Capítulo 2. Introducción a XML

1. Introducción

EL Lenguaje Extensible de Etiquetado (XML, "*Extensible Markup Language*") [4] describe un tipo de objetos de datos llamados *Documentos XML* y también describe parcialmente el comportamiento de los programas de ordenador que los procesan. XML es un perfil de aplicación o subconjunto de SGML ("*Standard Generalized Markup Language*", ISO 8879) [10]. Por construcción, los documentos XML son documentos SGML conformes. XML está descrito completamente en un documento de recomendación del W3C. Actualmente está en su versión 1.0 (Fourth Edition), aunque la versión que utiliza la especificación QTI del IMS [43] es la 1.0 (Second Edition). Su objetivo es permitir servir, recibir y procesar SGML genérico en la Web del mismo modo que se posible ahora hacerlo con HTML [49]. XML se ha diseñado buscando facilidad de implementación e interoperativilidad con SGML y HTML.

1.1 Historia

La versión 1.0 del lenguaje XML es una recomendación del W3C creada en febrero de 1998, aunque se trabajó en ella desde un par de años antes. Está basado en el anterior estándar SGML, que data de 1986, pero que empezó a gestarse a principios de los años 70, y a su vez basado en el GML (*Generalized Markup Language*) creado por IBM en 1969. Esto significa que aunque XML pueda parecer moderno, sus conceptos están más que asentados y aceptados de forma amplia. Está además asociado a la recomendación del W3C DOM (Document Object Model), aprobado también en 1998. Éste no es más que un modelo de objetos (en forma de API) que permite acceder a las diferentes partes que pueden componer un documento XML o HTML.

XML, a pesar de mantener la misma filosofía que SGML, al ser más exigente en la sintaxis, hace que sea más fácil la construcción de bibliotecas para procesarlo. Así derivó XML como subconjunto simplificado, eliminando las partes más engorrosas y menos útiles de SGML. Al igual que su antecesor, XML es un metalenguaje: es un lenguaje para definir lenguajes. Los elementos que lo componen pueden dar información sobre lo que contienen, no necesariamente sobre su estructura física o su presentación, como ocurre en HTML.

Los objetivos de diseño para XML son:

1. XML será utilizable directamente en Internet.
2. XML soportará gran variedad de aplicaciones.
3. XML será compatible con SGML.
4. Será fácil escribir programas que procesen documentos XML.
5. El número de características opcionales en XML se intentarán mantener al mínimo, idealmente cero.
6. Los documentos XML deberían ser razonablemente claros y legibles por una persona.
7. El diseño XML debería ser preparado rápidamente.
8. El diseño de XML será formal y conciso.
9. Los documentos XML serán fáciles de crear.

10. Es de mínima importancia que las marcas XML sean concisas.

XML no ha nacido sólo para su aplicación en Internet, sino que se propone como lenguaje de bajo nivel (a nivel de aplicación, no de programación) para intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo, y casi cualquier cosa que se pueda pensar.

Se puede suponer de este modo que XML constituye la capa más baja dentro del nivel de aplicación, sobre el que se puede montar cualquier estructura de tratamiento de documentos, hasta llegar a la presentación, como se ve en el ejemplo de la Figura 2.1

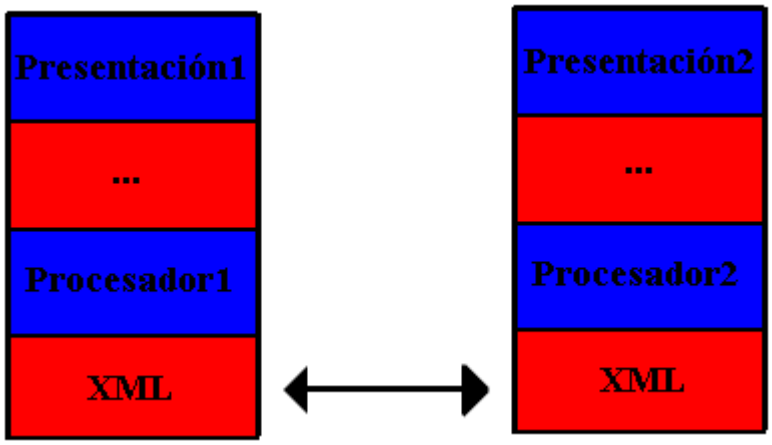

Figura 2.1. Intercambio de datos entre dos plataformas mediante XML

2. Características principales

XML no es más que un conjunto de reglas para definir etiquetas semánticas que organizan un documento en diferentes partes. XML es un metalenguaje que define la sintaxis utilizada para definir otros lenguajes de etiquetas estructurados.

Los ficheros XML son ficheros de texto, que en principio está en código Unicode, pero se pueden usar otros juegos de caracteres como el latin-1. Existen cinco caracteres especiales en XML: los símbolos menor que, `<`, mayor que, `>`, las comillas dobles, `&quot;`, la comilla simple, `&apos;`, y el carácter `&`. Los símbolos mayor que y menor se usan para delimitar las marcas que dan la estructura al documento. Cada marca tiene un nombre; como por ejemplo: la marca `<figura>`, que puede tener uno o más *atributos*: `<figura fichero="foto1.jpg" tipo="jpeg">` tiene dos atributos, "`fichero`" y "`tipo`". Los atributos toman valores que tienen que estar entre comillas dobles o simples.

Cuando sea necesario usar uno de los 5 caracteres especiales en el texto, para evitar que sean interpretados de forma especial se usan las siguientes *entidades*: `&lt;`, `&gt;`, `&quot;`, `&apos;`, `&amp;`, para `<`, `>`, `&quot;`, `&apos;` y `&`, respectivamente. Esto explica también porqué `&` es un carácter especial: se usa para representar entidades; una entidad es un carácter adicional que no forma parte del alfabeto usado por defecto en el texto (los

caracteres especiales obviamente quedan excluidos del alfabeto usado para el texto) comienza por `&`, seguido del nombre de la entidad e inmediatamente un punto y coma (aunque realmente, como se verá más adelante, existen más tipos de entidades).

