# CAPÍTULO 4: XML

En el proyecto que hemos realizado, **XML** (*eXtensible Markup Language*) es la sintaxis que subyace en la información que transmitimos. Dicho de otro modo, la información que nos intercambiamos en el desarrollo del proyecto sigue las normas y pautas que nos indica **XML**.

## 4.1.- Introducción

**XML** es una sintaxis universal para la descripción y el estructurado de datos independientemente de la lógica de una aplicación. Puede ser utilizado para definir un número ilimitado de lenguajes destinados a aplicaciones específicas. El significado de las tres letras que componen su nombre es *eXtensible Markup Language*, que significa *Lenguaje Extensible de Etiquetado*.

La versión 1.0 de XML es una recomendación del *W3C* (*World Wide Web Consortium*) publicada en Febrero de 1998, aunque se llevaba trabajando en ella desde dos años antes. Está basada en el anterior estándar **SGML** (*Standard Generalized Markup Language*, *ISO 8879*), que data de 1986, aunque empezó a gestarse a principios de los 70. Éste está a su vez basado en **GML**, creado por IBM en 1969.

Aunque pueda parecer moderno, sus conceptos están ampliamente asentados y aceptados. Se encuentra además asociado a la recomendación del *W3C* **DOM** (*Document Object Model*), aprobado también en 1998. Éste no es más que un modelo de objetos que, en forma de API, permite acceder a las diferentes partes que pueden componer un documento XML o HTML.

**SGML** proporciona un modo consistente y preciso de aplicar etiquetas para describir las partes que componen un documento, permitiendo además el intercambio de documentos entre diferentes plataformas. Sin embargo, el problema que se le atribuye es su excesiva dificultad; un indicativo de ella es el hecho de que su recomendación ocupe unas 400 páginas.

De esta forma, manteniendo su misma filosofía, de él se derivó **XML** como subconjunto simplificado, eliminando las partes más engorrosas y menos útiles. Al igual que sus padres, **XML** es un metalenguaje: un lenguaje para definir lenguajes. Los elementos que lo componen pueden dar información sobre lo que contienen, no necesariamente sobre su estructura física o presentación, como ocurre en **HTML**.

Al igual que **XML**, **HTML** se también se definió utilizando **SGML**. En una primera aproximación, las diferencias entre ambos residen en que **HTML** es simplemente un lenguaje, mientras que **XML** un metalenguaje; es decir, se trata de un lenguaje para definir lenguajes. De hecho, mediante **XML** también podríamos definir el **HTML**.

Desde su creación, ha tenido un crecimiento exponencial. La cantidad de artículos escritos sobre él y el número de aplicaciones compatibles con sus especificaciones se multiplicaron al poco tiempo de aparición. Su importancia queda patente atendiendo al elevado número de importantes compañías que han optado por este nuevo estándar.

**XML** 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 prácticamente cualquier aplicación concebible. Sin ir más lejos, algunos lenguajes, definidos en **XML**, recorren áreas como la química y la física, las matemáticas, el dibujo, tratamiento del habla, y otras muchas más.

Aplicado en Internet, establece un estándar fijo al que atenerse, y separa el contenido de su presentación. Esto significa que la visualización de documentos Web puede no estar sujeta al estándar de hojas de estilo (**CSS**) que soporte el navegador ni al lenguaje de *script* del servidor. Además, con él también tampoco habrá de estar atada a la plataforma, pudiendo apreciarse la misma información en dispositivos tan dispares como un PC o un PDA.

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.

## 4.2.- Conceptos básicos

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 él 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.

