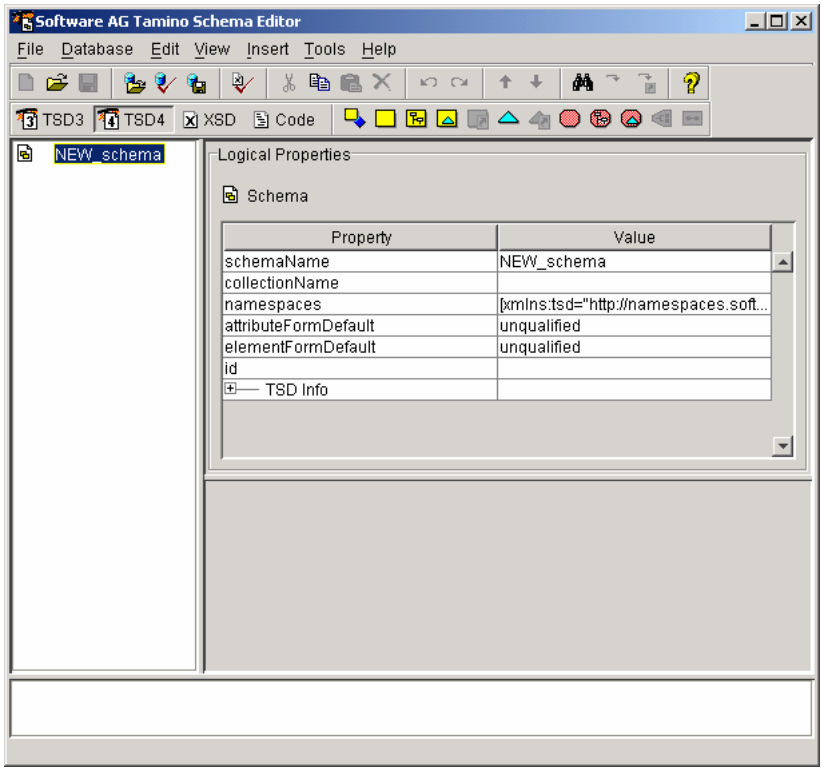


Figura 5.19. Uso del Tamino Schema Editor (creación del esquema de la base de datos I)

Modificamos el nombre NEW_Schema que encontramos como valor de la propiedad schemaName, en esta caso por biblio y especificamos el nombre de la colección. Una vez hecho esto, mediante el menú Insert seleccionamos DocType para insertar el archivo DTD que nos define el archivo bib.xml

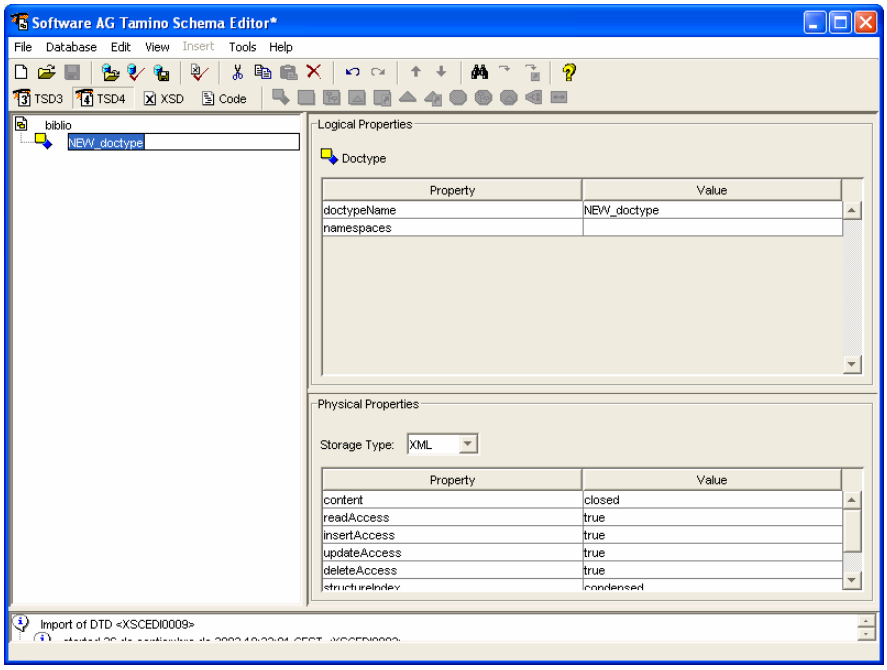


Figura 5.20. Uso del Tamino Schema Editor (creación del esquema de la base de datos II)

Una vez tenemos hecho esto, seleccionamos la opción Import DTD… del menú File para seleccionar el archivo DTD a partir del que queremos generar el schema. Al final de este proceso tendremos:

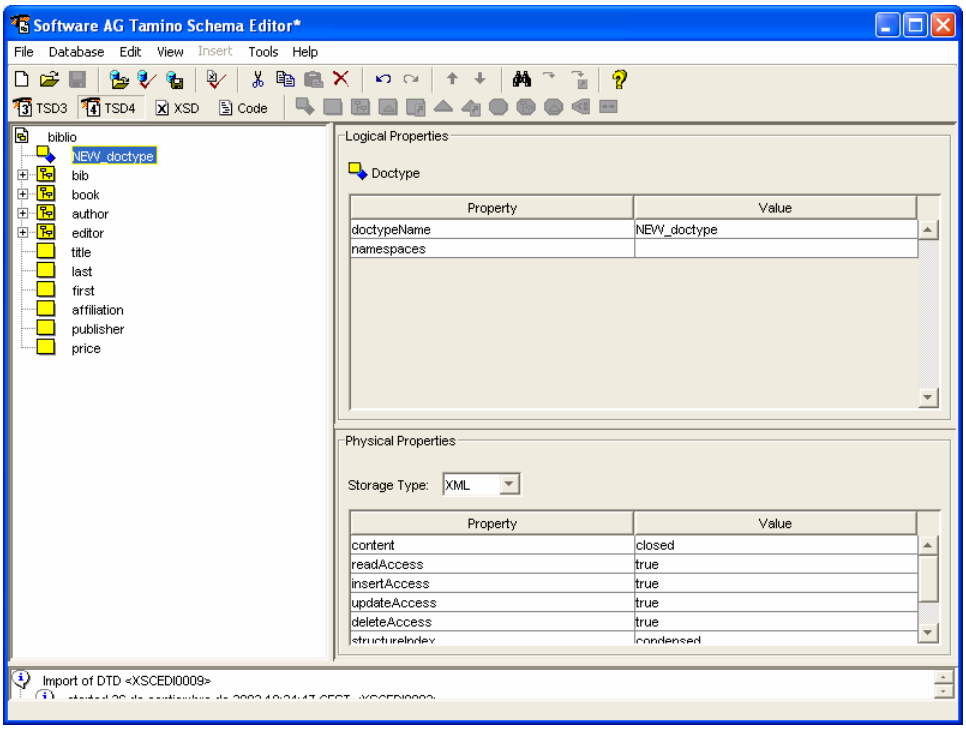


Figura 5.21. Uso del Tamino Schema Editor (creación del esquema de la base de datos III)

Observamos que si desplegamos todos los elementos del arbol de navegación tenemos el archivo bib.dtd representado gráficamente:

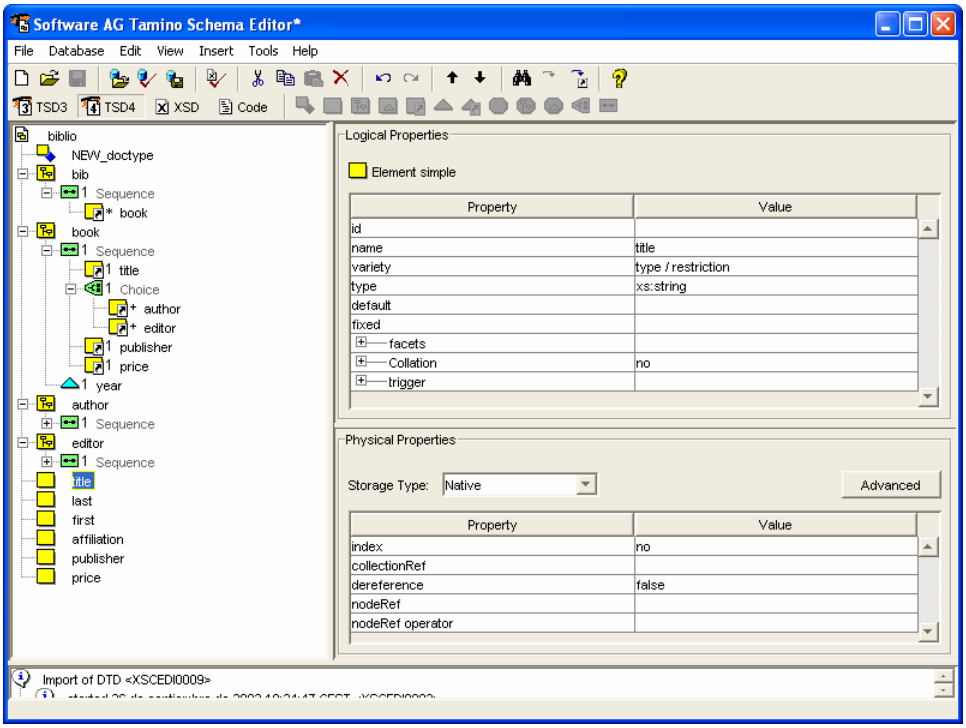


Figura 5.22. Uso del Tamino Schema Editor (creación del esquema de la base de datos IV)

Por ultimo finalizamos el proceso grabando el esquema que hemos generado como biblio.tsd, el contenido del cual es el siguiente:

```
<?xml version = "1.0" encoding = "UTF-8"?>
<xs:schema xmlns:tsd = "http://namespaces.softwareag.com/tamino/TaminoSchemaDefinition" xmlns:xs =
"http://www.w3.org/2001/XMLSchema">
  <xs:annotation>
    <xs:appinfo>
      <tsd:schemaInfo name = "biblio">
        <tsd:collection name = "biblio"></tsd:collection>
        <tsd:doctype name = "NEW_doctype">
          <tsd:logical>
            <tsd:content>closed</tsd:content>
          </tsd:logical>
        </tsd:doctype>
      </tsd:schemaInfo>
    </xs:appinfo>
  </xs:annotation>
  <xs:element name = "bib">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref = "book" minOccurs = "0" maxOccurs = "unbounded"></xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name = "book">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref = "title"></xs:element>
        <xs:choice>
          <xs:element ref = "author" maxOccurs = "unbounded"></xs:element>
          <xs:element ref = "editor" maxOccurs = "unbounded"></xs:element>
        </xs:choice>
        <xs:element ref = "publisher"></xs:element>
        <xs:element ref = "price"></xs:element>
      </xs:sequence>
      <xs:attribute name = "year" type = "xs:string" use = "required"></xs:attribute>
    </xs:complexType>
  </xs:element>
  <xs:element name = "author">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref = "last"></xs:element>
        <xs:element ref = "first"></xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name = "editor">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref = "last"></xs:element>
        <xs:element ref = "first"></xs:element>
        <xs:element ref = "affiliation"></xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name = "title" type = "xs:string"></xs:element>
  <xs:element name = "last" type = "xs:string"></xs:element>
  <xs:element name = "first" type = "xs:string"></xs:element>
  <xs:element name = "affiliation" type = "xs:string"></xs:element>
  <xs:element name = "publisher" type = "xs:string"></xs:element>
  <xs:element name = "price" type = "xs:string"></xs:element>
</xs:schema>
```

Una vez ya tenemos nuestro esquema el siguiente paso es cargar los datos XML a nuestra base de datos Tamino; la manera más sencilla de cargar los datos XML es mediante el Tamino Interactive Interface.