Una diferencia importante con SGML, y en particular HTML, es que los nombres de las marcas y de sus atributos distinguen entre mayúsculas y minúsculas: `<a>` y `<A>` serían dos marcas diferentes. Otra diferencia sobresaliente con SGML es que en XML ninguna marca se puede dejar abierta: o sea, por cada marca, por ejemplo `<p>` deberá existir una marca correspondiente `</p>` que indica donde termina el contenido de la marca. Y si una marca cualquiera no contiene ningún texto, por ejemplo `<hr></hr>`, se puede abreviar de la siguiente forma: `<hr/>`, pero nótese que la primera forma también es válida; en cambio, escribir únicamente `<hr>` daría un error.

2.1 Estructura de un Documento XML

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

Uno de los conceptos más relevantes de **XML** es la **distinción entre documentos XML validados y bien formados**:

- **Bien formados**: son todos los que cumplen las especificaciones del lenguaje respecto a sus reglas sintácticas, aunque sin estar sujeto a los elementos fijados en un **DTD** (definición de los elementos que puede haber en el documento **XML**). De hecho, los documentos **XML** deben tener una estructura jerárquica muy estricta, la cual deben cumplir los documentos bien formados.

- **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 dicte.

Como ya se ha mencionado, un **DTD** es una definición de los elementos que puede haber en el documento **XML**, así como su relación entre ellos, sus atributos, posibles valores, etc. De hecho **DTD** significa *Document Type Definition*, o *Definición de Tipo de Documento.* Es, en definitiva, una definición de la gramática del documento.

La primera tarea a llevar a cabo cuando se está procesando cualquier información formateada mediante **XML** es comprobar si está bien formada. Posteriormente, se habrá de verificar si sigue las reglas gramaticales de un **DTD** en caso de estar vinculado a alguno. Existen, pues, dos tipos de herramientas para procesar documentos **XML**: los **parsers no validadores**, que únicamente comprueban si están bien formados, y los **parsers validadores**, que verifican que además de bien formado se atiene a su **DTD** y es válido. En general, se llama "parser" a los programas que analizan sintácticamente o *leen* los documentos XML.

A continuación se presenta un documento XML simple:

```
<?xml version="1.0" encoding="UTF-8" standalone= "yes"?>
<ficha>
      <nombre>Ángel</nombre>
      <apellido>Barbero</apellido>
      <direccion>c/Ulises, 36</direccion>
</ficha>
```

La primera línea indica que lo que la sigue es XML. Aunque es opcional, es recomendable incluirla. Puede tener varios atributos, algunos obligatorios y otros no:

- **version:** Indica la versión de XML usada en el documento
- **encoding:** La forma en que se ha codificado el documento. Se puede poner cualquiera, y depende del *parser* el entender o no la codificación.
- **standalone:** Indica si el documento va acompañado de un DTD ("`no`"), o no lo necesita ("`yes`").

La "Declaración de Tipo de Documento" define qué tipo de documento se está creando para ser procesado correctamente. Es decir, define qué declaración de tipo de documento (DTD) valida y define los datos que contiene nuestro documento XML.

En ella se define el tipo de documento, y dónde encontrar la información sobre su Definición de Tipo de Documento, mediante:

- **Un identificador público (PUBLIC):** que hace referencia a dicha DTD.
- **Identificador universal de recursos (URI)** [9]**:** precedido de la palabra SYSTEM.

Ejemplos:

```
<!DOCTYPE MESAJE SYSTEM "mesaje.dtd">

<!DOCTYPE HTML PUBLIC "-/ /W3C/ /DTD HTML 3.2 Final/ /EN">

<!DOCTYPE LABEL SYSTEM "http://azuaje.ulpgc.es/dtds/label.dtd">
```

2.2 Documentos XML bien formados

Un documento XML se dice que está bien formado si encaja con las especificaciones XML, lo que implica:

Estructura jerárquica de elementos

Los documentos XML deben seguir una estructura estrictamente jerárquica con lo que respecta a las etiquetas que delimitan sus elementos. Una etiqueta debe estar correctamente *incluida* en otra. Así mismo, los elementos con contenido, deben estar correctamente *cerrados*. A continuación se muestra un ejemplo incorrecto y posteriormente otro ejemplo escrito correctamente.

```
<li>HTML <b>permite<i> esto</b></i>

<li>En XML la <b>estructura<i> es </i>jerárquica</b>.</li>
```

Etiquetas vacías

HTML permite elementos sin contenido. XML también, pero la etiqueta debe ser

de la siguiente forma <elemento sin contenido/>. A continuación se muestra un ejemplo incorrecto y posteriormente otro correcto.

```
<li>Esto es HTML <br> en el que casi todo está permitido </li>

<li>En XML, es <br/> más restrictivo.</li>
```

Un solo elemento raíz

Los documentos XML sólo permiten un elemento raíz, del que todos los demás sean parte. Es decir, la jerarquía de elemento de un documento XML bien formado sólo puede tener un elemento inicial.

Valores de atributos

Los valores de atributos en XML siempre deben estar encerrados entre comillas simples, `'` o dobles, `"`. En el siguiente ejemplo, la primera línea sería incorrecta en XML, no así la segunda:

```
<a HREF=http://www.dis.ulpgc.es/>

<a HREF="http://www.dis.ulpgc.es/">
```

Tipos de letras, espacios en blanco

XML es sensible al tipo de letra que se utiliza, es decir, trata las mayúsculas y minúsculas como caracteres diferentes. Por lo tanto, los elementos definidos como "`FICHA`", "`Ficha`", "`ficha`" y "`fiCha`" serían elementos diferentes.

