Informe Técnico – Technical Report
DPTOIA-IT-2001-003
Diciembre, 2001

XML y Comercio Electrónico

Manuel José Hernández Gajate
Francisco J. García Peñalvo

Departamento de Informática y Automática
Universidad de Salamanca

Revisado por:

Dra. María N. Moreno García

Departamento de Informática y Automática

Universidad de Salamanca

mmg@usal.es

Dr. Ángel Luis Sánchez Lázaro

Departamento de Informática y Automática

Universidad de Salamanca

angeluis@tejo.usal.es

Aprobado en el Consejo de Departamento de 19 de diciembre de 2001

Información de los autores:

D. Manuel José Hernández Gajate

Ingeniero Informático

mjhdezg@hotmail.com

Dr. Francisco José García Peñalvo

Área de Ciencia de la Computación e Inteligencia Artificial

Departamento de Informática y Automática

Facultad de Ciencias - Universidad de Salamanca

Plaza de la Merced S/N – 37008 – Salamanca

fgarcia@usal.es

Este trabajo ha sido parcialmente financiado por la Junta de Castilla y León y la Unión Europea a través del Fondo Social Europeo mediante el proyecto de investigación SA002/01

Resumen

XML (*eXtensible Markup Language*) se puede definir como una tecnología orientada a la definición de lenguajes de etiquetas de propósito general. Tomando como partida **SGML** (*Standard Generalized Markup Language*) se pretende obtener un nuevo metalenguaje que, manteniendo la capacidad expresiva y la compatibilidad de SGML, goce de una sencillez similar a la de **HTML** (*Hypertext Markup Language*).

Como todo buen lenguaje, XML define una sintaxis muy estricta basada en una serie de elementos bien conocidos (etiquetas, atributos, instrucciones de procesamiento...) que se combinan siguiendo una serie de reglas sintácticas precisas. Pero además, un documento XML posee una semántica subyacente que permite otorgar significado concreto a los elementos sintácticos en sí.

Aunque se habla de XML únicamente, dicho concepto se sustenta sobre un conjunto mayor de tecnologías que en conjunto forman una sinergia de primer orden.

Como descendiente de SGML, XML ha heredado las **DTD** (*Data Type Document*) que permiten definir la semántica de los documentos. Pero no se ha quedado ahí. La tecnología **XML Schema** supera las DTD proporcionando una sintaxis mucho más rica que permite definir la carga semántica de cada elemento dentro de un documento de forma más precisa. Gracias a las DTDs y a los *XML Schemas* un documento no sólo se puede considerar **bien formado** de acuerdo a las reglas sintácticas, sino **válido** conforme al significado de sus elementos.

XML Namespaces da soporte a toda la familia XML consiguiendo que los elementos dentro de un documento sean únicos y de esta forma no pierda su carga semántica. Este aspecto es fundamental para garantizar que la distribución de los documentos por la Red no suponga ningún problema.

La estructura sintáctica de XML no es casual y su morfología en árbol facilita su manejo. Así, formando parte de la familia XML, se encuentra **XPath** y **Xpointer**, dos lenguajes de expresiones que permiten desplazarse por las distintas partes de un documento de forma sencilla e intuitiva. Por supuesto, estas dos tecnologías no tienen ninguna utilidad de forma aislada pero sirven de apoyo a otras.

Un principio fundamental en la concepción de XML es la independencia entre datos y representación. De esta forma, un mismo documento puede ser visualizado de diferentes formas, y en diferentes medios sin necesidad de modificar los datos originales. Esto se logra gracias a **XSL** (*eXtensible Style Language*). Esta tecnología proporciona los mecanismos necesarios para procesar un documento XML válido, introduciendo todos aquellos aspectos de visualización que sean necesarios. El primero de ellos es **XSLT** (*XSL Transformation*) que empleando expresiones **XPath** y **Xpointer** y directivas concretas permite transformar el documento en otro al que se puede incorporar aspectos de visualización con **XSLFO** (*XSL Formatting Objects*), o con otra tecnología como HTML, **CSS** (*Cascading Style Sheets*), **WML** (*Wireless Markup Language*), etc.

XML también incluye la posibilidad de enlazar documentos. **XML Link** supera el concepto de *hyperlink* de HTML permitiendo relaciones más flexibles.

Puesto que en última instancia son las aplicaciones las que deben manejar los documentos, existen dos APIs (*Application Program Interface*) para el manejo de documentos XML: **SAX** (*Standard API for XML*) y **DOM** (*Document Object Model*). La gran aceptación de XML ha supuesto que numerosos fabricantes de *software* (Oracle, Sun Microsystems...) hayan implementado dichas API con lo que están disponibles para la mayoría de lenguajes de programación y plataformas.

Abstract

XML (*eXtensible Markup Language*) may be defined as a general-purpose label definition technology. Taking **SGML** (*Standard Generalized Markup Language*) as a starting point, it tries to produce a new metalanguage which, preserving SGML's expressive capabilities and compatibility, may enjoy a degree of ease of use similar to that of **HTML** (*Hypertext Markup Language*).

Like any other good language, XML defines a very strict syntax, which consists of a series of well-known elements (labels, attributes, processing instructions...) that can be combined in terms of precise syntactic rules. Further, an XML document possesses an underlying grammar that can give a precise meaning to syntactic elements in themselves.

Although we are only dealing with XML, this concept is based on a larger set of technologies, which, when put together, offer a first-order synergy.

As a descendant from SGML, XML inherits **DTDs** (*Data Type Documents*), which permit the semantics of documents to be defined. But this is not all. The **XML Schema** technology goes beyond DTDs, and offers a much richer semantics, which allows us to define mor precisely the semantic charge of the elements within a document. Thanks to DTDs and to XML Schemas, a document can be considered not just **well formed** according to syntactic rules, but also **valid** as far as the meaning of its elements is concerned.

XML Namespaces supports all of the XML family, thus making unique the elements within a document, and keeping them from loosing their semantic charge. This is essential in order to guarantee a problem-free distribution of documents through Internet.

XML's syntactic structure did not happen by chance, and it tree-like morphology eases it use. Thus, one can find **XPath** and **Xpointer** as a part of the XML family: they are expression languages that offer a simple and intuitive way of navigating through a document. Of course, these technologies are meaningless when isolated, but the support other technologies.

One of the mainstays in the conception of XML is the independence of data and their representation. Thus, the same document may be displayed in different ways, and on different media, with no need to modify the original data. This is done by means of an **XSL** (*eXtensible Style Language*). This technology offers the mechanisms used to process a valid XML document, introducing any visualization aspects that may be needed. The first of these mechanisms is **XSLT** (*XSL Transformation*), which, by means of **XPath** and **Xpointer** expressions, and through the use of precise directives, makes it possible to translate the document into another into which one may incorporate visualization aspects by means of **XSLFO** (*XSL Formatting Objects*), or by using some other technology like HTML, **CSS** (*Cascading Style Sheets*), **WML** (*Wireless Markup Language*), etc.

XML includes the ability to link documents. **XML Link** takes the concept of *hyperlink* of HTML one step further, allowing more flexible relationships.

Since it is applications that must finally handle documents, there exist two APIs (*Application Program Interface*) for handling XML documents: **SAX** (*Standard API for XML*) and **DOM** (*Document Object Model*). The wide acceptance of XML has prompted many

software developers (Oracle, Sun Microsystems...) to implement these APIs, thus making them available from most programming languages and platforms.

Tabla de Contenidos

1. Introducción

XML (*eXtensible Markup Language*) [Bray et al., 2000] se puede definir como una tecnología basada en el estándar de definición de lenguajes SGML (*Standard Generalized Markup Language*) [Wohler, 1994], que permite diseñar lenguajes de marcado, con lo que se puede definir XML como un metalenguaje.

Puesto que es un subconjunto del estándar SGML, los documentos XML están de acuerdo con las especificaciones SGML. Además, XML pretende ser un lenguaje de marcado con las ventajas nucleares de SGML pero con la simplicidad relativa de HTML (*HyperText Markup Language*) [Raggett, 1999].

El concepto de XML se apoya sobre toda una serie de lenguajes y elementos y que dan soporte a esta tecnología.

El primero de ellos es **XML Schema** [Fallside, 2001] [Thompson et al., 2001] [Biron y Malhotra, 2001]. Un documento XML está descrito siguiendo una sintaxis concreta. Pero, además, XML permite definir la semántica de los datos. Esto significa que cuando se genera un documento XML, se puede establecer su corrección sintáctica y semántica.

El germen de XML Schema es XML Data, idea inicial que surgió en el W3C (*World Wide Web Consortium*). Se intenta implementar un mecanismo propio, definido mediante sintaxis XML, que permita definir la semántica y que permita superar las limitaciones de las DTD (*Document Type Definition*) [Navarro et al., 2000] heredadas de SGML.

XSL (*eXtensible Style Language*) [Navarro et al, 2000] es el lenguaje que permite establecer la forma en la que se van a presentar los datos. De esta forma, se separan los datos, propiamente dichos, de la forma en la que éstos se muestran.

XSL, en realidad, está soportado sobre dos tecnologías: el lenguaje XSLT (XSL *Transformation*) [Clark, 1999] y el lenguaje XSLFO (XSL *Formating Objects*) [Clark, 1999]. Además, XSL incorpora elementos de otras tecnologías XML tales como XPath y Xpointer [Navarro et al., 2000].

El lenguaje **XLL** (*eXtensible Linking Language*) [DeRose et al., 2000] va a permitir definir la forma en la que establecer enlaces entre distintos documentos XML, así como enlaces a recursos concretos dentro de un determinado documento, desde el mismo documento o desde otro externo. Este lenguaje a su vez se sustenta sobre tres conceptos principales: XLink, XPath y XPointer.

Por último, los **XML Namespaces** [Morrison, 2000] no son, a diferencia de los elementos anteriores, un lenguaje de definición, sino un soporte global a toda la tecnología. Aseguran que los identificadores utilizados dentro de los distintos documentos sean correctos desde el punto de vista de la unicidad de nominación, evitando así que, en ningún momento, se produzcan colisiones de nombres que provocarían ambigüedad en el tratamiento de los documentos.