Lo primero que se debe hacer es mediante la pestaña DEFINE, seleccionar el esquema que definirá los datos de nuestra base de datos

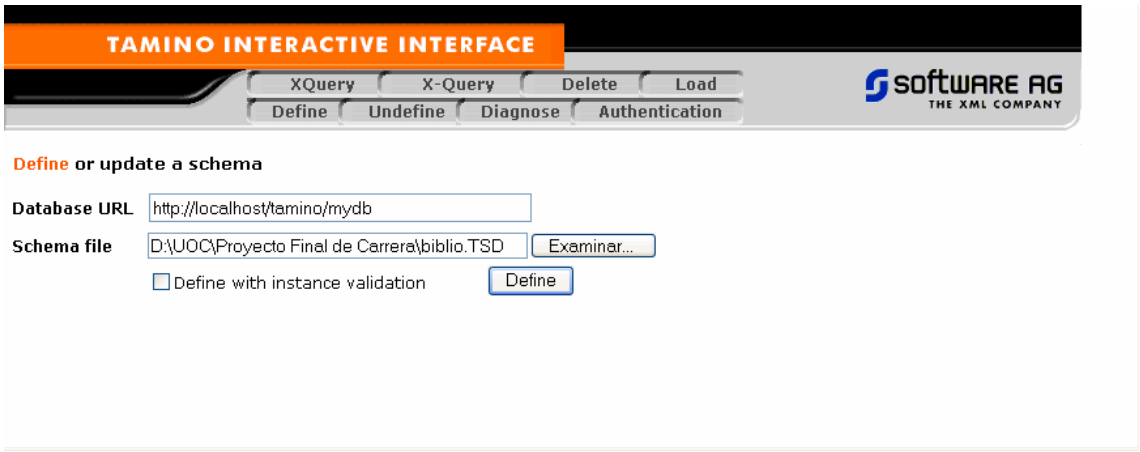


Figura 5.23. Carga del esquema de la base de datos en Tamino Interactive Interface

Una vez seleccionado el botón Define ya tenemos cargado el esquema de nuestra base de datos.

Ahora, mediante la pestaña Load seleccionamos el documento XML que aportarán los datos a la base de datos proyecto, en nuestro caso seleccionamos el archivo bib.xml

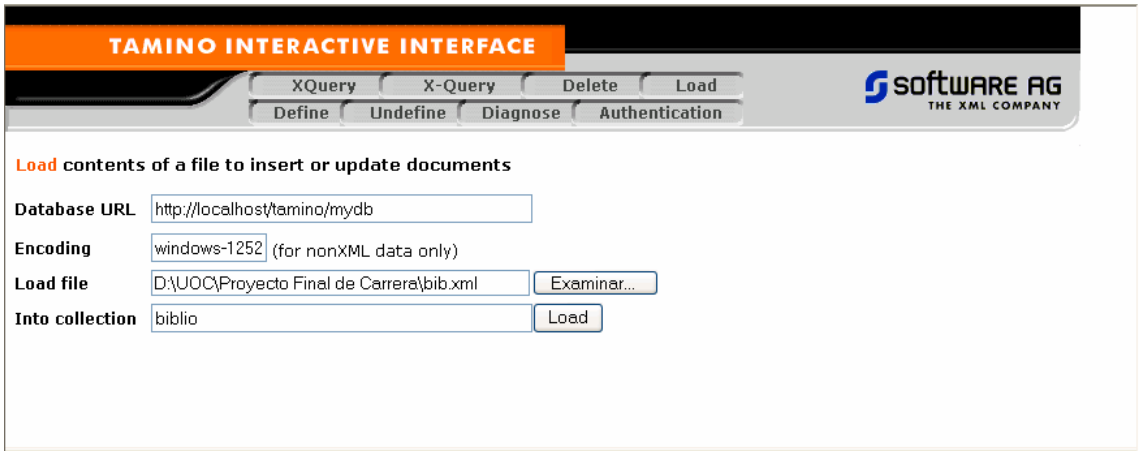


Figura 5.24. Carga del archivo XML de la base de datos en Tamino Interactive Interface

Con esto ya tenemos cargados en nuestra base de datos proyecto los datos del archivo bib.xml.

Para realizar la carga de datos XML a la base de datos también podemos utilizar el comando Data Loader en el que mediante la línea de comandos podemos incluir en nuestra base de datos el documento XML; el comando que ejecuta el data loader es el comando inoxmld al que se deben añadir los parámetros indicados:

```
Usage: inoxmld <params>
      where params are the following :
Function=(Load|Unload|Define)       - can be omitted for Load
Database=<database name>            or
Server=<machine name>:<port>        Tamino Server to connect to - MANDATORY
User=<uid>                          Tamino User-Id - if not provided, anonymous
Password=<password>
Collection=<collection>/<doc-type>   specification of collection name and
                                    doc-type to load data into - MANDATORY
Log=<log filename>                  where to write processing information -
                                    optional - default is STDERR
concurrentWrite                     allow parallel updates
                                    (normally slow, but uses less temporary space)
------ for Data Load ------
Input=<input filename>              file containing documents to load
                                    optional - default is STDIN
Norejects                           do not tolerate rejects
Rejects=<rejects filename>          where to write the rejected documents -
                                    optional - not written if omitted
------ for Data Unload ----
Output=<output filename>            where to write the unloaded documents -
                                    MANDATORY
```

Así si ejecutamos

```
inoxmld Database=proyecto Collection=biblio/bib.dtd Input=d:\bib.xml
```

Tendremos como resultado:

```
C:\Archivos de programa\Software AG\Tamino\Tamino 4.1.4.1\X_Tools\Tamino_Data_Lo
ader>inoxmld Database=proyecto Collection=biblio/bib.dtd Input=d:\bib.xml
<?xml version="1.0" encoding="utf-8" ?>
<ino:response xmlns:ino="http://namespaces.softwareag.com/tamino/response2" xmln
s:xql="http://metalab.unc.edu/xql/">
  <ino:message>
    <ino:messageline>Tamino Data Loader v4.1.4.1 - Copyright (c) Software AG</in
o:messageline>
    <ino:messageline>Loading from d:\bib.xml to Tamino database proyecto</ino:me
ssageline>
    <ino:messageline>Start: 2003-09-26T12:03:31</ino:messageline>
  </ino:message>
  <ino:message ino:returnvalue="7935"><ino:messagetext ino:code="INOXDE7935">Sch
ema not found</ino:messagetext><ino:messageline>Collection name = biblio</ino:me
ssageline></ino:message>
</ino:response>
```

Con lo que ya tendremos cargados en nuestra base de datos los datos XML.