Existe un conjunto de caracteres denominados "espacios en blanco" que los procesadores XML tratan de forma diferente en el marcado XML. Estos caracteres son: los "espacios", tabuladores, retornos de carro y los saltos de línea.

La especificación XML 1.0 permite el uso de esos "espacios en blanco" para hacer más legible el código, y en general son ignorados por los procesadores XML.

Nombrando cosas

Al utilizar XML, es necesario asignar nombres a las estructuras, tipos de elementos, entidades, elementos particulares, etc.

No se pueden crear nombres que empiecen con la cadena "`xml`", "`xML`", "`XML`" o cualquier otra variante. Las letras y guiones se pueden usar en cualquier parte del nombre. También se pueden incluir dígitos, guiones y caracteres punto, pero no se puede empezar por ninguno de ellos. El resto de caracteres, como algunos símbolos, y espacios en blanco, no se pueden usar.

Marcado y datos

Las construcciones con etiquetas, referencias de entidad y declaraciones se denominan "marcas".Éstas son las partes del documento que el procesador XML espera entender. El resto del documento que se encuentra entre las marcas son los datos que resultan entendibles por las personas.

Es sencillo reconocer las marcas en un documento XML. Son aquellas porciones que empiezan con "<" y acaban con ">", o bien, en el caso de las referencias de entidad, empiezan por "&" y acaban con ";".

2.3 Elementos

Los elementos XML pueden tener contenido (más elementos, caracteres, o ambos a la vez), o bien ser elementos vacíos.

Un elemento con contenido es, por ejemplo:

```
<aviso tipo="emergencia" gravedad="mortal">Que no cunda el
pánico</aviso>
```

Siempre empieza con una `<etiqueta>` que puede contener atributos o no, y termina con una `</etiqueta>` que debe tener el mismo nombre. Al contrario que HTML, en XML siempre se debe *cerrar* un elemento.

Hay que tener en cuenta que el símbolo "<" siempre se interpreta como inicio de una etiqueta XML. Si no es el caso, el documento no estará bien formado. Para usar ciertos símbolos se usan las entidades predefinidas, explicadas más arribas.

Un elemento vacío, es el que no tiene contenido. Por ejemplo:

```
<identificador DNI="23123244"/>

<linea-horizontal/>
```

Al no tener una etiqueta de cierre que delimite un contenido, se utiliza la forma `<etiqueta/>`, que puede contener atributos o no. La sintaxis HTML permite etiquetas vacías tipo `<hr>` o `<img src="...">`. En HTML reformulado para que sea un documento XML bien formado (XHTML [1]), se debería usar `<hr/>` o `<img src="..."/>`.

2.4 Atributos

Como se ha mencionado antes, los elementos pueden tener atributos, que es una forma de incorporar características o propiedades a los elementos de un documento. Por ejemplo, un elemento "`chiste`" puede tener un atributo "`tipo`" y un atributo "`calidad`", con valores "`lepe`" y "`bueno`" respectivamente.

```
<chiste tipo="lepe" calidad="bueno"> Esto es uno de Lepe que va
paseando... </chiste>
```

En una Definición de Tipo de Documento, se especifican los atributos que pueden tener cada elemento, así como sus valores y tipos de valor posible.

Al igual que en otras cadenas literales de XML, los atributos pueden estar marcados entre comillas simples (') o dobles ("). Cuando se usa uno para delimitar el valor del atributo, el otro tipo se puede usar dentro:

```
<verdura clase="zanahoria" longitud='15" y media'>

<cita texto="'Hola buenos días', dijo él">
```

2.5 Entidades Predefinidas

En XML 1.0, se definen cinco entidades para representar caracteres especiales y que no se interpreten como marcado por el procesador XML. Es decir, que así se puede usar el carácter "<" sin que se interprete como el comienzo de una etiqueta XML, como ya se explicó más arriba. Las cinco entidades son para los cinco caracteres reservados: el carácter "`&`", el menor que, "`<`", el mayor que, "`>`", la comilla simple, "`'`", y las comillas dobles, "`"`".

2.6 Secciones CDATA

Existe otra construcción en XML que permite especificar datos, utilizando cualquier carácter, especial o no, sin que se interprete como marcado XML. La razón de esta construcción llamada "`CDATA`" (Character DATA) es que a veces es necesario para los autores de documentos XML, poder leerlo fácilmente sin tener que descifrar los códigos de entidades, especialmente cuando son muchas.

Como ejemplo, el siguiente (primero usando entidades predefinidas y luego con un bloque CDATA):

```
<parrafo>Lo siguiente es un ejemplo de HTML.</html>
      <ejemplo>
            &lt;html>
            &lt;head>&lt;title>Rock &amp; Roll&lt;/title>&lt;/head>
      </ejemplo>

      <ejemplo>
            <![CDATA[
                  <html>
                  <head><title>Rock & Roll</title></head>
            ]]>
      </ejemplo>
</parrafo>
```

Como se ha visto, dentro de una sección CDATA se puede poner cualquier cosa, que no será interpretada como algo que no es así. Existe una excepción y es la cadena "`]]>`" con el que termina el bloque CDATA. Esta cadena no puede utilizarse dentro de una sección CDATA.

2.7 Comentarios

A veces es conveniente insertar comentarios en documentos XML, que son ignorados por el procesado de la información y las reproducciones del documento. Los comentarios tienen el mismo formato que los comentarios de HTML. Es decir, comienza por la cadena "`<!--`" y termina con "`-->`".