La organización de este documento es la siguiente. En primer lugar, se muestra una pequeña visión histórica de la génesis de XML y cómo ha ido evolucionando dicho lenguaje.

En el apartado 3, se muestra cómo se estructura un documento XML. Se empezará describiendo cómo se organiza desde un punto de vista puramente sintáctico, para luego centrarse en la componente semántica. En los apartados 4 y 5, se mostrarán las DTD y los esquemas XML como elementos que permiten especificar la semántica de los documentos, para en el apartado 6, mostrar una comparativa de ambos métodos.

En el apartado 7, se introducen los espacios de nombres como elemento esencial que sirve de soporte a toda la tecnología permitiendo la unicidad de nominación de cada uno de los elementos dentro de un documento XML.

En el apartado 8, se introduce el lenguaje XSL. Se analizarán las características de este lenguaje y cómo se puede utilizar para transformar un documento XML en otro documento, con vistas, principalmente, a la visualización de los contenidos.

Los mecanismos de enlace entre documentos XML presentes en distintas localizaciones, así como los métodos para hacer referencia a partes concretas dentro de un documento, se analizarán en el apartado 9 donde se verá XLL y XPointer.

La programación sobre documentos XML y las APIs necesarias para dicho tratamiento se aborda en el apartado 10.

Por último, los apartados 11 y 12 se centran en la introducción de algunos de los usos de XML así como las iniciativas en el ámbito del comercio electrónico que se sustentan sobre este lenguaje.

2. SGML y XML

No es el propósito de este apartado explicar todo lo que es SGML, sino mostrar qué características importantes, de las que goza SGML, ha tomado XML y qué aspectos no son tan deseables y que, por lo tanto, XML a desechado.

SGML nació como un lenguaje de definición de lenguajes, y concretamente de lenguajes de marcado.

Proporciona los mecanismos necesarios para poder definir los componentes que van a formar parte del lenguaje, tales como delimitadores de etiquetas, etiquetas...

Para definir la estructura e incluso contenido de todos y cada uno de los componentes que van a formar parte de la especificación del lenguaje, es decir la estructura del propio lenguaje, toda esta especificación se acompaña de un mecanismo para la validación de documentos: las DTD. De esta forma, se define sintaxis y semántica.

Estas dos características vistas, **estructura** y **validación**, han sido incorporadas por XML.

El inconveniente de SGML viene condicionado por su propia definición. Su objetivo es tan general que su estructura es demasiado compleja lo que dificulta tanto su aprendizaje, como su explotación. XML pretende evitar este aspecto con el objetivo claro de que pueda ser fácilmente adoptado. De esta forma, XML se perfila como un subconjunto de SGML.

3. Documentos en XML

En un documento XML existen dos elementos bien diferenciados: la **sintaxis** y la **semántica**. XML no impone una forma de estructurar determinada, siendo el usuario el que establece qué elementos de marcado usar, atributos de dichos elementos, restricciones para dichos elementos...

La definición de sintaxis y semántica está sujeta a fuertes reglas que permiten evitar ambigüedades y que ayudan a conseguir una homogeneidad en todos los documentos XML. Este tipo de normas se verán a medida que se detallen los elementos XML.

3.1. Sintaxis en XML

XML se basa en el etiquetado como forma de definir los contenidos de un documento.

El elemento principal de construcción de un documento XML es la **entidad**. Una entidad está compuesta por dos tipos de datos:

- Datos procesados sintácticamente (*<tag>..</tag>*).
- Datos no procesados sintácticamente.

Los datos procesados sintácticamente proporcionan la estructura al documento mientras que los datos no procesados sintácticamente son los que definen el contenido real de información del documento.

Bajo la terminología XML, un documento se puede estructurar desde dos puntos de vista:

Desde un punto de vista físico. El documento se ve como un conjunto de entidades.

Desde un punto de vista lógico. El documento es un conjunto de elementos así como las relaciones que se establecen entre ellos.

La sintaxis XML cuenta con los siguientes elementos de marcado:

3.1.1 Etiquetas

Son cadenas de texto que permiten describir elementos. Estas etiquetas permiten estructurar el documento. Esta estructura no viene impuesta en ningún momento, siendo el propio usuario el que define qué etiquetas definen mejor los elementos de su documento.

Las restricciones para la construcción de etiquetas son las siguientes:

- Toda etiqueta tiene la forma: *<nombre_etiqueta>*.
- Toda etiqueta debe tener su correspondiente etiqueta de cierre: *</nombre_etiqueta>*.
- El contenido se localiza entre la etiqueta de apertura y la de cierre. Si una etiqueta es vacía, es decir, carece de contenido, se puede representar en la siguiente forma: *<nombre_etiqueta />*.

Las etiquetas pueden contener a su vez atributos que se especificarán del siguiente modo:

<nombre_etiqueta nombre_atributo="valor"...>...</nombre_etiqueta>.

El número de atributos, así como sus posibles valores, también son definidos por el usuario. Es obligatorio que el valor del atributo vaya entrecomillado.

Dentro del contenido de una etiqueta, pueden aparecer nuevas etiquetas, que se denominarán hijas. Se forma así una estructura jerárquica, con forma de árbol, encabezada por un elemento raíz, que no es más que una etiqueta que se considera como tal y que engloba al resto.

3.1.2 Instrucciones de procesamiento

Estos elementos presentan la siguiente forma: *<? ....?>*.

Constituyen directivas que no son tratadas por el analizador XML y que deben proporcionársele a la aplicación pues contienen información valiosa para ésta.

Existe una directiva obligatoria que encabeza todo documento XML que es *<?xml version="1.0"?>*, y que indica que se trata de un documento XML versión 1.0. Otras directivas que también pueden aparecer se especificarán más adelante.

3.1.3 Comentarios

En un documento XML, también pueden aparecer comentarios con la sintaxis <!-- comentario -->.

3.1.4 Referencia de entidades

Presentan la forma *&nombre;*. Son procesadas por el analizador XML y hacen referencia a una determinada entidad definida con anterioridad. En analizador sustituye su referencia por el valor que se haya asignado.

Existen algunas definidas por defecto como son: *'* (identifica el símbolo ‘), *&* (identifica a &), *"* (identifica a “), *<* (identifica a <) y *>* (identifica a >).

3.1.5 Declaraciones de tipo de documento

Con la forma *<!DOCTYPE elemento_raiz ...>* se establece la definición de datos, en forma de DTD, con la que validar el documento. En el siguiente apartado se proporcionan más detalles acerca de este tipo de elemento de marcado.

3.1.6 Secciones marcadas

Contiene texto que no se desea que se analice sintácticamente. Su formato es *<![CDATA[datos]]>*.

3.2. Gramática en XML

Las distintas etiquetas y atributos de las mismas, así como su contenido puede ser creado a gusto y necesidad del usuario, pero no tiene ningún significado si no se le acompaña de una semántica. Además, esta semántica va a permitir determinar si un documento es válido.

Se pueden considerar dos tipos de documentos desde el punto de vista de su corrección:

- **Documentos bien formados**. Documentos que siguen las especificaciones sintácticas de XML. En muchos casos, es suficiente que el documento esté bien formado para trabajar con él.
- **Documentos válidos**. No sólo son documentos bien formados, sino que además están de acuerdo con una semántica definida.

La semántica se establece a través de las Definiciones de Tipos de Datos (DTD) o a través de los esquemas[1].

Las restricciones a la estructura de contenidos se pueden englobar dentro de dos grandes grupos:

- Modelo de contenidos.
- Tipos de datos de documento.

En el modelo de contenidos se definen aspectos tales como:

- Qué etiquetas se permiten dentro del documento, indicando cuál es la etiqueta raíz.

[1] Esquema es un concepto general que describe un modelo de datos y XML Schema es una tecnología específica de esquemas que sustituye a las DTD.

- El orden de aparición de las mismas, así como aspectos tales como el carácter opcional de aparición de etiquetas, contenido de las mismas, anidamiento de etiquetas...
- En lo referente a los atributos, qué atributos aparecerán dentro de qué etiquetas, opcionalidad de dichos atributos, qué tipos de valores pueden contener los atributos, valores por defecto...

En lo referente a los tipos de datos de documento, comentar que las DTD incorporan una funcionalidad limitada para definir tipos de datos. Ésta es una de las limitaciones que intentan superar los esquemas.

Todos estos aspectos, y muchos más, se deben especificar con el fin de que, no sólo un documento XML tenga significado para el que lo crea sino que lo tenga para todo aquel que lo lea, incluido el procesador XML.

Por supuesto, no es necesario que exista una semántica definida para poder crear un documento XML y trabajar con él, pero nunca sería posible validar dicho documento. En entornos, en los que el formato de los documentos sea bien conocido se puede omitir la especificación semántica.

A continuación, se pasará a describir de forma breve la forma de construir una DTD así como un XML Schema.

4. DTD

Aunque las DTD no son un mecanismo propio de XML, se van a mostrar de forma resumida sus características para luego introducir XML Schema y, de esta forma, el lector pueda sacar sus propias comparativas.

Las DTD son un mecanismo que permite definir la gramática de un documento SGML, y concretamente la gramática de un documento XML.

Su objetivo primordial es proporcionar al procesador XML una base sólida para determinar la validez del documento.

Dependiendo del lugar donde se defina la DTD, estas se pueden clasificar en:

- Externas. Existe un documento adjunto que incluye toda la declaración.
- Internas. Las declaraciones están dentro del propio documento.
- Mixtas. Se incorpora una DTD externa al documento y se le añaden nuevas definiciones dentro del propio documento. En este caso, las especificaciones locales tienen prioridad sobre las externas pudiendo sobrescribirlas.

Para incorporar una DTD externa se realiza a través de lo que se ha denominado **referencia de tipo de documento**: *<!DOCTYPE nombre SYSTEM "URI"> o <!DOCTYPE nombre PUBLIC "dominio público">*.

El campo *nombre* coincide con el nombre del elemento raíz dentro del documento. Este aspecto es obligatorio y debe cumplirse.

El elemento *SYSTEM* indica que se trata de un recurso externo y *URI* (*Universal Resouce Identifier*) es el identificador válido que indica dónde se encuentra la especificación.