Un documento XML tiene el siguiente aspecto:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ficha>
        <nombre>Alberto</nombre>
        <apellidos>Oliva Mateos</apellidos>
        <direccion>c/Playa de Matalascañas, 8</direccion>
</ficha>
```

Figura 4. 1: Ejemplo de documento XML

Aunque es opcional, prácticamente todos los documentos **XML** comienzan con una línea como la primera. Es la encargada de indicar que lo que la sigue es un documento **XML**. Puede tener varios atributos (los campos que van dentro de la declaración), algunos obligatorios y otros no:

- ***version***: indica la versión de **XML** usada en el documento; la actual es la 1.0. Es obligatorio ponerlo, a no ser que sea un documento externo a otro que ya lo incluía.

- ***encoding***: señala la forma en que se ha codificado el documento. Cualquier valor está permitido, y depende del parser el entender o no la codificación. Por defecto es *UTF-8*, aunque existen otras, como *UTF-16*, *US-ASCII*, *ISO-8859-1*, etc. No es obligatorio salvo que se trate de un documento externo a otro principal.

- ***standalone***: indica si el documento va acompañado de un **DTD** (no), o no lo necesita (yes). Su presencia no es necesaria, ya que más tarde se indicará el **DTD** en caso de ser necesario.

En cuanto a la sintaxis del documento, y antes de entrar en el estudio de las etiquetas, hay que resaltar algunos detalles de gran importancia:

- Los documentos **XML** son sensibles a mayúsculas. Es decir, en ellos se diferencia las mayúsculas de las minúsculas. Por ello <FICHA> sería una etiqueta diferente a <ficha>.

- Además todos los espacios y retornos de carro se tienen en cuenta (dentro de las etiquetas, en los elementos).

- Hay algunos caracteres especiales reservados, que forman parte de la sintaxis de **XML**: <, >, &, ″ y ′. En su lugar, cuando se desee representarlos habrá que usar las entidades *<*, *>*, *&*, *"* y *'* respectivamente.

- Los valores de los atributos de todas las etiquetas deben ir siempre entrecomillados. Son válidas las dobles comillas (") y la comilla simple (').

Analizando el contenido, se observan una serie de etiquetas que contienen datos. Cabe señalar la diferencia existente entre los conceptos de elemento y etiqueta: los elementos son las entidades en sí, lo que tiene contenido, mientras que las etiquetas sólo describen a los elementos. Un documento **XML** está compuesto por elementos, y en su sintaxis éstos se nombran mediante etiquetas.

Existen dos tipos de elementos: los vacíos y los no vacíos. Hay varias consideraciones importantes a tener en cuenta al respecto:

- Toda etiqueta no vacía debe tener una etiqueta de cerrado: <etiqueta> debe estar seguida de </etiqueta>. Esto se hace para evitar posibles errores de interpretación.

- Todos los elementos deben estar perfectamente anidados. No está permitido lo que se expone en el siguiente ejemplo:

```
<ficha><nombre>Alberto</ficha></nombre>
```

Figura 4. 2: Ejemplo de elementos mal anidados

- Los elementos vacíos son aquellos que no tienen contenido dentro del documento. Un ejemplo en **HTML** son las imágenes. La sintaxis correcta para estos elementos implica que la etiqueta tenga siempre esta forma: <etiqueta/>.

Hasta aquí llega esta reducida introducción a la sintaxis **XML**. Aunque la especificación íntegra es más extensa en cuanto a detalles sintácticos, codificaciones, etc., con lo expuesto hasta ahora es más que suficiente para el alcance del presente proyecto.

## 4.3.- DTD: Definición de Tipos de Documentos

Tal y como se comentó con anterioridad, los documentos **XML** puede ser válidos o bien formados. Con respecto a los primeros, también se mencionó que su gramática está definida en los **DTD**.

Los **DTD** no son más que definiciones de los elementos que puede incluir un documento **XML**, de la forma en que deben hacerlo (qué elementos van dentro de otros) y los atributos que se les puede dar. Normalmente la gramática de un lenguaje se define mediante notación **EBNF**, que es bastante engorrosa. **DTD** hace lo mismo pero de un modo más intuitivo.

Hay varios modos de hacer referencia a un **DTD** en un documento **XML**:

- Incluir dentro del documento una referencia al documento **DTD** en forma de **URI** (*Universal Resource Identifier*, o *Identificador Universal de Recursos*) y mediante la siguiente sintaxis:

```
<!DOCTYPE ficha SYSTEM “http://www.url_prueba/~aoliva/DTD/ficha.dtd”>
```

Figura 4. 3: Referencia a DTD dentro del documento

En este caso, la palabra *SYSTEM* indica que el **DTD** se obtendrá a partir de un elemento externo al documento e indicado por el **URI** que lo sigue.

- Incluir el **DTD** dentro del propio documento:

```
<?xml version=”1.0”?>
<!DOCTYPE ficha [
<!ELEMENT ficha (nombre+, apellido+, direccion+, foto?)>
<!ELEMENT nombre (#PCDATA)>
<!ATTLIST nombre sexo (masculino | femenino) #IMPLIED>
<!ELEMENT apellidos (#PCDATA)>
<!ELEMENT direccion (#PCDATA)>
<!ELEMENT foto EMPTY>
]>
<ficha>
        <nombre>Alberto</nombre>
        <apellidos>Oliva Mateos</apellidos>
        <direccion>c/Playa Matalascañas, 8</direccion>
</ficha>
```

Figura 4. 4: DTD dentro de un documento

La forma de incluir el **DTD** directamente como en este ejemplo pasa por añadir a la declaración <!DOCTYPE y después el nombre del tipo de documento, en vez de la **URI** del **DTD**, el propio **DTD** entre los símbolos '*[*' y '*]*'. Todo lo que hay entre ellos será considerado parte del **DTD**.

En cuanto a la definición de los elementos, tiene la virtud de ser bastante intuitiva: tras la cláusula <!ELEMENT se incluye el nombre del elemento (el que luego se indicara en la etiqueta), y después, en función del elemento:

- Entre paréntesis, si el elemento no está vacío, se indica el contenido que puede tener el elemento: la lista de elementos hijos o que descienden de él si los tiene, separados por comas; o el tipo de contenido, normalmente *#PCDATA*, que indica datos de tipo texto, que son los más habituales.

- Si es un elemento vacío, se indica con la palabra *EMPTY*.

A la hora de indicar los elementos descendientes (los que están entre paréntesis) vemos que van seguidos de unos caracteres especiales: '+', '*', '?' y '|'. Sirven para indicar qué tipo de uso se permite hacer de esos elementos dentro del documento:

- +: uso obligatorio y múltiple; permite uno o más elementos de ese tipo dentro del elemento padre, pero como mínimo uno.

- *: opcional y múltiple; puede no haber ninguna ocurrencia, una o varias.

- ?: opcional pero singular; puede no haber ninguno o como mucho uno.

- |: equivale a un OR, es decir, da la opción de usar un elemento de entre los que forman la expresión, y solo uno.

De este modo, si por ejemplo encontramos en un **DTD** la declaración *<!ELEMENT ficha (nombre+, apellidos+, direccion*, foto?, telefono*|fax*)>*, sabremos del elemento ficha que puede contener los siguientes elementos: un nombre y unos apellidos como mínimo, pero puede tener más de uno de cada; opcionalmente puede incluirse una o varias direcciones, pero no es obligatorio; opcionalmente también se puede incluir una única foto; y por fin, pueden incluirse, aunque no es obligatorio en ninguno de los dos casos, uno o más teléfonos o uno o más números de fax.

Como ya se comentó, un documento **XML** presenta una jerarquía muy determinada, definida en el **DTD** si es un documento válido, pero siempre inherente al documento en cualquier caso (siempre se puede inferir esa estructura a partir del documento sin necesidad de tener un **DTD** en el que basarse), con lo que se puede representar como un árbol de elementos. Existe un elemento raíz, que siempre debe ser único (sea nuestro documento válido o sólo bien formado) y que se llamará como el nombre que se ponga en la definición del <!DOCTYPE si está asociado a un **DTD** o cualquiera que se desee en caso contrario. Y de él descienden las ramas de sus respectivos elementos descendientes o hijos. De este modo, la representación en forma de árbol de nuestro documento XML de ejemplo sería:

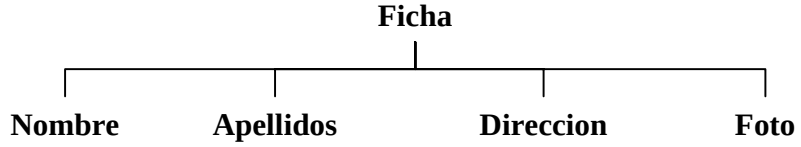


Figura 4. 5: Jerarquía del documento de ejemplo

Se observa que se trata de un documento muy sencillo, con una profundidad de dos niveles nada más: el elemento raíz ficha, y sus hijos nombre, apellido, dirección, y foto. Es

obvio que cuanta más profundidad, mayor tiempo se tarda en procesar el árbol, pero la dificultad siempre será la misma gracias a que se usan como en todas las estructuras de árbol algoritmos recursivos para tratar los elementos.

El **DTD**, por ser precisamente la definición de esa jerarquía, describe la forma del árbol. La diferencia (y la clave) está en que el **DTD** define la forma del árbol de elementos, y un documento **XML** válido puede basarse en ella para estructurarse, aunque no tienen que tener en él todos los elementos, si el **DTD** no obliga a ello. Un documento **XML** bien formado sólo habrá de tener una estructura jerarquizada, pero sin tener que ajustarse a ningún **DTD** concreto.

Para la **definición de los atributos**, se usa la declaración <!ATTLIST seguida de:

- El nombre de elemento del que se está declarando los atributos.
- El nombre del atributo.
- Los posibles valores del atributo, entre paréntesis y separados por el carácter |. Al igual que para los elementos, significa que el atributo puede tener uno y sólo uno de los valores incluidos entre paréntesis. O bien, si no hay valores definidos, se escribe *CDATA* para indicar que puede ser cualquier valor alfanumérico. También podemos indicar con la declaración *ID* que el valor alfanumérico que se le de será único en el documento, y se podrá hacer referencia a ese elemento a través del atributo y valor.
- De forma opcional y entrecomillado, un valor por defecto del atributo si no se incluye otro en la declaración.
- Por último, si es obligatorio declarar este atributo cada vez que se usa el elemento en cuestión, es necesario declararlo con la cláusula *#REQUIRED*; si no lo es, se debe poner *#IMPLIED*, o bien *#FIXED* si el valor de dicho atributo se debe mantener fijo a lo largo de todo el documento para todos los elementos del mismo tipo.

Cabe destacar un aspecto de cara a la optimización del diseño de **DTDs**. En muchas ocasiones es necesario decidir entre especificar atributos de los elementos como elementos descendientes o como atributos en sí mismos. Suponiendo una pieza de maquinaria con unas características determinadas, las cuales se pueden representar de las siguientes formas:

```
<pieza>MiPieza
        <color>Rojo</color>
</pieza>

<pieza color="Rojo">MiPieza</pieza>
```

Figura 4. 6: Diferentes formas de especificar atributos

En la primera forma, el procesador tiene que bajar al siguiente nivel del árbol de elementos para saber el color de MiPieza, mientras que en el segundo caso lo puede obtener haciendo referencia directamente el atributo color del elemento en el que estás.

Existen multitud de desarrolladores que prefieren el primer modo por estar más acorde con la filosofía de **XML**. Según ella, las etiquetas hacen referencia siempre a su contenido, sin necesidad de acudir al uso de atributos como se hace en **HTML**.

## 4.4.- Entidades

Mediante estos elementos especiales es posible dotar de modularidad a los documentos **XML**. Se pueden definir, del mismo modo que los ya mencionados **DTDs**, dentro del mismo documento **XML** o en **DTDs** externos.

Anteriormente, se han mencionado los caracteres especiales &, ", ', < y >. Se trata de entidades que han de escribirse mediante las declaraciones: *&*, *"*, *'*, *<* y *>*. Es decir, que cuando se desee hacer referencia alguna entidad definida dentro de un mismo documento o en otro externo, se debe utilizar la sintaxis *&nombre;*.

Existe un conjunto de entidades predefinidas de caracteres **ISO**. Sin embargo, las entidades no solo sirven para incluir caracteres especiales no **ASCII**. También se pueden usar para incluir cualquier documento u objeto externo al documento propio.

Por ejemplo, y como uso más simple, se puede crear en un **DTD** o en el documento **XML** una entidad que haga referencia a un nombre largo:

```
<!ENTITY DAT “Delegación de Alumnos de Telecomunicaciones” >
```

Figura 4. 7: Ejemplo de definición de entidad

De esta forma, cada vez que se desee que en un documento aparezca el nombre *“Delegación de Alumnos de Telecomunicaciones”*, bastará con escribir *&DAT;*. Se trata de un sencillo método cuyos beneficios son claros.

El texto al que se hace referencia mediante la entidad puede ser de cualquier tamaño y contenido. Si se desea incluir una porción de otro documento muy largo que haya sido guardado aparte, de tipo XML o cualquier otro, se puede hacer de este modo:

```
<!ENTITY midoc SYSTEM http://www.url_prueba/~aoliva/DTD/ficha.dtd >
```

Figura 4. 8: Ejemplo de entidad que referencia a un texto

Del mismo modo que con los **DTDs** externos, se indica al procesador con *SYSTEM* que la referencia es externa y que lo que sigue es una **URI** estándar, y es lo que queda

entrecomillado a continuación. Lo expuesto hasta ahora se aplica dentro de documentos **XML**, pero no de **DTDs**. Para incluir entidades en **DTDs** se debe emplear el carácter *%*:

```
<!ENTITY % uno “(uno | dos | tres) ” >
```

Figura 4. 9: Inclusión de entidades en DTDs

Y para expandirlo en un **DTD** habrá que escribir:

```
<!ELEMENT numero (%uno;) #IMPLIED>
```

Figura 4. 10: Expansión de entidades en DTDs

## 4.5.- Modelo de Objetos de Documentos: DOM

El modelo de objetos de documentos del **W3C**, o *Document Object Model* (**DOM**) es una representación interna estándar de la estructura de un documento, y proporciona un interfaz al programador (**API**) para poder acceder de forma fácil, consistente y homogénea a sus elementos, atributos y estilo. Es un modelo independiente de la plataforma y del lenguaje de programación. El **W3C** establece varios niveles de actuación, coincidiendo con el tiempo en que se presentan como recomendación:

- ***Nivel 1***: se refiere a la parte interna, y modelos para **HTML** y **XML**. Contiene funcionalidades para la navegación y manipulación de documentos. Tiene 2 partes: el *core o parte básica*, referida a documentos **XML**, y la *parte HTML*, referida precisamente a los **HTML**.

- ***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 mismo. También incluirá un modelo de eventos para soportar los espacios de nombres XML y consultas enriquecidas.

- ***Posteriores niveles*** especificarán interfaces a posibles sistemas de ventanas, manipulación de DTD y modelos de seguridad.

El objetivo es que de una vez por todas cualquier *script* pueda ejecutarse de forma más o menos homogénea en cualquier navegador que soporte dicho **DOM**. Tener una plataforma estándar en la que poder crear contenidos sin temor a no estar soportado por alguna marca o versión de navegador, que además sea potente y versátil.

Y por supuesto, como el conjunto de piezas que el **W3C** está creando para su uso en el intercambio de documentos e información, no estará sujeto al ámbito de los navegadores, sino que su uso será extensible a cualquier tipo de aplicación que acceda a esos documentos.