3.6 Consulta y manipulación de la base de datos Tamino

Una vez hemos realizado los pasos descritos anteriormente, ya podemos realizar las consultas y operaciones de manipulación sobre los datos de la base de datos. Para ello podemos utilizar el lenguaje de consultas XQuery 1.0 del W3C descrito en el capítulo 3 de este trabajo o bien utilizar el lenguaje de consultas XML Tamino X-Query basado en las especificaciones XPath del W3C.
Cualquiera que sea nuestra elección a la hora de elegir el lenguaje de consultas que utilizaremos, el mecanismo de consultas es el mismo en ambos casos, accederemos mediante el Tamino Interactive Interface a las pestañas de consultas XQuery (lenguaje de consultas XQuery 1.0 del W3C) o X-Query (lenguaje de consultas Tamino X-Query basado en las especificaciones del XPath del W3C) y escribiremos en la caja de texto reservada para la consulta el texto de la consulta y la ejecutaremos mediante el botón Query.

Así por ejemplo, ejecutamos la consulta siguiente para mostrar el nombre del autor seguido de todos los libros que ha escrito.

```
<authlist>
{
for $a in distinct-values($books)//author
order by $a
return
<author>
<name>
{ $a/text() }
</name>
<books>
{
for $b in $books//book[author = $a]
order by $b/title
return $b/title
}
</books>
</author>
}
</authlist>
```

La ejecutamos sobre el Tamino Interactive Interface y obtendremos en la pantalla resultado:

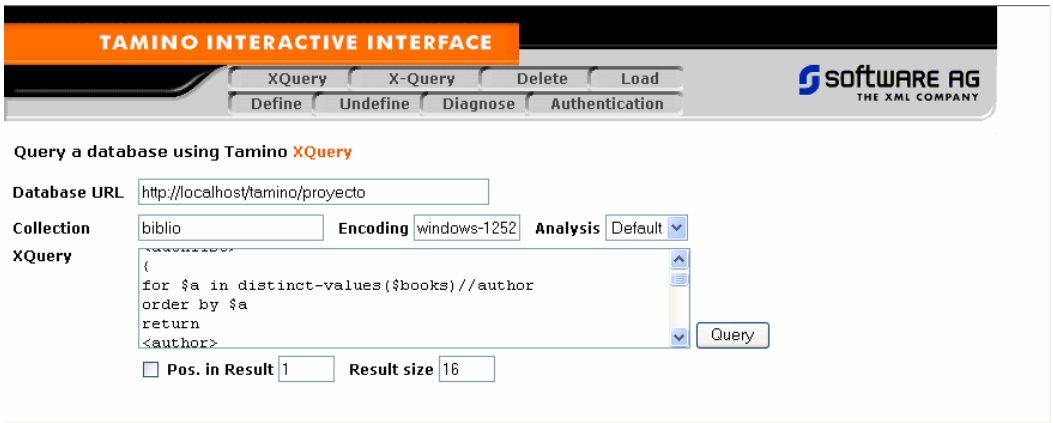


Figura 5.25. Ejecución de consulta XQuery

Si utilizamos una consulta utilizando el lenguaje de consultas Tamino X-Query, el procedimiento es similar, ejecutamos la consulta en que se nos muestren los libros que han sido escritos por más de un autor:

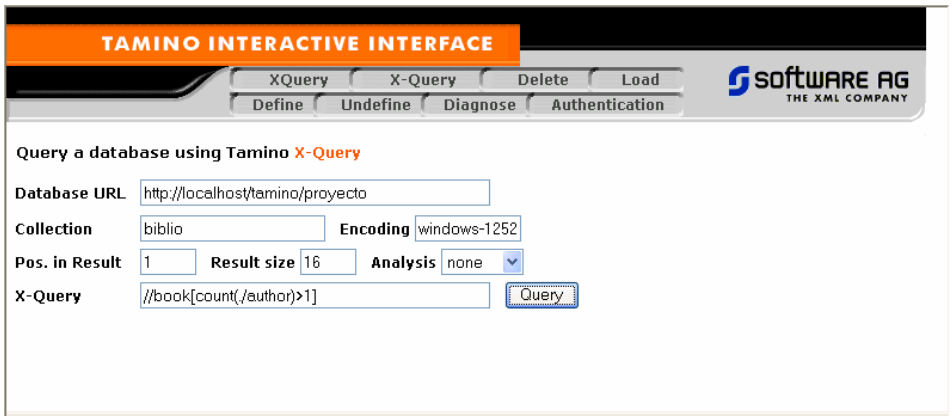


Figura 5.26. Ejecución de consulta Tamino X-Query

Conclusiones y Resumen

1 Introducción

Ya hemos ido viendo a lo largo de este trabajo que XML se está convirtiendo en el formato de datos elegido para una amplia gama de aplicaciones basadas en el intercambio de información vía Internet.
El principal problema que encontramos en otros tipos de formatos de datos, normalmente propietarios, si estos quieren intercambiarse entre distintas organizaciones con sistemas distintos es que ambas organizaciones deben poseer la misma aplicación o sistema que generó esos datos, de otra forma los datos deben adaptarse al sistema de una forma más o menos manual y muy rudimentaria.

XML ofrece la posibilidad que el intercambio de información sea automático ya que gran número de aplicaciones pueden trabajar con este estándar.

La mayoría de aplicaciones que utilizan la red dependen de las bases de datos. Es necesario disponer de las herramientas para poder crear, recuperar y compartir información digital; y por descontado, son las bases de datos las herramientas informáticas que nos permiten crear y sobretodo acceder a la información según unos criterios de eficacia y eficiencia.