El elemento PUBLIC, alternativo a SYSTEM, también identifica un recurso externo, pero en la forma de un identificador de dominio universalmente accesible.

Las DTD locales se definen dentro del elemento *DOCTYPE* entre corchetes: *<!DOCTYPE [definiciones]>*.

Si se desea utilizar la solución mixta *<!DOCTYPE nombre SYSTEM "URI" [..más definiciones...]>*.

Una DTD define los siguientes aspectos:

- Los tipos de elementos que se permiten en el documento, así como sus modelos de contenido.
- Los atributos de cada elemento.
- Las entidades que se permiten en el documento.
- Las notaciones que permiten usar las entidades externas.

A continuación, se procederá a analizar cada uno de los elementos de una DTD.

4.1. Elementos

Es el componente principal de una DTD. Su formato es: *<!ELEMENT nombre tipo>*.

El elemento *nombre* es un identificador único dentro del ámbito de la DTD.

El elemento *tipo* indica el tipo de elemento que se está declarando. Los tipos posibles son:

- Vacío. El elemento no puede tener contenidos: *<!ELEMENT nombre EMPTY>*.
- Sólo elementos. El elemento sólo puede contener otros elementos secundarios: *<!ELEMENT nombre lista_elementos>*.
- Mixto. Puede contener tanto datos como elementos anidados: *<!ELEMENT nombre (#PCDATA | lista_elementos)>*.
- Any. Tipo especial de elemento mixto que carece de estructura: *<!ELEMENT nombre ANY>*.

Para el caso de elementos que sólo pueden contener elementos, así como para los mixtos, la forma de especificar su contenido se basa en la notación EBNF (*Extended Backus-Naur Form*).

A través de los mecanismos proporcionados por esta notación, es posible especificar la cardinalidad de aparición de un elemento: "*" indica 0 o varias veces, "?" indica 0 ó 1 vez...

4.2. Atributos

Permiten incluir características adicionales a los elementos. Considerar que una información acerca de un elemento es atributo de éste, o contenido o nuevos elementos adicionales, es siempre subjetivo. Normalmente, los atributos de un elemento constituyen información concreta, simple (que no necesita elementos asociados) y que no puede constituirse como elemento por sí misma.

Su sintaxis es:

<!ATTLIST nombre_elemento nombre_atributo tipo_atributo valor_defecto>.

El campo *nombre_elemento* identifica al elemento al que pertenece el atributo.

El campo *nombre_atributo* es el nombre del atributo que debe ser único dentro del elemento.

El campo *tipo_atributo* identifica el tipo. Se hablará de los tipos más adelante.

El campo *valor_defecto* indica el valor por defecto que puede tomar el atributo. Puede tomar los siguientes valores:

- #REQUIRED. El atributo es obligatorio.
- #IMPLIED. El atributo es opcional.
- #FIXED *valor*. Toma un valor fijo especificado por *valor*.
- Valor por defecto. Un determinado valor.

4.2.1 Tipos de atributos

Atributos tipo cadena

Son atributos que pueden contener cadenas de caracteres. Se declaran del siguiente modo: *<!ATTLIST nombre_elemento nombre_atributo CDATA valor_defecto>*. El tipo CDATA incluye todos los tipos de datos primitivos. Las DTD no cuentan con mecanismos para especificar tipos de datos concretos, numérico, lógico...

Atributos enumerados

Un atributo enumerado se caracteriza porque los posibles valores que puede tomar están dentro de un conjunto prefijado de elementos. Su declaración sería la siguiente: *<!ATTLIST nombre_elemento nombre_atributo (valor1 | valor2 | ...) valor_defecto>*.

Un tipo especial de atributo enumerado es el atributo de notación, que especifica un conjunto de notaciones declaradas con anterioridad en el documento. Se declaran anteponiendo la palabra NOTATION a la lista de posibles opciones. *<!ATTLIST nombre_elemento nombre_atributo NOTATION (notación1 | notación2 | ...) valor_defecto>*.

Atributos con símbolo

Dentro de esta clase general de atributos se encuentran los siguientes atributos:

- ID, IDREF e IDREFS. El atributo ID permite asociar identificadores únicos a los elementos dentro de una DTD. IDREF permite hacer referencia a un elemento con un cierto ID que ya ha sido declarado. IDREFS permite incluir una lista de ID de elementos ya existentes.
- ENTITY y ENTITIES. Permiten definir un atributo como una referencia a una entidad. Concretamente este tipo de atributos permite incluir en un documento entidades externas no analizadas sintácticamente. Este concepto se tratará más adelante. Por otro lado, ENTITIES permite incluir una lista de referencias a entidades.
- NMTOKEN y NMTOKENS. Especifican atributos que contienen valores de símbolos con nombre. Para este tipo de atributos, su valor consta de un solo nombre, sin espacios en blanco. Se compone de caracteres alfanuméricos además de los símbolos "**:**", "**.**", "**_**" y "**-**".

4.2.2 Declaración múltiple de atributos

Se pueden agrupar declaraciones de atributos de un mismo elemento sin más que enumerar cada uno de los atributos uno detrás de otro.

<!ATTLIST nombre elemento

nombre_atributo1 tipo_atributo1 valor_defecto1

nombre_atributo2 tipo_atributo2 valor_defecto2 ...>

4.3. Entidades

Las entidades definen la estructura física de los documentos. Representan una unidad de almacenamiento. Cada entidad tiene un nombre único, con la salvedad de la denominada **entidad de documento** que carece de él. Esta entidad hace referencia al documento en sí propiamente dicho, y es la única entidad que existe siempre.

Toda entidad, salvo la entidad de documento, debe ser declarada antes de ser usada.

Las entidades dependiendo de su localización física se pueden clasificar en:

- Internas. Declaradas dentro del documento donde se referencian.
- Externas. Son entidades que están en un documento diferente del documento en el que se referencian.

Por otro lado, atendiendo a si son procesadas o no por el analizador XML se pueden clasificar en:

- Analizadas sintácticamente. Entidades que tienen contenido XML que va a ser procesado. Por lo tanto, sólo pueden tener contenido de texto. Estas entidades acaban formando parte del propio documento desde el que se referencian.
- No analizadas sintácticamente. Entidades que hacen referencia a recursos de diferente naturaleza que no son procesados por el analizador XML, tales como ficheros binarios... Para el manejo de estas entidades es necesario asociar notaciones que permitan incluir información adicional de cómo procesarlas. El concepto de notación se verá más adelante.

Todas las entidades internas son analizadas sintácticamente mientras que las externas pueden serlo o no.

Dentro de las entidades analizadas sintácticamente se pueden identificar dos tipos:

- Entidades generales. Son definidas en el documento de definición de datos y utilizadas en el propio documento XML. Se declaran como *<!ENTITY nombre "valor">*. En el documento XML, se pueden utilizar mediante *&nombre;*.
- Entidades de parámetro. Son declaradas y usadas en el documento de definición de datos. Permiten agrupar patrones o conjuntos de datos que se repiten con frecuencia haciendo así el diseño de la DTD mucho más modular y elegante. Se declaran de forma similar a las anteriores pero anteponiendo al nombre un % separado por un espacio: *<!ENTITY % nombre "valor">*. Para hacer referencia a ellas se emplea: *%nombre;*.

Anteriormente se ha definido un tipo de entidades denominadas externas que se caracterizan porque se ubican fuera del documento desde el cual se referencian. La forma de hacer referencia a este tipo de entidades depende de si deben ser analizadas sintácticamente o no.

Para hacer referencia a entidades externas que son analizadas sintácticamente, es necesario incluir la siguiente directiva: *<!ENTITY nombre SYSTEM "URI">* o *<!ENTITY nombre PUBLIC "dominio">*. La primera se usa cuando la entidad se encuentra en local o en una determinada red. La segunda cuando la entidad es de uso público y se encuentra en un dominio universalmente accesible.

En el caso contrario, es necesario incluir la palabra reservada *NDATA* para evitar que el analizador XML intente analizarlas: *<!ENTITY nombre SISYTEM "URI" NDATA nombre_notación>*.

Una vez referenciada la entidad externa puede ser utilizada únicamente como un atributo del tipo *ENTITY* o *ENTITIES*:

En el documento de definición de datos:

<!ENTITY nombre_entidad SYSTEM "URI">

<!ATTLIST nombre_elemento nombre_atributo ENTITY valor_defecto>

En el fichero XML:

<nombre_elemento nombre_atributo="nombre_entidad">..</nombre_elemento>

4.4. Notaciones

Las notaciones permiten incluir información adicional acerca de cómo manejar las entidades externas no analizadas sintácticamente. Normalmente, permiten indicar qué programas externos deben ejecutarse para tratar la entidad.

Su sintaxis es: *<!NOTATION nombre_notación SYSTEM "URI recurso">*.

5. XML Schema

Este lenguaje, al igual que las DTD, va a permitir definir la gramática de un documento XML, pero con una diferencia sustancial respecto a las DTD: la propia especificación está escrita siguiendo la notación XML.

XML Schema incorpora ciertas mejoras importantes frente a las DTD[2]:

- Las DTD son un mecanismo fijo que no es extensible, principio que choca frontalmente con la filosofía XML. XML Schema presenta un modelo de datos abierto.
- XML Schema está descrito en XML por lo que se suaviza la curva de aprendizaje al no tener que aprender un nuevo lenguaje.
- Enriquece la semántica de las DTD permitiendo definir tipos de datos, incluir restricciones sobre tipo de datos ya existentes...
- Permite incorporar los espacios de nombres. Este concepto se verá más adelante.

El concepto de espacio de nombre es un aspecto fundamental que afecta a los esquemas. Es aconsejable comprender este concepto previamente para evitar caer en confusión. En el apartado 7, se explica el concepto de los espacios de nombres.

A continuación, se describirán distintos aspectos de XML Schema de forma breve, siempre tomando como base la especificación del W3C (*Wold Wide Web Consortium*).

5.1. Conceptos principales

En este apartado, se detallan los principales aspectos de los esquemas XML, centrándose en las diferencias más importantes con respecto a las DTD.