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

Se pueden introducir comentarios en cualquier parte del documento salvo dentro de las declaraciones, etiquetas, u otros comentarios.

3. Definición de Tipo de Documento (Document Type Definition, DTD)

Crear una definición del tipo de documento (DTD) es como crear nuestro propio lenguaje de marcado, para una aplicación específica. Por ejemplo, se podría crear una DTD que defina una tarjeta de visitas. A partir de esa DTD, se tendría una serie de elementos XML que permitirían definir tarjetas de visita.

La DTD define los tipos de elementos, atributos y entidades permitidas, y puede expresar algunas limitaciones para combinarlos. Los documentos que se ajusten a su DTD, se denominan "válidos". También existen documentos XML sin una DTD asociada, en ese caso no son "válidos", pero tampoco "inválidos".

Una DTD puede residir en un fichero externo, y quizás compartido por varios (puede que miles de) documentos XML. O bien, puede estar contenido en el propio documento XML, como parte de su declaración de tipo de documento.

A continuación se muestra un ejemplo:

```
<! DOCTYPE etiqueta[
<!ELEMENT etiqueta (nombre, calle, ciudad, pais, codigo)>
<!ELEMENT nombre (#PCDATA)>
<!ELEMENT calle (#PCDATA)>
<!ELEMENT ciudad (#PCDATA)>
<!ELEMENT pais (#PCDATA)>
<!ELEMENT codigo (#PCDATA)>
]>

<etiqueta>
      <nombre>Fulano Mengánez</nombre>
      <calle>c/ Mayor, 27</calle>
      <ciudad>Valderredible</ciudad>
      <pais>España</pais>
      <codigo>39343</codigo>
</etiqueta>
```

La declaración del tipo de documento empieza en la primera línea y termina con "`]>`". Las declaraciones DTD son las líneas que empiezan con "`<!ELEMENT`" y se denominan declaraciones de tipo elemento. También se pueden declarar atributos, entidades y anotaciones para una DTD.

En el ejemplo anterior, todas las declaraciones DTD que definen "etiquetas" residen dentro del documento. Sin embargo, la DTD se puede definir parcial o completamente en otro documento. Por ejemplo:

```
<?xml version="1.0"?>
<!DOCTYPE coche SYSTEM "http://www.sitio.com/dtd/coche.dtd">
<coche>
      <modelo>...</modelo>
      ...
```

```
</coche>
```

Así, una DTD define qué tipos de elementos contiene el documento XML, qué subelementos y en qué orden puede contener cada uno de ellos, qué atributos tienen los elementos y de qué tipos son, mediante un lenguaje definido para ello.

3.1 Declaración de Entidades

XML hace referencia a objetos (ficheros, páginas Web, imágenes, cualquier cosa) que no deben ser analizados sintácticamente según las reglas de XML, mediante el uso de entidades. Se declaran en la DTD mediante el uso de "<!ENTITY".

Una entidad puede no ser más que una abreviatura que se utiliza como una forma corta de algunos textos. Al usar una referencia a esta entidad, el analizador sintáctico reemplaza la referencia con su contenido. En otras ocasiones es una referencia a un objeto externo o local.

4. Esquemas XML (XML Schemas)

Un "esquema XML" es algo similar a un DTD, es decir, define qué elementos puede contener un documento XML, cómo están organizados, y qué atributos y de qué tipo pueden tener sus elementos.

Las ventajas de los esquemas con respecto a las DTDs son:

- **Usan sintaxis de XML**, al contrario que las DTDs.
- **Permiten especificar los tipos de datos.**
- **Son extensibles.**

Por ejemplo, un esquema permite definir el tipo del contenido de un elemento o de un atributo, y especificar si debe ser un número entero, una cadena de texto, una fecha, etc. Las DTDs no permiten hacer estas cosas.

A continuación se ve un ejemplo de un documento XML, y su esquema correspondiente:

```
<documento xmlns="x-schema:personalSchema.xml">
      <persona id="fulano">
            <nombre>Fulano Mengánez</nombre>
      </persona>
</documento>
```

Como se observa en el documento XML anterior, se hace referencia a un espacio de nombres (namespace) llamado "`x-schema:personalSchema.xml`". Es decir, se le está diciendo al analizador sintáctico XML (parser) que valide el documento contra el esquema "`personalSchema.xml`".

El esquema sería algo parecido a esto:

```
<Schema xmlns="urn:schemas-microsoft-com:xml-data"
            xmlns:dt="urn:schemas-microsoft-com:datatypes">
      <AttributeType name='id' dt:type='string' required='yes'/>
      <ElementType name='nombre' content='textOnly'/>
```

```
      <ElementType name='persona' content='mixed'>
            <attrubyte type='id'/>
            <element type='nombre'/>
      </ElementType>
      <ElementType name='documento' content='eltOnly'>
            <element type='persona'/>
      </ElementType>
</Schema>
```

El primer elemento del esquema define dos espacios de nombres. El primero, "`xml-data`", le dice al analizador que esto es un esquema y no otro documento XML cualquiera. El segundo "`datatypes`" permite definir el tipo de elementos y atributos utilizando el prefijo "`dt`".

ElementType
Define el tipo y contenido de un elemento, incluyendo los subelementos que pueda contener.

AttributeType
Asigna un tipo y condiciones a un atributo.

attribute
Declara que un atributo previamente definido por **AttributeType** puede aparecer como atributo de un elemento determinado.

element
Declara que un elemento previamente definido por **ElementType** puede aparecer como contenido de otro elemento.

Tal como se ha visto, es necesario empezar el esquema definiendo los elementos más profundamente anidados dentro de la estructura jerárquica de elementos del documento XML. Es decir, hay que trabajar "de dentro hacia fuera". Visto de otra manera, las declaraciones de tipo **ElementType** y **AttributeType** deben preceder a las declaraciones de contenido **element** y **attribute** correspondientes.