La mayor parte de sistemas gestores de bases de datos son relacionales con lo que el proceso de almacenamiento de los datos XML para más tarde poderlos modificar, actualizar o acceder a ellos deben sufrir unas modificaciones o bien unas traducciones con el objetivo de almacenar de una forma consistente y fiable estos datos. Están apareciendo, sin embargo, bases de datos nativas para XML, estas son unas bases de datos que permiten almacenar los datos XML sin necesidad de estas transformaciones necesarias en las bases de datos relacionales.

Los documentos XML pueden diferenciarse en dos categorías bien definidas: documentos XML centrados en los datos y documentos XML centrados en los documentos. Los documentos centrados en los datos son aquellos en que el documento tiene una estructura bien definida y contiene, en la mayoría de casos, datos actualizables, estos tipos de documentos suelen utilizarse como mero transporte de la información; en cambio los documentos centrados en el documento son aquellos en que los datos XML son utilizados para representar documentos.

Así pues, cuando hablamos de sistemas de almacenamiento y acceso de datos XML, estos deben poderse utilizar de manera eficiente con ambos tipos de documentos ya que XML se utiliza en sistemas que manipulan ambos tipos de datos y es difícil encontrar un sistema que solo manipule un solo tipo de estas dos categorías de documentos XML.

La mayoría de los sistemas gestores de bases de datos se enfocan actualmente en gestionar de manera más eficaz y eficiente un solo tipo de formato; así las bases de datos relacionales son mejores para tratar con datos XML centrados en los datos mientras que las bases de datos nativas o sistemas de administración de contenido y documentos son mejores para almacenar datos centrados en el documento.

2 Estrategias para el almacenamiento de datos XML

Cuando queremos almacenar datos de un documento XML en una base de datos; los datos deben ser mapeados a la estructura del documento XML; la forma más simple de mapeado consisten en convertir o mapear el documento completo XML a una columna única en una relación de la base de datos. Sin embargo podemos encontrar estrategias más elaboradas en que se mapea cada elemento del documento XML y se hacen corresponder a una columna de la tabla de la base de datos.

Actualmente las estrategias que existen para almacenar documentos XML en las bases de datos son tres:

1. Almacenamiento del documento completo XML como texto en un objeto CLOB o BLOB en la base de datos relacional
2. Modificación del documento XML y almacenamiento en el sistema de archivos
3. Mapeado de la estructura del documento XML en la base de datos.

2.1 Almacenamiento del documento XML como un objeto CLOB

Esta estrategia es una buena opción cuando el documento XML contiene información estática que solamente será modificada cuando el documento completo sea reemplazado. Esta opción es la que Oracle 9i y los dos grandes sistemas gestores de bases de datos relacionales como Microsoft SQL Server y IBM DB2 aportan como solución para el almacenamiento de los datos.
El almacenamiento de datos XML siguiendo esta opción es sencilla de implementar ya que no necesita realizar ningún mapeo sobre el documento pero presenta deficiencias a la hora de realizar búsquedas de información dentro del documento o indexación de datos.

2.2 Modificación del documento XML y almacenamiento en el sistema de archivos

Esta opción suele utilizarse cuando el número de documentos XML es pequeño y difícilmente la información que contiene es actualizable.
Esta opción tiene sus limitaciones en cuanto a escalabilidad, flexibilidad a la hora del almacenamiento y recuperación y obviamente deficiencias de seguridad ya que los datos se almacenan fuera de la base de datos como archivos externos.
Al igual que la opción anterior, los tres grandes sistemas de bases de datos (Oracle 9i, MS SQL Server y DB2 de IBM) permiten esta estrategia de almacenamiento.

2.3 Mapeado de la estructura de datos del documento y almacenamiento en la base de datos

La idea principal de esta opción es observar la estructura del documento XML y los elementos que aparecen en el documento y almacenar los datos correspondientes a estos elementos del documento en diferentes columnas de la tabla que representará el ítem o elemento principal del documento; así tendremos la siguiente tabla de conversión entre elementos u objetos del documento XML y elementos u objetos de la base de datos relacional:

Correspondencia de elementos XML a elementos relacionales	
Documento XML (Esquema)	**Esquema Relacional**
Elemento	Tabla
Atributo o Elemento Anidado	Columna
Atributo ID	Clave Primaria
IDREF o Elemento Anidado	Clave Secundaria
#REQUIRED, #IMPLIED	NULL, NOT NULL

3 Oracle 9i y XML

Oracle 9i es una de las bases de datos líderes dentro del mundo de las bases de datos relacionales y a partir de la versión 8i ya incorpora soporte para XML aportando herramientas para el uso de datos XML en las bases de datos.
Oracle 9i permite que un documento XML pueda ser almacenado como una única columna en la base de datos pero con ciertas limitaciones a la hora de realizar consultas e indexación de los datos. Además, Oracle 9i también ofrece herramientas para el particionamiento de documentos XML en columnas y tablas de la base de datos y también permite el almacenamiento de documentos XML como archivos externos a la base de datos. Como vemos, Oracle 9i aporta en su producto las tres estrategias básicas de almacenamiento de datos XML.

Oracle 9i también aporta una utilidad SQL XML para Java, la cual se constituye de una serie de clases Java que permiten la inserción de datos XML en tablas o vistas. Estas clases Java también permiten generar documentos XML a partir de consultas SQL sobre la base de datos.
Como hemos visto también la utilidad SQL XML y todas las clases Java pueden ser envueltas por PL/SQL y incluirse las funcionalidades dentro de aplicaciones, procedimientos u otros objetos programados en SQL y PL/SQL.

La utilidad Servlet Java XSQL también incorporada en el producto, permite que la ejecución de consultas SQL retornen el resultado como un documento XML y a partir de este documento XML realizar una transformación utilizando las hojas de estilo que nos den como resultado un documento HTML.
Estas utilidades Java que Oracle aporta son correctas en su uso cuando el usuario tiene amplios conocimientos en el lenguaje Java y conoce de una manera suficiente el sistema Oracle.

Oracle 9i también aporta el tipo de datos XMLType como dato elemental de las bases de datos, lo que nos permite almacenar en una misma tabla del modelo relacional datos propiamente relacionales y datos XML. En relación a este tipo de datos, Oracle 9i incluye funciones y métodos SQL para poder acceder y modificar estos tipos de datos.