Dada la riqueza de notación de los esquemas, únicamente se prestará atención a aquellos elementos base para la comprensión de las características más importantes.

[2] Una comparación más profunda entre las DTD y XML Schema se realizará en el apartado "XML Schema VS DTD".

5.1.1 Elementos y atributos

Al igual que en las DTD, el componente principal de un esquema es el elemento. Los elementos identifican cada una de las etiquetas que van a aparecer en el documento XML.

Los elementos se especifican a través de la directiva *element*[3]. Dicha directiva especifica, entre otros datos el nombre del elemento y el tipo de datos.

El modelo de contenido de un elemento, es decir qué atributos tiene, qué elementos anidados tiene... se especifica a través de los tipos de datos, que se verán en el siguiente apartado.

El primer elemento que aparece dentro de la especificación del esquema es el elemento raíz, y es el único elemento indispensable dentro de un esquema.

Los elementos se pueden clasificar atendiendo al alcance de su definición en dos tipos:

- Globales. Son elementos accesibles desde cualquier parte del documento de esquema. El elemento raíz es un ejemplo de elemento global.
- Locales. Son elementos que se definen dentro de una definición de tipos de datos.

El alcance de los elementos va a tener repercusión en los conflictos de nominación de elementos dentro de un esquema. Se analizarán dichos conflictos en el siguiente apartado.

Además, los elementos se pueden clasificar dentro de un esquema, y atendiendo a su contenido en los siguientes grupos:

- Sólo contenido. No tienen elementos anidados.
- Mixtos. Elementos que alternan subelementos y contenidos de datos.
- Sin contenidos. En la especificación del elemento, no indicar ningún tipo de contenido, salvo quizás atributos.

En lo referente a los atributos, ya no se establecen como un elemento aislado dentro del esquema como pasaba en las DTD, sino que ahora su declaración está incluida dentro de la declaración de tipos de datos.

5.1.2 Tipos de datos

XML Schema maneja tipos de datos y, concretamente, los clasifica en dos tipos: tipos de datos complejos y tipos de datos simples.

En XML Schema existe una diferencia básica entre los tipos de datos complejos que pueden contener elementos y llevar atributos y los tipos simples que no pueden contener ni elementos ni atributos. Así, un tipo de datos complejo permitirá definir todos aquellos elementos que tengan anidados otros elementos y/o tengan atributos, mientras que un tipo de datos simple permitirá definir, únicamente, elementos que sólo tengan contenido.

Con la existencia de tipos de datos, se supera la limitación de las DTD a la hora de definir el modelo de contenidos de los elementos de un documento.

Tipos simples

Dentro de este grupo, se encuentran los definidos en la propia especificación. Además, el propio usuario puede definir sus propios tipos de datos simples.

[3] Acudir a http://www.w3c.org para ver la especificación completa de esta directiva.

Como ya se ha comentado, no pueden contener atributos ni otros elementos.

Es posible derivar nuevos tipos de datos simples a partir de los ya definidos sin más que añadir un conjunto de restricciones, es decir definiendo un subconjunto de valores válidos dentro de un cierto tipo. De esta manera, se personalizan los datos de forma que se adapten mejor a las necesidades específicas.

Existe un elemento, *restriction*, que permite incluir dichas restricciones. Se establece el tipo base y se incluyen aquellas características deseables mediante distintos mecanismos: expresiones regulares, rangos de valores permitidos, lista de valores permitidos... La forma de especificar las restricciones depende del tipo de datos base (por ejemplo, un tipo numérico admite restricciones que no admite un tipo cadena, tales como rango de valores permitidos...).

Existen dos tipos de datos simples que merece la pena destacar: las listas y las uniones.

Tipos de datos complejos

Los tipos de datos complejos se declaran haciendo uso de los elementos *complexType.* Mediante este tipo de datos se pueden definir contenidos complejos para un elemento dado. Es la forma de determinar qué atributos contiene un determinado elemento, el valor para dichos atributos, qué elementos aparecerán anidados, en qué orden...

Los elementos que contiene el tipo de datos se declaran, como ya se ha visto, usando el elemento *element* y los atributos usando el elemento *attribute*.

Algunas de las características más importantes de los tipos de datos son las que se muestran a continuación.

Los elementos declarados dentro de un tipo de datos son locales a él, con lo que su nombre sólo debe ser único respecto al resto de elementos dentro del tipo de datos. Esto se podría asemejar a declaraciones locales. La localidad de los nombres facilita la escritura de esquemas de tamaño considerable, sobre todo en lo referente a controlar la unicidad de nominación.

Los distintos aspectos que antes estaban incluidos de forma implícita en la propia notación EBNF en las DTD (secuencia, opcionalidad...), se incorporan a XML Schema como nuevos elementos de notación o como atributos de ciertos elementos de notación, mejorando la potencia de especificación.

Se incorporan mecanismos avanzados para especificar los tipos de datos. Entre dichos mecanismos cabe destacar los grupos de elementos con nombre, los grupos de atributos con nombre y las referencias a elementos y atributos globales. Estos aspectos mejoran la legibilidad del esquema.

5.1.3 Espacios de nombres en los esquemas

Hasta ahora, los esquemas se pueden resumir, básicamente, como un conjunto de declaraciones de elementos y definiciones de tipos de datos cuyos nombres pertenecen a un espacio de nombres denominado espacio de nombres objetivo (*target namespace*).

El objetivo del espacio de nombres objetivo es el de diferenciar definiciones y declaraciones procedentes de distintos tipos de vocabulario. Una utilidad clara se puede encontrar cuando es necesario validar un documento con varios esquemas. Sería muy probable que al incluir varios esquemas, los nombres no fueran únicos con respecto a todos ellos (el mismo nombre puede identificar distintos elementos en distintos esquemas). Gracias a los espacios de nombres, es posible identificar de forma única cada uno de los nombres, puesto que cada uno de ellos pertenece a un espacio de nombres único. Esto posibilita la reutilización de definiciones y declaraciones entre esquemas.

El uso de espacios de nombres objetivos no es obligatorio. Pero, un esquema que no tenga asociado un espacio de nombres objetivo, únicamente podrá validar documentos XML que tampoco utilicen espacio de nombres. Además, no sería posible que un documento XML pudiera ser validado con varios esquemas.

5.2. Extensibilidad en los esquemas

Hasta ahora se ha analizado como los esquemas incluyen una mayor riqueza de representación que las DTD, lo que supera muchas de las deficiencias de éstas últimas.

Otra de las características que se comentaban al principio de este apartado sobre esquemas, es la **extensibilidad**. Seguidamente se examinan ciertos aspectos que permiten sostener dicha característica.

5.2.1 Tipos derivados

Como ya se ha comentado en la parte de conceptos previos, los esquemas, además de aumentar la riqueza de los tipos predefinidos de datos, permiten al usuario crear sus propios tipos de datos. A continuación, se analizarán dos formas de derivar tipos de datos a partir de otros.

Tipos derivados por extensión

Cuando un tipo de datos complejo se deriva por extensión, el modelo de contenidos de datos del nuevo elemento, incluye al anterior además del modelo de contenidos del nuevo elemento.

Esto no constituiría una forma especial de derivación de tipos de datos, si no fuera porque el nuevo tipo de datos puede ser usado en cualquier lugar donde se use el tipo derivado.

Cuando se derivan nuevos tipos de elementos de otros ya existentes, se permite trabajar con los elementos derivados donde debería aparecer el tipo primitivo (similar al polimorfismo de los sistemas orientados a objetos). Para que esto funcione, es necesario indicar qué tipo derivado, del tipo primitivo, se intenta utilizar.

Tipos derivados por restricción

La segunda forma de derivación de tipos de datos complejos, es la derivación por restricción.

Las restricciones en los tipos de datos complejos son, conceptualmente, las mismas que las de los tipos de datos simples, pero con salvedades. Se obtiene un nuevo tipo de datos subconjunto del tipo de datos origen.

En este caso, no se puede realizar la equivalencia entre el tipo de datos base y el nuevo que sí se podía hacer en la derivación por extensión.

5.2.2 Esquema en múltiples documentos

Cuando los esquemas se vuelven grandes en exceso, su mantenimiento y legibilidad puede llegar a ser costosa. Para solventar este problema, XML Schema proporciona mecanismos que permiten dividir dicho esquema en varios documentos mucho más fáciles de mantener.

Todos los documentos pertenecientes al mismo esquema deberán tener el mismo espacio de nombres objetivo. Así, una vez incluidos cada uno de los documentos en uno de entidad mayor, todas las definiciones y declaraciones de elementos y tipos de datos serán accesibles como si se tratara de un único esquema.

Esta estructura de múltiples documentos se amplía con las características de redefinición de tipos vistas anteriormente. Es decir, es posible incluir dentro de un esquema otro, con el mismo

espacio de nombres objetivo, pero con el propósito explícito de redefinir algunas de sus definiciones de tipos de datos.

5.2.3 Control de derivación

Los esquemas cuentan con artificios que permiten con cierta facilidad crear nuevos elementos y tipos de datos, e incluso esquemas completos, a partir de otros elementos, tipos y esquemas ya existentes.

Con el fin de controlar la forma en la que se puede llevar a cabo todo lo anterior, los esquemas incorporan medios que posibilitan controlar qué se puede derivar e incluso cómo se puede derivar.

Un primer procedimiento permite indicar si un tipo particular de datos no se puede derivar o si se puede derivar, se puede establecer la forma de derivación: por restricción, por extensión o ambas.

En lo referente a tipos simples, los esquemas permiten especificar aspectos concretos que pueden ser aplicados en el proceso de derivación de esta clase de tipos.

5.3. Otros aspectos de los esquemas

En el siguiente apartado, se muestra otro aspecto de los esquemas que pueden llegar a ser muy útiles para la definición de la semántica de los documentos.

5.3.1 Unicidad

XML en sí, ya proporciona medios para asegurar la unicidad: los atributos ID, IDREF e IDREFS. Este artificio también lo incluyen los esquemas, pero no son lo únicos.