5. Modelo de Objetos de Documento (Document Object Model, DOM)

El Modelo de Objetos del Documento (DOM) [46] es una interfaz de programación de aplicaciones (API) para analizar sintácticamente documentos HTML y XML. Es una interfaz estándar que posibilita a scripts y programas, y de forma dinámica, el acceso y manipulación de la estructura, estilo y contenido de los documentos. XML se utiliza cada vez más como un medio para representar muchas clases diferentes de información que puede ser almacenada en diversos sistemas, y mucha de esta información se vería, en términos tradicionales, más como datos que como documentos. Sin embargo, XML presenta estos datos como documentos, y se puede usar DOM para manipular estos datos.

Siendo una especificación del W3C, uno de los objetivos importantes del Modelo de Objetos del Documento es proporcionar una interfaz estándar de

programación que pueda utilizarse en una amplia variedad de entornos y aplicaciones. DOM se ha diseñado para ser utilizado en cualquier lenguaje de programación.

5.1 Características Principales

DOM es una API de programación para documentos. Guarda una gran similitud con la estructura del documento al que modela. Una implementación DOM que procese un Documento estructurado (por ejemplo, un documento XML bien formado) lee el documento completo y construye un árbol de objetos *nodo* en memoria. Una aplicación puede entonces acceder y procesar este árbol, el cual reside en memoria, como modelo del documento XML. El *parser* realiza la transformación de la Figura 2.2:

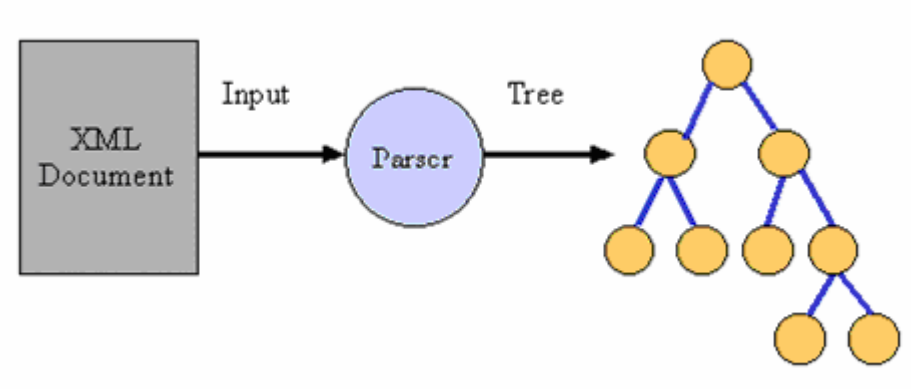

Figura 2.2. Transformación realizada por el parser DOM

Por ejemplo, considérese este documento HTML:

```
<HTML>
      <HEAD>
            <TITLE>Mi primera página para DOM</TITLE>
      </HEAD>
      <BODY>
            <P id="primera" align="center">¡Hola Mundo!</P>
            <P id="segunda">
                  <A href="index.htm">Inicio</A>
            </P>
      </BODY>
</HTML>
```

El DOM lo representa según el modo indicado en la Figura 2.3:

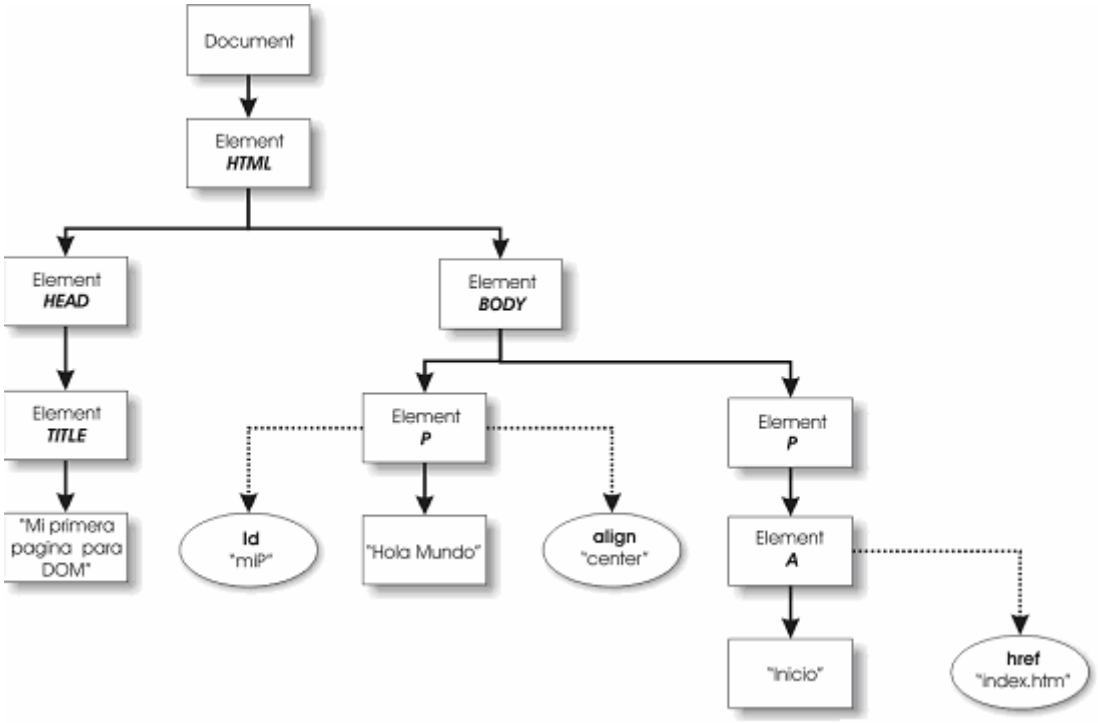

Figura 2.3. Representación DOM de un documento XML

El W3C establece varios niveles de actuación:

- **Nivel 1:** se refiere a la parte interna, y modelos para HTML y XML. Contiene funcionalidades para la navegación y manipulación de documentos.
- **Nivel 2:** incluye un modelo de objetos e interfaz de acceso a las características de estilo del documento, definiendo funcionalidades para manipular la información sobre el estilo del documento.
- Posteriores niveles especificarán interfaces a posibles sistemas de ventanas, manipulación de DTD y modelos de seguridad.