4 Tamino y XML

Tamino de Software AG es el producto representativo de las bases de datos nativas, esto son, bases de datos que para almacenar los datos XML en la base de datos no necesitan traducirse a la estructura relacional sino que se almacenan como datos estructurados XML.
Tamino XML Server ofrece un almacenamiento, mantenimiento, publicación e intercambio de documentos XML. Combinado con las herramientas adecuadas, Tamino ofrece potentes capacidades de gestión, búsqueda y recuperación de datos.

Tamino nos ofrece tres funcionalidades importantísimas en cuanto a la gestión de documentos e información XML, Tamino XML Server tiene la funcionalidad de servidor de presentación , servidor de contenidos y repositorio de datos XML

Tamino permite trabajar con XQL (eXtensible Query Language), un lenguaje de consultas similar al lenguaje de consultas SQL pero destinado a realizar consultas sobre datos XML; es importante destacar que Tamino ofrece la posibilidad de realizar consultas utilizando el lenguaje XQuery basado en el estándar XQuery 1.0 del W3C o bien utilizando el lenguaje de consultas X-Query basado en el estándar XPath 2.0 del W3C.

Es importante destacar que para trabajar con Tamino se debe primero describir los datos que se han de almacenar mediante el esquema del documento o el archivo de definición de datos (DTD), aspecto que no se da en Oracle, aunque si se aportan las herramientas necesarias para detectar si un documento XML es valido o está bien formado.

Tamino permite almacenar casi cualquier tipo de documentos incluyendo entre otros: documentos XML, páginas HTML, plantillas de cálculo, audio, vídeo, imagen y datos de bases de datos relacionales.
Gracias a esta característica de posibilitar el almacenamiento de datos como XML nativo o utilizando el mecanismo de almacenamiento relacional, es posible mediante el lenguaje XQL poder consultar datos heterogéneos y obtener los resultados de estas consultas en formato XML.

Las principales ventajas que aporta disponer de una buena gestión de información y documentos XML mediante un sistema nativo como puede ser Tamino son entre otras que:

- Permite y simplifica la reutilización de los contenidos y de fragmentos de contenido.
- Elimina las saturaciones y capacita a los propietarios de la empresa para crear contenido, lo que reduce la carga de los administradores de Web.
- Garantiza que la marca global se proyecta de una manera coherente mediante plantillas estándares y un procesamiento del flujo de trabajo: el contenido es propiedad de la unidad empresarial, no del administrador de Web.
- Sirve mejor a las audiencias objetivo a través de una gestión de contenidos multilingüe: localización de sitios.
- Crea relaciones más eficaces gracias a un contenido específico para el usuario: perfiles de los visitantes, personalización.
- Aumenta la capacidad de respuesta debido a un acceso a la información mejorado.
- Potencia el intercambio del conocimiento de una manera eficaz y reduce esfuerzos duplicados.
- Sitúa los recursos de contenido necesarios para un trabajo eficaz allí donde los empleados los necesitan.
- Permite la distribución de contenido a través de múltiples canales, por ejemplo, Web, teléfono fijo y teléfono móvil mediante una fuente de contenido única.
- Habilita información de primera calidad y en modo autoservicio para clientes y socios.

5 Conclusiones

A la pregunta sobre que sistema elegir a la hora de almacenar datos XML, si elegir Oracle 9i o elegir Tamino, lo primero que debemos plantearnos es que Oracle 9i, ofrece un gran producto como sistema de gestión de bases de datos relacionales, pero es eso en definitiva, un sistema para bases de datos relacionales y no para datos XML; característica que si aporta Tamino de AG.

Se ha visto durante el análisis práctico de los dos productos que Oracle ha ido incorporando a partir de la versión 8i herramientas o "parches" para poder trabajar más o menos decentemente con datos XML, pero en definitiva solo son eso, parches. La mayor parte de las utilidades que Oracle ofrece para poder trabajar sobre datos XML son clases externas en Java o en C pero que no son parte integrante del producto Oracle; en cambio las herramientas que Tamino ofrece para poder trabajar sobre XML son partes integrantes del propio producto.

Por otra parte, las soluciones de almacenamiento de datos XML que ofrece Oracle no son del todo adecuadas, y ya se ha visto que todas ellas, aunque totalmente funcionales presentan alguna limitación, siendo la principal que el documento XML se almacena en un solo campo al estilo de los tradicionales campos "memo" de otras bases de datos; es importante destacar las limitaciones que este tipo de almacenamiento tiene de cara al acceso y consulta de determinados datos, no ocurriendo esto cuando se quiere acceder a todo el documento

Oracle, a partir de la versión 9i incorpora como tipo de dato elemental el XMLType pero esto solo contribuye a seguir almacenando los datos XML en una sola columna como un campo del tipo CLOB, cuando realmente lo interesante y efectivo sería poder almacenar los datos como lo que son: datos XML; otra característica que si aporta Tamino.

En definitiva, si debemos comparar Oracle 9i versus Tamino para poder trabajar y hacer un tratamiento completo y efectivo de datos XML es Tamino quien vence y ofrece las mejores características y funcionalidades; porque claro está, Tamino está diseñado especialmente para este fin.

Por otra parte debemos centrarnos y comentar aspectos más prácticos como instalación, configuración y manejo de los productos. Disponemos de un gran producto como es Oracle 9i en que la instalación y configuración del producto es relativamente sencilla y su uso, aun cuando trabajamos sobre datos XML no es excesivamente complicado si se conoce un poco el producto. Además, en caso de encontrarnos con problemas cuando utilizamos Oracle 9i disponemos o podemos encontrar gran cantidad de documentación sobre cualquier posible aspecto sobre el que queramos indagar. En cambio Tamino no es sencillo de instalar, ni mucho menos de configurar y por descontado de utilizar. Uno de los principales riesgos que corríamos al plantearnos este trabajo era la poca información, documentación y sobretodo costumbre de trabajar con un producto como Tamino y este riesgo se ha cumplido al 100x100. Tamino ofrece multitud de funcionalidades y herramientas que en este trabajo no se han podido analizar bien por falta de tiempo, bien por problemas con el propio producto, quedando el análisis del producto en un análisis más teórico que práctico.