Los esquemas permiten determinar si el valor de un determinado atributo o elementos debe ser único dentro de un determinado alcance[4]. Usando únicamente el procedimiento básico proporcionado por XML, no sería posible extender esta característica a los atributos. El alcance tampoco se puede especificar con los atributos ID.

Además, no sólo se puede indicar que un atributo o elemento debe ser único, sino que se puede determinar que una combinación de campos sea única.

6. XML Schema VS DTD

Una vez introducidas las dos tecnologías, se puede profundizar más en las diferencias existentes entre ambas.

Las DTD son una tecnología ampliamente utilizada y aceptada por la comunidad, y han sido el soporte para la validación de documentos desde el principio. Esto aporta ciertas ventajas entre las que cabe resaltar:

- Al ser una tecnología ampliamente utilizada, existen numerosas herramientas que permiten procesarlas.
- Está ampliamente comprobada la validez de dicho artificio.

[4] El alcance se establece a partir de expresiones XPath (se analizará este lenguaje más adelante).

Pero, las DTD presentan varios inconvenientes importantes:

- Están descritas a través de una sintaxis especializada (EBNF).
- Los contenidos de las DTD son cerrados frente a la filosofía de extensibilidad en la que se basa XML.
- Las DTD presentan un pobre soporte a la hora de definir los tipos de datos. El único tipo que definen es el CDATA.
- La flexibilidad que permite la notación EBNF es muy limitada. Esto es, en ocasiones, un inconveniente importante a la hora de definir el modelo de contenidos de un determinado elemento. Definir aspectos tales como cierto orden de aparición de los elementos anidados, opcionalidad de ciertos elementos con respecto a otros..., supone abusar de la notación EBNF con lo que se obtiene una definición oscura. En otros casos, incluso es imposible definir ciertas características.

Los esquemas XML van a aportar algunas de las características de las que carecen las DTD:

- En primer lugar, los esquemas están descritos siguiendo la notación XML, con lo que no es necesario aprender una notación nueva. Además, gracias a esto los mismos procesadores XML sirven para el tratamiento de los esquemas.
- Los esquemas presentan un modelo abierto fácilmente extensible. Incorporan diversos mecanismos para ello, tales como la herencia de elementos, entre otros, que permiten su extensibilidad sin invalidar los documentos ya existentes.
- Soportan una gran variedad de tipos de datos, a los que además pueden incorporarse elementos que definen rangos de valores permitidos, formato... El propio usuario puede crear sus propios tipos de datos. Esto no se puede llevar a cabo en una DTD.
- Al incorporar una sintaxis mucho más rica para definir el modelo de contenidos de los elementos, este modelo puede ser mucho más complejo que el definido en una DTD sin necesidad de abusar de la notación.
- Los esquemas soportan los espacios de nombres, aspecto que no puede aparecer en las DTD.
- Los esquemas incorporan mecanismos, tales como la agrupación de atributos, la definición de tipos... que permiten una mejor estructuración de los propios esquemas.

El inconveniente principal de los esquemas, hoy por hoy, es su falta de estandarización que impide que se puedan considerar al cien por cien una alternativa para la validación de documentos. A fecha de redacción de este documento, la única implementación comercial de los esquemas es la que aporta Microsoft en Internet Explorer 5.0 y 5.5.

En resumen, los esquemas se pueden considerar como justos herederos de las DTD puesto que incluyen las funcionalidades de éstas, incorporando una mayor riqueza semántica para cubrir necesidades de definición, tales como las de tipos de datos.

7. Espacios de nombres

Los espacios de nombres, como ya se comentó en la introducción, se dibujan como un artificio dentro de XML que sirve de soporte a toda la familia de tecnologías. No va a aparecer sólo dentro de los documentos XML sino que es un medio ampliamente utilizado en los *XML Schemas*, en XSL...

La idea fundamental de los espacios de nombre es la unicidad de nominación de los elementos. Puesto que en la práctica, numerosos documentos XML van a convivir, es muy probable que si se desea combinar varios de ellos surjan conflictos a la hora de denominar elementos de forma única.

Para evitar esto, es necesario un medio que permita establecer nombres únicos en un espacio global. Y este artificio son los espacios de nombres.

Los espacios de nombres se articulan en torno a los URI (*Universal Resource Identifier*), o identificadores universales de recursos[5]. Puesto que los URI son únicos proporcionan el medio necesario para el objetivo que se persigue.

Pero, los espacios de nombre, además de permitir identificar de forma única los distintos elementos, posibilitan distintas interpretaciones de los elementos dependiendo del espacio de nombrse al que pertenezcan. Es decir, una determinada etiqueta <*titulo*> no se interpretará igual si pertenece a un espacio de nombres relativo a películas, que si pertenece a un espacio de nombres de discos de música. En cada caso, identificará un objeto diferente.

Para incorporar un espacio de nombres se utiliza un atributo especial: **xmlns**. La sintaxis de este atributo es: *xmlns:prefijo=”URI”*. De esta forma, cada vez que se quiera establecer de forma explícita que un determinado elemento pertenece a un determinado espacio de nombres, se recurre a la siguiente sintaxis: <*prefijo:tag*>...</*prefijo:tag*>. Esta forma de declaración de los espacios de nombres se conoce como declaración explícita.

Existe otro tipo de declaración, denominada predeterminada, que consiste en declarar el espacio de nombres sin prefijo con lo que se aplica a todos los elementos dentro de su alcance: *xmlns=”URI”*, que estén incluidos de forma explícita en otro espacio de nombres.

El alcance de un espacio de nombres es el de todos los hijos del elemento determinado en el que se incluye. De esta forma si se establece el espacio de nombres en el elemento raíz, el espacio de nombres alcanzará a todos los elementos.

Por supuesto, es posible encontrar más de un espacio de nombres en un mismo documento. Puede existir uno predeterminado que puede ser sobrescrito para ciertos elementos por un explícito, pueden existir diferentes espacios de nombre para diferentes cometidos (por ejemplo, uno que defina los elementos y otro los tipos de datos).

8. Formato de documentos XML

Hasta este punto, se han tratado aspectos de XML referentes a su estructura sintáctica, semántica, a cómo validar un documento, además de otros aspectos como son los espacios de nombres.

En este apartado, se van a introducir los elementos principales que permiten transformar un documento XML en un formato más apto para mostrar su contenido. Esto se realizará a través de la tecnología conocida como XSL.

En primer lugar, y antes de entrar a detallar XSL, se realizará una pequeña introducción a otro lenguaje dentro de la familia XML, que es XPath, dado que XSL va a utilizar la sintaxis de este lenguaje para transformar el documento XML en un nuevo árbol formateado para ser visualizado.

---

[5] Los URI son un superconjunto que engloba a otros dos tipos de identificadores: URL y URN. Estos dos tipos también pueden ser usados para denominar un espacio de nombres.

8.1. XPath

Este lenguaje no sólo es utilizado por XSL sino también por XLink como ser verá más adelante.

XPath proporciona un mecanismo válido para moverse a lo largo de un documento XML y para hacer referencia a partes concretas dentro de un documento.

Está basado en la estructura jerárquica de los documentos XML, de tal forma que, los métodos que proporciona XPath van a basarse en dicha estructura de árbol.

Las expresiones XPath son muy similares a las rutas de acceso (*path*) a los ficheros en un sistema de ficheros tradicional tipo UNIX. El nombre que toman dichas expresiones es el de **caminos de localización**. Cada camino de localización está formado por una serie de **pasos de localización** separados, normalmente, por "/".

La evaluación de una expresión XPath da como resultado una referencia a un nodo o conjunto de nodos. XPath considera que cada uno de los elementos del documento es un nodo. Además, también son nodos en XPath los comentarios, el contenido de los elementos, los atributos...

Los distintos pasos de localización dentro de un camino XPath dirigen la búsqueda paso a paso, seleccionando nodos y profundizando en la jerarquía. Un paso de localización puede contener:

- Nodo contexto. Determinado elemento dentro de un documento que se establece como punto de partida. Por defecto, si no se indica nada, es el nodo raíz.
- Eje. XPath incorporar mecanismos avanzados, basados en la estructura jerárquica, para hacer referencia a partes del documento. Estos elementos son:
  - self. Hace referencia al propio nodo.
  - parent. Hace referencia al elemento inmediatamente anterior en la jerarquía respecto al nodo actual.
  - ancestor. Identifica todos los nodos por encima en la estructura jerárquica del nodo actual. Mientras *parent* identifica a un solo nodo, *ancestor*, a un conjunto completo.
  - preceding-sibling. Nodo anterior al nodo actual que se encuentra en el mismo nivel que él (hermano anterior).
  - followin-sibling. Siguiente hermano del nodo actual.
  - descendet. Todos los nodos por debajo en la jerarquía del documento.
  - child. Todos los nodos inmediatamente por debajo en el árbol.
- Nodo evaluado. El resultado de un paso de localización es un nuevo punto de referencia que sirve como punto de partida para el siguiente paso de localización.

La sintaxis de los pasos de localización se completa con una serie de elementos adicionales entre los que destacan:

- Metacaracteres. "." indica el contexto actual; "*" hace referencia a todos los elementos descendientes inmediatos del nodo evaluado; "//", también llamado colapso de jerarquía, se utiliza en algunos casos en vez de "/" y permite establecer relaciones padre – cualquier descendiente sin importar la profundidad de localización dentro de la jerarquía.
- "::". Este símbolo separa el nombre de un eje de la expresión que se le desea aplicar a dicho eje.

- Filtros XPointer[6]. Se incorporan al paso de localización con el fin de filtrar la lista de nodos recuperados. Se incluyen entre corchetes. Existen varios tipos de filtros:
  - Indexados. Hacen referencia a un determinado nodo en función de la posición que éste ocupa en el total de nodos recuperados. Existen varias formas de hacerlo entre las que están [position() = n], [n] ó [last()][7].
  - Por contenido o presencia de un elemento. Condiciona la recuperación de un nodo al hecho de que contenga un determinado elemento e incluso a que contenga un determinado elemento con un contenido dado. Son formas habituales, *.../elemento[otro_elemento]/..., .../elemento[otro_elemento="contenido"]/...*
  - Por atributo o valor de atributo. Condiciona la recuperación de un elemento a que contenga un atributo con un cierto valor. Muy similar al anterior tipo pero referente a atributos. Ejemplos habituales son *.../elemento[@atributo]/..., .../elemento[@atributo="valor"]/...*

Los filtros pueden complicarse. Es posible usar expresiones lógicas mediante operadores lógicos y de comparación. Además, es posible crear filtros más complejos mediante la concatenación de los filtros. De esta forma, el conjunto de nodos recuperados por un filtro, son pasados al siguiente y así sucesivamente.