En DOM, los documentos tienen una estructura lógica que es muy parecida a un árbol. Para ser más precisos, es más bien como un *bosque* o una *arboleda*, que puede contener más de un árbol. Sin embargo, DOM no especifica que los documentos deban ser *implementados* como un árbol o un bosque, ni tampoco especifica cómo deben implementarse las relaciones entre objetos. DOM es un modelo lógico que puede implementarse de cualquier manera que sea conveniente. Una propiedad importante de los modelos de estructura del DOM es su *isomorfismo estructural*: si dos implementaciones cualesquiera del Modelo de Objetos del Documento se usan para crear una representación del mismo documento, ambas crearán el mismo modelo de estructura, con exactamente los mismos objetos y relaciones.

6. API Simple para XML (Simple API for XML, SAX)

SAX [47] es una sencilla API estándar para analizar XML, implementada por muchos y diferentes analizadores XML. La ventaja principal de SAX es que es ligera y rápida. Esto es principalmente porque es una API basada en eventos, lo que significa que informa de eventos de análisis (como el comienzo y el final de los elementos) directamente a la aplicación usando servicios repetidos.

6.1 Características Principales

SAX procesa (*parse*) el documento o información XML de una manera muy diferente a DOM, SAX procesa la información *por eventos*. A diferencia de DOM que genera un *árbol jerárquico en memoria*, SAX procesa la información en XML conforme esta es presentada (*evento por evento*) de forma *secuencial*, efectivamente manipulando cada elemento uno detrás de otro, sin incurrir en un uso excesivo de memoria. Así, se pueden destacar las siguientes características:

- SAX es un *parser* ideal para manipular archivos de gran tamaño, ya que no genera un *árbol en memoria* como DOM.
- Es más rápido y sencillo que utilizar DOM
- La sencillez antes mencionada tiene su precio, debido a que SAX funciona *por eventos* no es posible manipular información una vez procesada, en DOM no existe esta limitación ya que se genera el *árbol jerárquico en memoria* y es posible regresar a modificar nodos.

Por lo tanto, la aplicación del usuario debe implementar manejadores para ocuparse de los diversos eventos generados por SAX, como muestra la Figura 2.4:

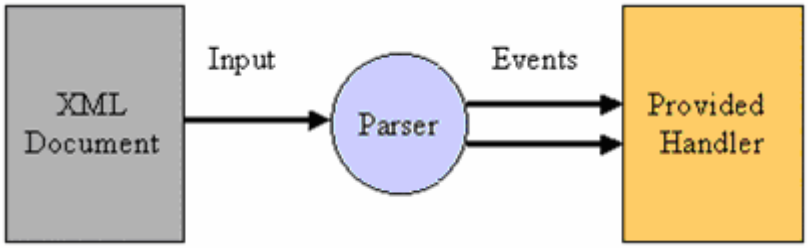

Figura 2.4. SAX usa retrollamadas para notificar a los manejadores los eventos

Es labor del programador preparar los manejadores para todos los tipos de eventos que el *parser* pueda encontrar en el documento XML. Así, la sencillez de la API da como lugar que la programación pueda ser más compleja.

SAX se ha desarrollado con vistas a aplicaciones de servidor ya que el servidor debe suministrar rápidamente el resultado de transformar un documento XML, o para sistemas con limitaciones importantes de memoria, en los que el uso de DOM y almacenar en memoria todo el árbol sería prohibitivo.

7. Streaming API for XML (StAX)

StAX (Streaming API for XML) [11] es una API de analizado XML tipo *pull-parsing*, conducida por eventos, basada en Java, de flujo, para leer y escribir documentos XML. StAX permite crear procesadores de XML bidireccionales que son rápidos, relativamente fáciles de programar y con poco uso de memoria.

Es una alternativa a SAX y DOM para los desarrolladores que buscan filtrado de flujo, procesado y modificación de alto rendimiento, en particular con poca memoria y pocos requisitos.

7.1 ¿Por qué StAX?

El proyecto StAX lo dirigió BEA [38] con el soporte de Sun Microsystems [39], y la especificación JSR 173 pasó la votación de aprobación final del *Java Community Process* [14] en Marzo de 2004 (http://jcp.org/en/jsr/detail?id=173). El primer objetivo de la API StAX es proporcionar "control al programador sobre el analizador mediante una API basada en un iterador simple. Esto permite al programador preguntar por el siguiente evento (*pull*, o sacar el evento) y permite que se almacene el estado en forma de proceso". StAX se creó para tratar de superar las limitaciones de las Apis de *parsers* (o analizadores sintácticos) de XML más extendidas, SAX y DOM.

7.2 Analizador basado en Flujo frente a DOM

Hay dos formas de trabajar con documentos XML: flujo (*streaming*) de documento y el modelo de objetos de documento (DOM).

El modelo DOM implica crear objetos en memoria representando el documento, y toda la información de estado para todo el documento XML. Una vez en memoria, se puede navegar por los árboles DOM libremente y procesarlos arbitrariamente, proporcionando así a los desarrolladores máxima flexibilidad. No obstante, el coste de esta flexibilidad es un gran uso de memoria potencialmente y requerimientos significativos de procesador, ya que toda la representación del documento se debe mantener en memoria como objetos durante toda la duración del procesado del documento. Esto puede que no sea un problema cuando se trabaja con documentos pequeños, pero los requisitos de memoria y procesador pueden aumentar rápidamente a medida que lo hace el tamaño del documento.