Para alguien como yo, más diestra en el manejo de Oracle que cualquier otro sistema de bases de datos y con conocimientos en el lenguaje Java y el lenguaje PL/SQL, la realización del análisis sobre Oracle no ha representado un excesivo trabajo, nada que ver a la hora de centrarme en el análisis de Tamino.
Quizás, si este trabajo lo hubiera realizado una persona que no hubiese tenido contacto con Oracle en su vida esta conclusión sería diferente.

La utilización de XML como estándar de transmisión e intercambio de información continuará creciendo, solo hay que ver para darnos cuenta de esto, la gran repercusión que tiene este estándar en algunas de las asignaturas del plan de estudios del segundo ciclo de la ingeniería informática, lo que nos muestra hacia donde tienden las tecnologías de la información actualmente.
Este crecimiento va a provocar, obviamente que la necesidad que las bases de datos manejen este tipo de datos continúe creciendo.
Posiblemente no pasará mucho tiempo hasta que las grandes compañías de bases de datos relacionales, como Oracle, ofrezcan herramientas y funcionalidades nativas de almacenamiento y manipulación de datos XML conjuntamente con las capacidades SQL y relacionales que actualmente ya ofrecen.
Observamos que la incorporación de herramientas para el almacenamiento de datos XML y su posterior recuperación es signo claro de la progresión que se está haciendo hacia esta nueva arquitectura.

Pero hasta entonces, la mejor opción para almacenar documentos XML y poderlos manipular y gestionar de una manera eficaz son las bases de datos nativas como Tamino a pesar de su enorme complejidad.

6 Posibles líneas de trabajo

Este proyecto, desde el principio se planteó como un proyecto inicial en el que se debía determinar el estado del arte en que se encuentra el lenguaje de marcas XML así como los lenguajes de consultas, y eso ha pretendido ser, se ha hecho una recopilación de información sobre el XML y todos los estándares vinculados a él, se ha hecho un análisis y una comparativa sobre los XQuery Languages en general y se ha profundizado sobre el lenguaje de consultas XQuery 1.0 y por último se ha analizado como dos sistemas de bases de datos tan distintos como Oracle 9i y Tamino permiten trabajar con datos XML viendo las principales diferencias entre ambos.

¿Cuáles deberían ser las líneas de trabajo de cara al futuro? Durante el trabajo se ha recabado información, suficiente, espero, sobre el estándar XML y todos los elementos relacionados con éste, estándares de transformación, elementos de sintaxis,… Observemos que ya este estándar y su sintaxis, como los lenguajes de transformación o las hojas de estilo en cascada ya se han ido incorporando a los planes de estudio de una carrera como esta, el segundo ciclo de ingeniería informática, lo que denota la importancia que está tomando el XML como lenguaje de integración e intercambio de información en campos tan dispares como las redes inalámbricas o el mundo de los compiladores, dos de las asignaturas de este plan de estudios donde se incorporan prácticas utilizando el XML.

Quizás el aspecto más novedoso y menos investigado es sin duda la relación del XML con las bases de datos. Actualmente solo se estudia el XML como metalenguaje de intercambio y transporte de la información, pero no se hace especial incidencia en el almacenamiento y posterior recuperación, siguiendo unos criterios de eficiencia y eficacia, de esta información. Así, actualmente se trabaja con el lenguaje XSLT como lenguaje de transformación y recuperación del XML, pero no mediante bases de datos, sean del tipo que sean.

En el campo de las bases de datos suele trabajarse con sistemas gestores de bases de datos relacionales (Oracle 9i, MS SQL Server, DB2,…), por otro lado, adecuada esta opción ya que son los sistemas más extendidos actualmente. Pero aún así, es difícil trabajar el tema de bases de datos relacionales y su relación con el XML. Quizás una propuesta de trabajo futura podría ser el análisis de cómo las principales bases de datos relacionales trabajan y permiten manipular XML. Ya hemos visto que la mejor opción, por prestaciones y funcionalidades es una base de datos nativa, pero esta idea no es tan descabellada si pensamos en el fuerte asentamiento que este tipo de bases de datos tienen tanto en el mundo empresarial como en otros ámbitos. Posiblemente empresas que ya tienen un sistema de bases de datos basado en el modelo relacional optarán por continuar con estas bases de datos incorporando las herramientas y utilidades que permiten para poder manipular información XML.

Por último, destacar que, sin lugar a dudas, una propuesta firme que se lanza desde aquí es un trabajo más profundo y riguroso dedicado a las bases de datos nativas en general y Tamino en concreto. Este es, sin duda, el apartado de este trabajo introductorio que en menos profundidad se ha podido trabajar y que pienso será un punto vital en el momento que cada vez más empresas utilicen el XML como lenguaje de integración e intercambio y se vean en la necesidad de almacenar esa información sin largos, tediosos y costosos procesos de integración a sus respectivos sistemas informáticos.

Recalcar también, la importancia que tiene el poder disponer de un lenguaje de consultas propio como el XQuery para poder realizar los accesos y manipulación de una manera más efectiva que hacerlo mediante el lenguaje de consultas SQL, el cual no está diseñado para la manipulación de este tipo de información.

Hemos visto en relación a los lenguajes de consulta que Oracle 9i aporta "soluciones" de dentro del estándar SQL para poder acceder a datos almacenados en formato XML, pero en el fondo, solo es esto, pequeños apaños que se han hecho sobre este lenguaje. Tamino, a diferencia de Oracle, aporta la posibilidad de manipular los datos mediante el lenguaje de consultas XQuery del W3C además de aportar el propio lenguaje de consultas X-Query de Tamino, un lenguaje de consultas basado en el XPath 2.0. Quizás comprobar las diferencias que estos dos lenguajes presentan en el acceso y manipulación de datos XML también sería una buena propuesta de trabajo de cara al futuro: analizar si es mejor utilizar el estándar del World Wide Web Consortium o bien el propio lenguaje de consultas de Tamino.