8.2. XSL

Como ya se ha comentado anteriormente, una de las características principales de XML es la separación entre contenido y presentación de dicho contenido. Va a ser XSL el encargado de incorporar dicha funcionalidad para poder procesar un documento XML y transformarlo a un formato más apropiado para la visualización.

Aunque XML incorpora esta característica, existen otras tecnologías con un objetivo similar. Un ejemplo son las hojas de estilo en cascada (CSS) [Morrison et al., 2000] para dar formato a los documentos HTML.

XSL está constituido por dos partes, el lenguaje denominado XSLT (*XSL Transform*) y el lenguaje denominado XSLFO (*XSL Formating Objects*). Se puede concluir que, XSLT es la parte fundamental de XSL al ser la parte que transforma el documento XML en una estructura de árbol en la cual se incluye un determinado formato mediante XSLFO. Tanto XSLT como XSLFO tiene sintaxis XML.

Pero XSLFO no es la única forma de establecer la presentación de cada nodo en el documento, y XSLT se usa junto con HTML y CSS[8], con VRML (*Virtual Reality Markup Language*), WML (*Wireless Markup Language*)...

---

[6] Aunque XPointer se analiza en el apartador referente a Xlink, se van a introducir aquellos conceptos relacionados con XPath.

[7] Éste hace referencia al último de los nodos recuperados.

[8] Esto viene motivado por la novedad de XSLFO y por el intento de mantener una compatibilidad con los navegadores actuales, los cuales no soportan XSLFO y sí HTML y CSS.

8.2.1 XSLT

XSLT toma una representación en forma de árbol del documento origen y la transforma en otro árbol donde cada nodo tiene asociado un conjunto de opciones de visualización (etiquetas).

XSLT está pensado para trabajar de forma independiente a XSL. Sin embargo, XSLT no está pensado como un lenguaje de propósito general dentro de la tecnología XML sino como parte de XSL. Por esta razón, se ha introducido como subapartado de XSL.

Las transformaciones XSLT se describen en un fichero diferente del fichero XML fuente que se quiere procesar. A través de la directiva *xml-stylesheet* se establece la relación entre el fichero XML y su correspondiente hoja de estilo[9]. Aunque esta es la forma habitual de trabajar, también es posible incluir dentro del mismo documento XML el código XSLT que permite llevar a cabo la transformación. Esto es lo que se conoce como XSLT empotrado.

Para XSLT todos los elementos que pueden aparecer en un documento XML son nodos. Así, se pueden distinguir los siguientes tipos de nodos [Navarro et al., 2000]:

- Nodo raíz. Se denomina, por lo general, instancia de documento y primer hijo es el elemento raíz del documento.
- Nodo de elemento. Cada uno de los elementos tienen asociado un nodo de este tipo. El orden de los nodos en el árbol XSL coincide con el orden de aparición en el documento origen.
- Nodos de texto. Nodo asociado a un nodo de elemento que contiene el texto de un determinado elemento.
- Nodo de atributo. Cada atributo de un elemento, se transforma en un nodo que en la jerarquía se encuentra por debajo del nodo del elemento al que pertenece el atributo.
- Nodos de espacio de nombres. Cada elemento tiene un nodo asociado al prefijo del espacio de nombres y otro asociado al espacio de nombre por defecto.
- Nodo de instrucción. No aparecen en el árbol procesado.
- Nodo de comentario. Contiene el valor del comentario. Tampoco se incluye en el árbol.

XSLT va a permitir definir plantillas, a través del elemento *xsl:template*, que se aplicarán sobre un conjunto de elementos del documento, elementos determinados a través de expresiones XPath. Estas expresiones se conocen como **patrones**. Las plantillas van a permitir asociar una entrada con una salida y controla la forma en la que los elementos son referenciados y transformados. El contenido de la plantilla es lo que realmente incorpora contenido al árbol resultado.

Cuando una plantilla se instancia se hace siempre con respecto a un **nodo actual** y a una **lista de nodos actuales**, cuyo primer elemento es el nodo actual. XSLT proporciona mecanismos para procesar cada uno de los nodos de dicha lista, permitiendo navegar de un nodo a otro, seleccionar un determinado nodo...

La dinámica de trabajo se basa en ir instanciando todas las plantillas para cada uno de los nodos de la lista de nodos actuales, o sólo para aquellos nodos que se decida. Para ello XSLT cuenta con una serie de instrucciones (*xsl:apply-templates, xsl:for-each...*) que permiten procesar los distintos nodos de una lista de nodos. La forma de seleccionar un determinado nodo dentro de una lista de nodos se basa, también, en el uso de expresiones XPath.

---

[9] Es equivalente al mecanismo de importar hojas de estilo, CSS, en HTML a través de la etiqueta <*style*>.

Cuando existe más de una plantilla que puede instanciarse, se resuelve el conflicto mediante el uso de prioridades. Estas prioridades se pueden especificar de forma explícita, y en caso de no detallarse, se asocia a cada plantilla una prioridad por defecto[10].

De esta forma, durante el proceso de transformación del árbol origen se van instanciando diferentes plantillas, donde cada una de las cuales procesa un conjunto de nodos introduciendo contenido al árbol resultado.

En este punto, cabe destacar una diferencia importante entre la expresión XPath usada para determinar los nodos a los que aplicar la plantilla, es decir los patrones, y la expresión para aplicar transformaciones a dichos nodos. La primera sólo genera listas de nodos, mientras que la segunda puede generar, además, los siguientes tipos de datos: cadenas, números, valores lógicos... [Navarro et al., 2000].

Existen diversas formas de introducir nodos en el árbol resultado. Una primera forma es el uso de **instrucciones** propias de **XSLT**. Así, existen instrucciones que permiten extraer el contenido de datos de un determinado elemento del documento origen (*xsl:value-of*), instrucciones que permiten copiar elementos del árbol origen al árbol destino (*xsl:copy, xsl:copy-of...*), instrucciones que permiten crear nuevos elementos (*xsl:element*) y atributos para dichos elementos (*xsl:attribute, xsl:attribute-set*), instrucciones que permiten crear comentarios (*xsl:comment*), texto (*xsl:text*), instrucciones de procesamiento (*xsl:processing-instruction*)...

Otra forma es incluir de forma explícita en las plantillas etiquetas que formarán parte del árbol resultado. XSLT incluirá en el árbol destino todo elemento que aparezca dentro de una plantilla (etiquetas HTML, datos VRML...), siempre que no se trate de una instrucción que conozca y pueda ejecutar.

La sintaxis de XSLT es mucho más amplia. Permite procesamiento condicional de nodos (*xsl:if, xsl:choice...*), el uso de variables, la ordenación de resultados (*xsl:sort...*), incluir plantillas procedentes de otros documentos XSLT externos, formatos explícitos de salida (*xsl:ouput*)... [Navarro et al., 2000] [Morrison et al., 2000].

8.2.2 XSLFO

La segunda parte de XSL está constituida por XSLFO que se encarga de dar formato realmente a los contenidos obtenidos con XSLT.

Es una tecnología nueva y poco usada por lo que, normalmente, se sustituye por las etiquetas de HTML y CSS, como mecanismos de representación. Aún así, la potencia de XSLFO es considerable a la hora de dar formato visual a los nodos.

Los objetos de formato incluyen un gran número de construcciones sintácticas para definir una amplísima gama de posibilidades de visualizar contenidos con diferentes aspectos. Algunas de estas construcciones permiten definir el tipo de área donde se mostrarán los datos, las propiedades de las líneas, fuentes, gráficos... Incluso los objetos de formato se pueden utilizar para generar contenido binario a partir de un documento XML. En este sentido el procesador de objetos de formato **FOP** del **Apache XML Group** (http://xml.apache.org) permite transformar un documento XML en un documento PDF incluyendo en la hoja de estilo XSL los correspondientes objetos de formato.

[10] Esta prioridad es más alta cuanto más específica sea la plantilla. Para más información sobre el tema de las prioridades acudir a http://www.w3c.org/TR/1999/REC-xslt-19991116.

9. Enlazado de documentos. XML Link

Este lenguaje va a permitir enlazar distintos documentos entre sí, e incluso enlazar un documento con partes dentro de otro documento[11]. Para este segundo aspecto, se cuenta con un lenguaje específico, denominado XPointer, que se analiza más adelante.

El concepto principal dentro de XLL (*XML Link*) es el de **enlace** (*link*). Por enlace se entiende la relación existente entre recursos o porciones de recursos (nótese que se habla en plural). Esta relación se hace explícita a través de un elemento de enlace. Un recurso es cualquier unidad de información o servicio accesible. Concretamente, un recurso es un documento XML, aunque los enlaces en XLL no se limitan a relacionar documentos XML.

Una característica de XLL es que no existe un elemento explícito que defina un enlace como sucede en HTML sino que cualquier elemento puede funcionar de elemento de enlace sin más que incluir el espacio de nombres para los enlaces a través del atributo *xmlns:xlink*[12].

Además del concepto de enlace, cabe destacar el concepto de **arco** (*arc*). Un arco establece la manera en la que se relacionan dos recursos a través de un enlace. Define la forma de uso de un enlace, el comportamiento de dicho enlace, los recursos origen y destino del enlace y otro tipo de información adicional. Existen tres tipos de arcos desde el punto de vista de los recursos a los que referencian:

- **outbound**. En este tipo de arcos, el elemento origen del enlace es un elemento local y el recurso destino es un elemento remoto (identificado por un URI). Este es el comportamiento de un enlace tradicional (entendiendo por tradicional el comportamiento de los enlaces en HTML).
- **inbound.** En este tipo de arcos, el elemento origen es remoto, mientras que el recurso destino es local.
- **third-party.** En este tipo de arcos, tanto origen como destino hacen referencia a recursos remotos.