El flujo, o *streaming*, se refiere al modelo de programación en el que la información XML se transmite y se procesa en serie en tiempo de ejecución de la aplicación, a menudo en tiempo real, y de fuentes dinámicas cuyo contenido no se conoce de antemano. Además, los *parsers* basados en flujo pueden generar su salida inmediatamente, y los elementos de información se pueden descartar y desechar inmediatamente después de haber sido usados. Aunque usan menos memoria, tienen menos requisitos de procesador, y mayor rendimiento en ciertas situaciones, el principal problema de los *parsers* basados en flujo es que sólo se ve el evento correspondiente al estado en ese momento. Esto implica que hay que saber qué procesado se quiere hacer antes de leer el documento XML. Los modelos de analizadores de XML basados en *streaming* son particularmente útiles cuando la aplicación tiene estrictas limitaciones de memoria, o cuando la aplicación procesa de forma simultánea varias peticiones, como en un servidor de aplicaciones.

7.3 *Pull Parsing* frente a *Push Parsing*

El término *streaming pull parsing* se refiere al modelo de programación en el que una aplicación cliente llama a los métodos de una biblioteca de proceso XML cuando necesita interactuar con la información XML – es decir, el cliente sólo adquiere

(*pull*, saca) datos XML cuando los reclama explícitamente(es el modelo usado por el analizador StAX).

Por el contrario, el *streaming push parsing* hace referencia al modelo en el que el procesador XML envía (*push*, mete) datos XML al cliente a medida que el procesador encuentra elementos en el documento XML – es decir, el procesador envía los datos independientemente de si el cliente está listo o no para usarlos en ese momento (es el modelo usado por el analizador SAX).

El modelo de *pull parsing* proporciona algunas ventajas sobre *push parsing* cuando se trabaja con flujos XML:

- Con *pull parsing* el cliente controla el hilo de la aplicación, y puede llamar a los métodos del procesador cuando lo necesita, al contrario que con *push parsing*, donde el hilo de la aplicación lo controla el procesador XML, y el cliente sólo acepta invocaciones del analizador.
- Las bibliotecas *pull parsing* pueden ser mucho más pequeñas y el código del cliente para interactuar con estas bibliotecas son mucho más simples que con las bibliotecas *push*, incluso para documentos más complejos.
- Los clientes *pull* pueden leer múltiples documentos a la vez en un mismo hilo.
- Un *pull parser* StAX puede filtrar documentos XML de tal forma que se pueden ignorar los elementos innecesarios para el cliente.

7.4 *Comparación de StAX con otras APIs XML*

StAX no es tan poderoso o flexible como JDOM (API para trabajar con DOM en Java), pero no requiere tanta carga de memoria ni de procesador para usarse, y StAX puede tener más rendimiento en muchos casos que las APIs basadas en DOM. En general tiene las ventajas de una API basada en un modelo de flujo frente a una basada en DOM.

Así, la comparación lógica que se puede hacer es entre StAX y SAX, que es donde StAX ofrece características beneficiosas en muchos casos, como los siguientes:

- Los clientes que usan StAX generalmente son más fáciles de codificar que los clientes SAX. Mientras se puede decir que los analizadores SAX son más fáciles de escribir, el código de los analizadores StAX puede ser más corto y el código para interactuar con él más simple.
- StAX es una API bidireccional, es decir, que puede leer y escribir documentos XML. SAX es de solo lectura, así que hace falta otra API para escribir documentos XML.
- SAX es una API de tipo *push*, mientras que StAX lo es de tipo *pull*. Más arriba se indicaron las diferencias entre ambas.

La Tabla 2.1 muestra las diferencias entre las distintas APIs:

Característica	StAX	SAX	DOM
Tipo de API	*Pull*, de flujo	*Push*, de flujo	Árbol en memoria
Facilidad de uso	Alta	Media	Alta
Capacidad XPath [15]	No	No	Sí
Eficiencia de CPU y Memoria	Buena	Buena	Varía
Sólo hacia delante	Sí	Sí	No
Lee XML	Sí	Sí	Sí
Escribe XML	Sí	No	Sí
Crear, Leer, Actualizar, Borrar	No	No	Sí

Tabla 2.1 Comparación entre distintas APIs de analizadores XML para Java

7.5 *API StAX*

La API StAX proporciona métodos para procesado de documentos XML basados en eventos e iteradores. Los documentos XML se tratan como una serie filtrada de eventos, y los estados se pueden almacenar. Y además, a diferencia de SAX, la API StAX es bidireccional, permitiendo lectura y escritura de documentos XML.

La API StAX en realidad consta de dos juegos distintos de APIs: una API *cursor* y una API *iterator*.

7.5.1 API *Cursor*

Como indica su nombre, la API *cursor* representa un cursor con el que se puede recorrer un documento XML desde el principio hasta el final. Este cursor apunta a una cosa en cada momento, y siempre se mueve hacia delante, nunca hacia atrás.