Como vemos, quedan a partir de este trabajo varios caminos de estudio abiertos, todos ellos innovadores en cuanto al mundo de las bases de datos y el XML y estos son en definitiva, aquellos que a partir de ahora deberían desarrollarse como futuras propuestas de proyectos de final de carrera.

Bibliografía y recursos

Bibliografía y recursos sobre XML

"Iniciación a XML"
David Hunter y otros
Inforbooks Ediciones y WROX Programmer to Programmer, 2001

"Compiladors II. Apunts de l'assignatura"
David Megías y Julià Minguillón
Universitat Oberta de Catalunya, 2003

"XML Lenguaje de Marcas. Trabajo Final"
Desconocido
Universidad Católica de Salta, 2001

"Recuperación de información en la web"
Ricardo Baeza Yates, Meter Schäuble
Novatica núm. 157 Mayo-Junio 2002

"XML como arquitectura de sistemas de información"
Pedro Pastor, Universitat d'Alacant
URL: http://nereida.deioc.ull.es/~paco/ib2001/Pedro.pdf

Programa y materiales de la Asignatura Marcado de Textos
Departament de Llenguatges i Sistemas Informàtics. Universitat d'Alacant
URL: http://www.dlsi.ua.es/assignatures/doctorat/mt/

"XML RoadMap"
AQS Advanced Quality Solutions, Febrero 2001
URL: http://www.it.uc3m.es/~xml/documentos/xml-roadmap.pdf

"Lenguajes de consulta para XML. Programación XML"
Daniel Gayo Avello
Universidad de Oviedo
URL: http://facartes.unal.edu.co/posmulti/saxDom.PDF

"Problemática y tendencias en la arquitectura de metadatos web"
Pedro Manuel Diaz Ortuño
Anales de documentación núm. 6, 2003
URL: http://www.um.es/fccd/anales/ad06/ad0603.pdf

"La Disciplina de los sistemas de Bases de Datos. Historia, Situación actual y perspectivas"
José Hernandez Orallo
Departament de Sistemas Informàtics i Computació. Universitat Politécnica de València
URL: http://www.dsic.upv.es/~jorallo/docent/BDA/DisciplinaBD.pdf

Bibliografía y recursos sobre los XQuery Languages y XQuery 1.0

"XML Query (XQuery) Requirements"
W3C Working Draft, Noviembre 2003
URL: http://www.w3.org/TR/xquery-requirements/

"XML Query Uses Cases"
W3C Working Draft, Noviembre 2003
URL: http://www.w3.org/TR/xquery-use-cases/ .

"XML Path Language (Xpath) 2.0
W3C Working Draft, Noviembre 2003
URL: http://www.w3.org/TR/xpath20/

"Xquery 1.0 and Xpath 2.0 Formal Semantics"
W3C Working Draft Noviembre 2003
URL: http://www.w3c.org/TR/xquery-semantics/

"Xquery 1.0: An XML Query Language"
W3C Working Draft Noviembre 2003
URL: http://www.w3.org/TR/xquery/

Xquery 1.0 and Xpath 2.0 Functions and Operators
W3C Working Draft Noviembre 2003
URL: http://www.w3.org/TR/xpath-functions/

"An Early Look at XQuery", SIGMOD record, vol.31, n.4, 2002,
Andrew Eisenberg, Jim Melton
URL: http://www.acm.org/sigmod/record/issues/0212/AndrewEJimM.pdf

"Un análisis de lenguajes de consulta para XML"
Adelaida Delgado Domínguez, Ricardo Baeza Yates
Novatica núm 157, Mayo – Junio 2002

"Introducción a X-Query. Introducción al estándar propuesto por W3C para un lenguaje de consultas XML"
Howard Katz, Enero 2002
URL: http://www-106.ibm.com/developerworks/es_es/xml/library/x-xquery.html

Bibliografia y Recursos sobre Oracle 9i

"Oracle9i: Desarrollo Web"
Bradley D. Brown
Oracle Press, Anaya Multimedia, 2002

"Manual de Oracle XML"
Ben Chang
Oracle Press, Osborne-Mc Graw Hill, 2001

"Desarrollo de Aplicaciones e PL/SQL"
Oracle Education
Projecte Universitat Empresa PUE – Oracle Corporation, 1998

"What are the Oracle9i Native XML Database Features? Database Support for XML."
Oracle Corporation, 2002
URL: http://otn.oracle.com/tech/xml/htdocs/about_oracle_xml_products.htm

Información sobre Oracle 9i XDK
URL: http://otn.oracle.com/tech/xml/htdocs/about_oracle_xml_products.htm

"Oracle XML DB. Technical White Paper"
Oracle Corporation, 2002
URL: http://otn.oracle.com/tech/xml/xmldb/pdf/xmldb_92twp.pdf

"Fast Track to Oracle 9i. Paper Number 131. XML in the Database. Storing XML with all your other critical data"
Mark Drake
Oracle Open World 2001

"Form Meets Function in SQL/XML"
Joe McKee
Oracle Magazine, November/December 2002

"Make XML Native and Relative"
Jonathan Gennick
OracleMagazine, January/February 2003

Bibliografia y recursos sobre Tamino

"Tamino. The information server for bussiness"
Software AG, 2003
URL: http://www.softwareag.com.mx/mexico/index.htm

"Administrando el Almacenamiento de Datos XML"
Jerry Emerick
URL: http://www.acm.org/crossroads/espanol/xrds8-4/XML_RDBMS.html

"XML: Transporte de contenido"
William Diaz
URL: http://jungla.dit.upm.es/~santiago/externos/docencia/doctorado/drci/trabajos00-01/wdiaz/sld021.htm

"Tamino. The next generation XML Server. Concept, benefits and features"
AG Software

"General Migration Strategies"
URL: http://www.xmlone.co.kr/ttn/binaries/manual/Documentation/pdf/migrate/migstratgen.pdf

"Servicios profesionales de Software AG"
Software AG
URL: http://www.softwareag.com/spain/servicios/default.htm