A la vista de los dos últimos tipos de arco, se puede establecer un enlace entre dos recursos externos al documento en el cual se describe, físicamente, en enlace.

Dentro de XLL se definen dos tipos de enlaces, los extendidos y los simples.

9.1. Enlaces extendidos

Los enlaces **extendidos**, también llamados multidocumento, se denominan así porque no hacen referencia a un único recurso destino sino a un número arbitrario de documentos. La funcionalidad de estos enlaces viene dada por arcos *inbound* y *third-party*.

Normalmente, este tipo de enlaces se almacenan de forma separada de los recursos que asocian. De esta forma, los enlaces extendidos son útiles cuando los recursos accedidos son recursos de sólo lectura, difíciles de modificar y actualizar, o recursos que no soportan enlaces en sí mismos (por ejemplo un fichero de imagen).

---

[11] XLink, a fecha de redacción de este documento, está aún en fase de recomendación. Debido a esto, es posible que alguno de los aspectos que aquí se detallan cambien, e incluso, que la bibliografía consultada no sea uniforme a la hora de especificar conceptos e incluso sintaxis. Como se ha realizado a lo largo de todo el documento, se seguirá la recomendación del W3C.

[12] El *namespace* para los enlaces es http://www.w3c.org/1999/xlink

Si para un enlace de este tipo se definen dos arcos que relacionan los mismos recursos pero de forma que el origen de un arco es el destino del otro y viceversa, se obtiene un enlace multidireccional. El concepto de multidireccionalidad no es el mismo que el de "volver hacia atrás" después de haber usado un enlace.

Desde el punto de vista de implementación, un enlace extendido se define a través del atributo *xlink:type = "extended"*.

La forma de establecer los recursos que relacionan, así como el comportamiento del enlace e información adicional se realiza a través de los elementos del tipo *locator*, *arc*, *title* o *resource,* elementos cuyo atributo *xlink:type* toma los valores de *locator*, *arc*, *title* o *resource*, respectivamente.

Los elementos del tipo *locator* permiten establecer un nexo con recursos remotos dentro de un enlace. Además, incorpora atributos adicionales que permiten incluir información acerca del propio nexo (atributos *xlink:role* y *xlink:title*). El recurso se identifica a través del atributo *xlink:href*. Un atributo adicional, *xlink:label*, permite identificar este elemento.

Los elementos del tipo *resource* proporcionan la referencia a los recursos locales que participan en el enlace. Este elemento posee los mismos atributos que el elemento anterior.

Los elementos del tipo *arc* establecen las reglas que dictan cómo se relacionan los recursos referenciados por los elementos *locator* y *resource*. Un elemento tipo *arc* puede hacer referencia a objetos *locator* o *resource* a través del atributo *xlink:label* de estos últimos. El elemento del tipo arc establece el punto origen y destino del enlace (*xlink:from*, *xlink:to*), cómo se mostrará el recurso remoto (*xlink:show*) y cuándo se desencadenará el enlace (*xlink:actuate*), bien por interacción del usuario o de forma automática... También incluye atributos adicionales para incorporar información adicional (*xlink:arcrole* y *xlink:title*).

El elemento adicional tipo *title* permite incluir más información acerca del enlace. Este elemento complementa a los atributos propios de cada uno de los elementos analizados para incluir información.

9.1.1 Linkbases

Para que una aplicación pueda navegar a través de un enlace necesita localizar tanto el punto de partida como el enlace. Esto no supone ningún problema para el caso de los enlaces del tipo *outbound* puesto que el recurso origen es el propio elemento de enlace o un hijo de este elemento. El problema surge cuando se usa un enlace de cualquiera de los otros tipos.

Para solucionar este problema se establece lo que se conoce como *linkbases*. Se trata de un recurso externo en formato XML que contiene los enlaces. La forma de establecer su ubicación es a través de elementos tipo *arc*, caracterizados porque su atributo *xlink:arcrole* tiene el valor http://www.w3c.org/1999/xlink/properties/linkbase. En este caso, mediante un elemento del tipo *locator* se establece la ubicación del documento *linkbase*.

9.2. Enlaces simples

Los **enlaces simples** conectan un documento origen con un documento destino completo. No usan XPointer y, por lo tanto, no pueden acceder de forma directa a ningún recurso dentro del documento al que referencian.

Un enlace simple tiene implícito un arco del tipo *outbound*.

Se especifica a través del atributo *xlink:type = "simple"*.

Al implementar de forma implícita un arco es posible adjuntar información adicional, ya que con atributos *xlink:arcrole*, *xlink:role* y *xlink:title*, además de poder incluir elementos del tipo *title*.

Otra característica es la posibilidad de establecer cómo se mostrarán los recursos identificados, reemplazando el documento desde el que se referencian, en una nueva ventana o incluidos dentro del propio documento origen.

También, puede definirse si es el usuario el que los activa o se activan de forma automática.

9.3. XPointer

La segunda parte de XLL está formada por este lenguaje, que incluye los mecanismos necesarios para referenciar partes dentro de un documento.

Su uso principal se da dentro de XLL aunque, como ya se ha visto, también se usa con XPath. Pero esta relación va más allá, ya que XPointer usa, para acceder a los recursos de un documento, expresiones XPath.

Los mecanismos con los que cuenta para hacer referencia a los recursos son referencias por número, nombre, tipo o relación con otros elementos del documento. El resultado puede ser un elemento o conjunto de elementos.

El mecanismo es el siguiente. Mediante un enlace se hace referencia a un documento, en el cual se desea acceder a una determinada parte. Una vez identificado el documento, se usa un camino de localización (XPath) para acceder a un conjunto de recursos y filtrarlos con XPointer, o bien, a través de mecanismos que no necesitan XPath, acceder al recurso concreto que se necesita.

XPointer, además, incorpora ciertas características a destacar. La primera de ellas, es la recuperación de rangos dentro de un documento. Se establece un punto inicial y otro punto final para identificar todo el rango a recuperar.

Otra es la posibilidad de incorporar alternativas de búsqueda sin más que concatenar apuntadores XPointer. De esta forma, si no se encuentra el recurso con el primer apuntador, se usa el segundo como alternativa.

Es posible determinar si se carga todo el documento y luego se accede al recurso local, o si se recupera del documento referenciado únicamente el recurso accedido por XPointer [Navarro et al., 2000].

10. Procesamiento de documentos XML

Hasta este momento se han analizado los fundamentos de XML. Se ha visto como escribir documentos en XML, como procesarlos con XSL para transformar el documento con vistas a su visualización (entre otras finalidades), como enlazar documentos... El siguiente paso es introducir cómo llevar a cabo la programación sobre XML, es decir el procesamiento de la información que se presenta en formato XML.

Para realizar dicho procesamiento se cuenta con dos API (*Application Programming Interfaces*) principales: SAX y DOM. En los siguientes apartados se analiza cada una de estas APIs.

10.1. SAX

SAX (*Standard API for XML*) (http://www.saxproject.org/) [Morrison, 2000], [Navarro et al., 2000] se encarga de procesar el documento carácter a carácter con el objetivo de identificar fragmentos de información útiles que son enviados a la aplicación que hace uso de dicha API. Se pretende analizar el documento XML identificando cada uno de los elementos sensibles dentro del documento: etiquetas de apertura, etiquetas de cierre, datos...

En efecto, la tarea que desarrolla SAX es la de un *parser*. Procesa todo el documento de forma secuencial proporcionando a la aplicación cada una de las partes de la que consta.

Existe un aspecto fundamental en el procesado de un documento XML que es el tratamiento de los espacios en blanco. El problema es cuándo decidir ignorarlos y cuándo no (un carácter en blanco dentro de los datos posiblemente no se deberá ignorar). Para ello, SAX tiene en cuenta la estructura del documento, especificada a través de una DTD o un esquema a la hora de considerar este carácter como elemento constitutivo del fragmento de información.

La principal ventaja de esta API es el reducido uso de memoria que conlleva y la rapidez de procesamiento del documento XML. Debido a estas razones, se utiliza en el lado del servidor dentro de una arquitectura cliente / servidor. El ahorro de memoria se consigue gracias a que SAX, únicamente, tiene en cuenta en cada momento el punto actual del documento sin considerar el resto. En ningún momento, tiene conciencia de todo el documento en sí.

Por otro lado, la forma de trabajar de SAX trae ciertas limitaciones que es necesario comentar. La primera de ellas es la posibilidad única de procesar el documento siempre hacia delante. Al funcionar como un *parser*, el sentido de trabajo siempre es secuencial, con lo que en ningún momento se puede volver hacia atrás en el documento.

Otra limitación importante, es la imposibilidad de introducir nuevo contenido en el documento. Esto únicamente se podría llevar a cabo si se mantuviera en memoria una instancia del documento completo, y como ya se ha comentado, SAX no maneja, en ningún momento, el documento completo.

10.2. DOM

Mientras la API SAX no tiene conciencia del documento completo en un instante de tiempo, el API DOM (*Document Object Model*) (http://www.w3.org/DOM) [Morrison, 2000], [Navarro et al., 2000] mantiene todo el documento en memoria.

El objetivo del DOM es permitir trabajar sobre el documento XML de forma más flexible. Se permite navegar por el documento en cualquier sentido, se permiten añadir nuevos contenidos al documento...

En general, el DOM procesa el documento XML y lo transforma en una estructura de árbol que mantiene y gestiona en memoria. Esto trae consigo un coste elevado de consumo de recursos. Una vez el árbol está construido, el programa, a través de DOM, puede operar con el documento.

Para permitir este procesamiento, el DOM tiene constancia, en todo momento, del punto actual donde se centra el procesamiento. El desplazamiento dentro del árbol se lleva a cabo invocando a funciones de esta API. Es frecuente que dichas funciones tomen como punto de partida el nodo actual. Lo mismo sucede con operaciones de modificación del árbol, tales como añadir nodos, eliminar nodos, mover un nodo de una posición del árbol a otra, cambiar el contenido del nodo...