Las dos interfaces principales son `XMLStreamReader` y `XMLStreamWriter`. El primero incluye métodos para extraer toda la información obtenible del modelo de información XML, incluyendo codificación del documento, nombres de elementos, atributos, espacios de nombres, nodos de texto, etiquetas de apertura, comentarios, instrucciones de procesado, etc.; por ejemplo:

```
public interface XMLStreamReader {
  public int next() throws XMLStreamException;
  public boolean hasNext() throws XMLStreamException;
  public String getText();
  public String getLocalName();
  public String getNamespaceURI();
  // ... resto de los métodos
}
```

Se pueden llamar a los métodos de `XMLStreamReader`, como `getText` o `getName` para extraer los datos de la posición actual del cursor. `XMLStreamWriter` proporciona métodos que corresponden a los tipos de evento `StartElement` (inicio de elemento) y `EndElement (fin de elemento)`; por ejemplo:

```
public interface XMLStreamWriter {
        public void writeStartElement(String localName) throws
                XMLStreamException;
        public void writeEndElement() throws XMLStreamException;
        public void writeCharacters(String text)
                throws   XMLStreamException;
        // ... resto de los métodos
}
```

La API *cursor* es similar a SAX en muchos aspectos. Por ejemplo, los métodos están disponibles para acceder directamente a la información de caracteres y cadenas, y los índices de enteros se pueden usar para acceder a información de espacios de nombres y atributos. Como con SAX, los métodos de la API *cursor* devuelven la información XML en forma de cadenas, que minimizan los requisitos de asignación de objetos.

7.5.2 API *Iterator*

El API *iterator* de StAX representa un flujo de un documento XML como un juego objetos de eventos discretos. La aplicación saca estos eventos y el analizador los proporciona en el orden en que los lee en el documento XML origen.

La interfaz base *iterator* se llama `XMLEvent`, y hay subinterfaces por cada tipo de evento definido que se puede encontrar al analizar un documento XML, como pueden ser comienzo o fin de elemento, atributo, caracteres, etc. La interfaz primaria del analizador para leer eventos *iterator* es `XMLEventReader`, y la interfaz primaria para escribir eventos *iterator* es XMLEventWriter. La interfaz XMLEventReader contiene cinco métodos., el más importante de los cuales es nextEvent(), que devuelve el siguiente evento en el flujo XML. XMLEventReader implementa java.util.Iterator, lo que significa que lo que devuelve XMLEventReader se puede capturar o pasar a rutinas que pueden trabajar con el estándar Java Iterator.

7.6 *Eligiendo entre las APIs Iterator y Cursor*

Los autores de StAX decidieron que sería más útil definir dos APIs pequeñas y eficientes en lugar de una sobrecargada API más grande y necesariamente más compleja.

7.6.1 Comparación entre las APIs *Iterator* y *Cursor*

Antes de elegir entre las APIS *iterator* y *cursor* se deben conocer ciertas características disponibles con la API *iterator* que no se tienen con la API *cursor*:

- Los objetos creados de las subclases de `XMLEvent` son inmutables, y se pueden usar en tablas, listas y mapas, y se pueden pasar por las aplicaciones incluso después de que el analizador haya pasado a otros eventos.
- Se pueden crear subtipos de `XMLEvent` que son objetos de información completamente nueva o extensiones de objetos existentes pero con métodos adicionales.
- Se pueden añadir y quitar eventos de un flujo de eventos XML de forma mucho más simple que con la API *cursor*.

7.6.2 Recomendaciones para efectuar la elección

A continuación se presentan algunas recomendaciones que conviene tener en mente a la hora de elegir entre las APIs *iterator* y *cursor*:

- Si se está programando para un entorno con limitaciones de memoria, como J2ME [12], se puede hacer un código más eficiente con la API *cursor*.
- Si el rendimiento es la principal prioridad – por ejemplo, cuando se crean bibliotecas o una infraestructura de bajo nivel – la API *cursor* es más eficiente.
- Si se quieren crear conductos de proceso XML, mejor la API *iterator*.
- Si quiere modificar un flujo de eventos, mejor la API *iterator*.
- Si se quiere que la aplicación sea capaz de manejar procesado del flujo de eventos que sea conectable, mejor la API *iterator*.
- En general, si no se tiene una especial preferencia por uno u otro, se recomienda el uso de la API *iterator* ya que es más flexible y extensible, preparando para el futuro las aplicaciones.

7.7 *Conclusiones y elección de una API para el proyecto*

Se ha elegido la API de analizado XML StAX ya que se necesita para el proyecto de capacidad de escribir XML, así que SAX queda descartada automáticamente. Además, debido a sus pequeños requisitos de memoria y capacidad de procesado, alto rendimiento y facilidad de utilización, con un tiempo de aprendizaje realmente corto para un programador Java, resulta ideal para su aplicación en el proyecto. Además, debido a su modelo de *pull parsing* permite que el cliente tenga el control sobre su comportamiento, facilitando así leer sólo una parte del documento XML cuando sólo se requiera algún elemento, y no el documento completo, como es el caso de la necesidad de lectura de XML del proyecto.

Entre las dos APIs que contiene StAX se ha optado por la utilización de la *cursor* debido a que no se necesita modificar el flujo de eventos que se trata, ya que cuando se van a escribir en el documento ya están en su estado final, y las necesidades de lectura son muy pequeñas, y sin tener que modificar nada. Además, la API *cursor* proporciona una mayor velocidad de puesta en marcha del uso de la API, requiriendo menos programación, en casos en los que, como éste, no se tienen grandes requisitos de tratamiento XML. Y, finalmente, al utilizar esta API, se obtiene un mayor rendimiento y menor uso de memoria que con la *iterator*.

8. Conclusiones

XML es una tecnología cada vez más utilizada que constituye la capa más baja dentro del nivel de aplicación, sobre el que se puede montar cualquier estructura de tratamiento de documentos. El uso de XML permite la representación de los datos correspondientes a un test o ítem según la norma QTI del IMS de forma directa, permitiendo que estos documentos se puedan utilizar en cualquier entorno de aplicación.

Para el tratamiento de documentos XML existen diversos métodos, adecuados para distintos entornos, según las necesidades de cada uno, y de si se necesita leer y escribir, o sólo leer documentos XML.