A fecha de redacción de este documento, el API DOM no está estandarizado. El tratamiento que se haga del documento XML depende en gran medida de la librería que se esté

utilizando. Por ejemplo, la inclusión de los comentarios dentro del árbol es uno de los aspectos que puede variar entre las distintas implementaciones.

10.3. JAXP

Otra API que conviene referenciar es JAXP (*Java API for XML Processing*) (http://java.sun.com/xml/jaxp/index.html), API de Java para procesar XML. Esta API, desarrollada por **Sun Microsystems** dentro de su denominado Proyecto X, intenta dar cohesión tanto a SAX como a DOM facilitando el trabajo a los desarrolladores de software en Java. La palabra cohesión fue uno de los aspectos sobre los que más se ha incidido dentro de esta API, dado que no pretende sustituir ninguna de las APIs ya existentes, sino únicamente ofrecer un conjunto de interfaces, de forma gratuita como casi todos los elementos de la distribución de Java, para hacer más sencillo el manejo a aquéllos que, teniendo soltura con Java, deciden incorporar XML dentro de sus proyectos.

11. Usos de XML

XML y todas las tecnologías relacionadas con él han tomado posición firme dentro de la comunidad científica hasta el punto de ser obligado referente para múltiples propósitos y cometidos.

El primer uso es consecuencia directa de una de las propiedades más explotadas de XML: la posibilidad de separar los contenidos a mostrar al cliente de la presentación que finalmente tomarán dichos contenidos. Los datos se presentan en formato XML mientras que XSL y XSLT se encargan de dar forma a los contenidos. Hoy en día existen una gran variedad de dispositivos potenciales en los que se puede mostrar contenidos (navegadores, teléfonos móviles, ordenadores de bolsillo...) y cada uno de ellos soporta un conjunto de primitivas propias para formatear los datos. Gracias a XML se evita tener que crear tantas versiones como dispositivos destino se desee cubrir.

Otro uso cada vez más extendido es la transferencia de información con formato XML. Ya se ha visto la flexibilidad de XML para definir datos y la estructura de los mismos. Además, no se puede olvidar que los ficheros XML son documentos de texto que se transmiten fácilmente a través de la red. Esta idea inicial se ha extendido y han surgido especificaciones que explotan esta propiedad de XML. Una de estas especificaciones es XML – RPC (*XML for Remote Procedure Call*) que utiliza XML para llevar a cabo llamadas a procedimientos remotos, bien entre componentes distribuidos de una misma aplicación, o entre aplicaciones que comparten un conjunto de servicios[13]. Utilizar XML en este tipo de entornos mejora los rendimientos. El problema de las RPC estaba en la transmisión de objetos complejos. Intentar transformar dichos objetos en una especificación textual era demasiado complejo. Sin embargo la potencia de XML hace posible esta transformación. De esta forma el motor XML – RPC puede hacer corresponder los parámetros de instancia de un objeto con los elementos XML, pudiendo generar fácilmente documentos XML y reconstruir los objetos a partir de dichos documentos.

Sin entrar en temas de especificaciones de transmisión de datos, XML se utiliza en ámbitos de comunicación tan importes como las relaciones B2B (*Business – to – Business*) y B2C

[13] Para saber más sobre XML – RPC acudir a www.xml-rpc.com.

(*Business – to –Consumer*) mejorando y sustituyendo, en muchos casos, los sistemas EDI (*Electronic Data Interchange*) tradicionales[14].

Aunque pueda parecer de menor trascendencia, XML, también, se ha impuesto como formato estándar de definición de ficheros de configuración en múltiples sistemas. La mayor parte de las nuevas aplicaciones ya incluyen este tipo de ficheros para configurar sus entornos. Algunos ejemplos son el motor de *servlets* Tomcat de desarrollo conjunto entre Apache y Sun Microsystems, el propio servidor *web* Apache...

12. XML y el comercio electrónico

En este apartado se pretende ofrecer una visión general de la aportación de XML al mundo del comercio electrónico.

En la actualidad se pueden describir dos marcos principales de trabajo con XML en el mundo del comercio electrónico: la definición de vocabularios y la creación de protocolos.

El primer grupo, los **vocabularios**, se centran en definir una sintaxis y una semántica que debe respetarse a la hora de crear un documento válido. Se han desarrollado numerosos vocabularios en distintas áreas como contabilidad (*Extensible Financial Reporting Markup Language*) (http://www.oasis-open.org/cover/xfrml.html), banca (*Bank Internet Payment System*) (http://www.fstc.org/projects/bips/), servicios de directorio (*Directory Service Markup Language*) (http://www.dsml.org/about.html), recursos humanos (*Human Resources Markup Language*) (http://www.hrml.com/)...

La creación de **protocolos** para intercambiar datos basados en XML se ve potenciada debido a que XML no establece ninguna restricción a la hora de intercambiar información entre aplicaciones. Además de **XML-RPC**, ya comentado, existen otros protocolos a tener en cuenta como son **XML-CORBA** (http://www.omg.org/technology/xml/index.htm) y **SOAP** (*Simpe Object Access Protocol*) (http://www.develop.com/soap/)

Centrándose en aplicaciones concretas, XML se aplica a multitud de áreas dentro del comercio electrónico. Algunos de estos ámbitos son la publicación web, el intercambio de datos y sistemas distribuidos en red, donde XML compite con otras tecnologías como **ASN.1** (*Abstract Syntax Notation 1*) (http://www.iso.org) o **EDI**[15] que trabajan sobre soporte binario, en cadenas de distribución[16] (*supply-chain integration*) donde tradicionalmente sólo ha existido EDI, *marketing*, servicio al cliente... [Kamthan y Pai, 2000].

En cuanto a las propiedades que XML posee de cara al mundo del comercio electrónico cabe destacar las siguientes [Kamthan y Pai, 2000]:

- **Estándar**. XML es independiente de plataforma y es de libre distribución. Además, el hecho de ser un estándar permite garantizar que la representación y la transferencia de información. Esto es un aspecto crucial en las relaciones B2B y B2C.
- **Facilidad de manejo**. El hecho de que XML facilite la separación entre contenido y presentación permite solventar los problemas derivados de la diversidad de dispositivos

[14] Un artículo donde se puede ver como se combinan estas dos tecnologías se puede consultar en http://www.xml.org/xml/news_market.shtml.

[15] EDI también se está extendiendo hacia XML (http://www.xmledi.com)

[16] En este ámbito se ha desarrollado un vocabulario explícito conocido como **xCBL** (*Common Business Library*)

finales, así como las dificultades procedentes de la incompatibilidad de versiones. Esto permite centrar esfuerzos en la edición de contenidos y facilita el mantenimiento de la información.

- **Mayor tiempo de vida**. El tiempo de vida de los formatos binarios y propietarios coincide con los de los sistemas que los soportan. El hecho de que XML sea texto plano permite que su periodo de validez supere el de los sistemas que lo usan.
- **Mejora de relaciones B2B**. XML permite simplificar este tipo de relaciones gracias a las siguientes características:
  - Para intercambiar información en XML únicamente es necesario que las partes estén de acuerdo en el vocabulario XML a utilizar.
  - Ninguna de las partes necesita conocer la organización de los sistemas del interlocutor.
- **Interfaz hombre – máquina y máquina – máquina**. XML proporciona una forma de intercambiar información no sólo entre sistemas informáticos, sino también entre personas con el mismo formato.
- **Internacionalización**. Gracias a que XML soporta diferentes codificaciones permite crear documentos con las características de propias del idioma de cada región.

13. Bibliografía

[Bertrand, 2000] Bertrand, E. (2000). "*XML. Gestión de Datos en la Era de los Negocios Electrónicos: Situación Actual y Perspectiva*". Tutorial en las V Jornadas de Ingeniería del Software y Bases de Datos, JISBD2000 (Valladolid, 8-10 de noviembre de 2000).

[Biron y Malhotra, 2001] Biron P. V., Malhotra, A. (2001). "*XML Schema Part 2: Datatypes*". World Wide Web Consortium. Recommendation, 2 May 2001. http://www.w3.org/TR/xmlschema-2/.

[Bray et al., 2000] Bray, T., Paoli, J., Sperberg-MacQueen, C. M. (2000). "*Extensible Markup Language (XML) 1.0*". 2nd edition. World Wide Web Consortium Recommendation October 2000. http://www.w3c.org/TR/2000/REC-xml-20001006.

[Clark, 1999] Clark, J. (1999). "*XSL Transformation (XSLT) Version 1.0*". World Wide Web Recommendation 16 November 1999. http://www.w3c.org/TR/1999/REC-xslt-19991116.

[DeRose et al., 2000] DeRose, S., Maler, E., Orchard, D. (2000). "*XML Linking Language (XLink) Version 1.0*". World Wide Web Consortium Proposed Recommendation 20 December 2000. http://www.w3c.org/TR/2000/PR-xlink-20001220.

[Fallside, 2001] Fallside, D. C. (2001). "*XML Schema Part 0: Primer*". World Wide Web Consortium. Recommendation, 2 May 2001. http://www.w3.org/TR/xmlschema-0/.

[Kamthan y Pai, 2000] Kamthan P., Pai, H. (2000). "*Perspectives of XML in E-Commerce*". Diciembre 2000. http://tech.irt.org/articles/js215/

[Morrison, 2000] Morrison, M. (2000). "*XML Al descubierto*". Ed. Prentice Hall.

[Navarro et al, 2000] Navarro A., White, C., Burman, L. (2000). "*Mastering XML*". Ed.Sybex.

[Raggett, 1999] Raggett, D. (1999). "HTML 4.01 Specification". W3C Recommendation. 24 December 1999. http://www.w3c.org/TR/REC-html40.

[Thompson et al., 2001] Thompson, H. S., Beech, D., Maloney, M., Mendelsohn. N. (2001). "*XML Schema Part 1: Structures*". World Wide Web Consortium. Recommendation, 2 May 2001. http://www.w3.org/TR/xmlschema-1/.

[Wohler, 1994] Wohler, W. (1994) "*SGML Declarations*" Publishing Solutions Development IBM Corporation. http://www.oasis-open.org/cover/wlw11.html.