3 CONCEPTOS PREVIOS

Para poder ahondar más en el mundo de los Servicios Web, y más particularmente en REST, es necesario introducir algunos **conceptos previos**. En este capítulo se verán los conceptos de localizador de recursos (**URL y UUID**), protocolo de transferencia (**HTTP**) y lenguaje de marcas para describir y estructurar los datos (**XML**). Estas son las bases sobre las que se construyen los estilos de arquitectura Web REST y SOAP. Su estudio es imprescindible para poder comprenderlos.

3.1 URL

URL significa ***Uniform* *Resource* *Locutor*** [4], es decir, localizador uniforme de recurso. Es una secuencia de caracteres, de acuerdo a un formato estándar, que se usa para nombrar recursos, como documentos e imágenes en Internet, por su localización.

Las **URL** [RFC 1738] fueron una innovación fundamental en la historia de Internet. Fueron usadas por primera vez por Tim Berners-Lee en 1991, para permitir a los autores de documentos establecer hiperenlaces en la World Wide Web (WWW o Web). Desde 1994, en los estándares Internet, el concepto de URL ha sido incorporado dentro del más general de **URI** (***U**niform **R**esource **I**dentifier* - Identificador Uniforme de Recurso), pero el término URL aún se utiliza ampliamente.

Aunque nunca fueron mencionadas como tal en ningún estándar, mucha gente cree que las iniciales URL significan Universal Resource Locator (Localizador Universal de Recurso). Esta interpretación puede ser debida al hecho de que, aunque la U en URL siempre ha significado Uniforme, la U de URI significó en un principio Universal, antes de la publicación del [RFC 2396].

La URL es la cadena de caracteres con la cual se asigna una dirección única a cada uno de los recursos de información disponibles en Internet. Existe un URL único para cada página de cada uno de los documentos de la *World Wide Web*, para todos los elementos de *Gopher* y todos los grupos de debate *USENET*, y así sucesivamente.

El URL de un recurso de información es su dirección en Internet, la cual permite que el navegador la encuentre y la muestre de forma adecuada. Por ello el URL combina el nombre del ordenador que proporciona la información, el directorio donde se encuentra, el nombre del fichero y el protocolo a usar para recuperar los datos. Y reemplaza la dirección numérica o IP de los servidores haciendo de esta manera más fácil la navegación, si no de otra forma se tendría que hacer bajo direcciones del tipo `http://148.210.01.7` en vez de `http://pagina.com`

3.1.1 Sintaxis

El formato general de una URL es:

```
protocolo://máquina/directorio/fichero
```

También pueden añadirse otros datos:

```
protocolo://usuario:contraseña@máquina:puerto/directorio/fichero
```

Por ejemplo: `http://es.Wikipedia.org/`

La especificación detallada se encuentra en la [RFC 1738], titulada Uniform Resource Locators.

3.1.1.1 Esquema URL

Una URL se clasifica por su esquema, que generalmente indica el protocolo de red que se usa para recuperar, a través de la red, la información del recurso identificado. Una URL comienza con el nombre de su esquema, seguida por dos puntos, seguido por una *parte específica del esquema*.

Algunos ejemplos de esquemas URL:

- http – recursos HTTP
- https - HTTP sobre SSL
- ftp – File Transfer Protocol
- mailto - direcciones E-mail
- ldap - búsquedas LDAP Lightweight Directory Access Protocol
- file - recusos disponibles en el ordenador local, o en una red local
- news - grupos de noticias Usenet (newsgroup)
- gopher - el protocolo Ghoper(ya en desuso)
- telnet - el protocolo telnet
- data - el esquema para insertar pequeños trozos de contenido en los documentos Data: URL

Algunos de los esquemas URL, como los populares "mailto", "http", "ftp" y "file", junto a los de sintaxis general URL, se detallaron por primera vez en 1994, en el Request for Comments [RFC 1630].

3.1.1.2 Sintaxis Genérica URL

Todas las URLs, independientemente del esquema, deben seguir una sintaxis general. Cada esquema puede determinar sus propios requisitos de sintaxis para su parte específica, pero la URL completa debe seguir la sintaxis general.

Usando un conjunto limitado de caracteres, compatible con el subconjunto imprimible de ASCII, la sintaxis genérica permite a las URLs representar la dirección de un recurso, independientemente de la forma original de los componentes de la dirección.

Los esquemas que usan protocolos típicos basados en conexión usan una sintaxis común para "URI genéricas", definida a continuación:

`esquema://autoridad/ruta?consulta#fragmento`

La *autoridad* consiste usualmente en el nombre o Dirección IP de un servidor, seguido a veces de dos puntos (":") y un número de puerto TCP. También puede incluir un nombre de usuario y una clave, para autenticarse ante el servidor.

La *ruta* es la especificación de una ubicación en alguna estructura jerárquica, usando una barra diagonal ("/") como delimitador entre componentes.

La *consulta* habitualmente indica parámetros de una consulta dinámica a alguna base de datos o proceso residente en el servidor.

El *fragmento* identifica a una porción de un recurso, habitualmente una ubicación en un documento.

3.1.1.2.1 Ejemplo: URLs en HTTP

Las URLs empleadas por HTTP, el protocolo usado para transmitir páginas Web, es el tipo más popular de URL y puede ser usado para mostrarse como ejemplo. La sintaxis de una HTTP URL es:

`esquema://anfitrión:puerto/ruta?parámetro=valor#enlace`

- `Esquema`, en el caso de HTTP, en la mayoría de las veces equivale a `http`, pero también puede ser `https` cuando se trata de HTTP sobre una conexión TLS (para hacer más segura la conexión).
- Muchos navegadores Web permiten el uso de la siguiente estructura: `esquema://usuario:contraseña@anfitrión:puerto/...` para autenticación en HTTP. Este formato ha sido usado como una "hazaña" para hacer difícil el identificar correctamente al servidor involucrado. En consecuencia, el soporte para este formato ha sido dejado de lado por algunos navegadores. La sección 3.2.1 de RFC 3986 recomienda que los navegadores deben mostrar el usuario / contraseña de otra forma que no sea en la barra de direcciones, a causa de los problemas de seguridad mencionados y porque las contraseñas no deben ser nunca mostradas como texto claro.

- `Anfitrión`, la cual es probablemente la parte que más sobresale de una URL, es en casi todos los casos el nombre de dominio de un servidor, p.ej.: `www.wikipedia.org`, `google.com`, etc.
- La porción `:puerto` especifica un número de puerto TCP. Normalmente es omitido (en este caso, su valor por omisión es `80`) y probablemente, para el usuario es lo que tiene menor relevancia en todo el URL.
- La porción `ruta` es usada por el servidor (especificado en `anfitrión`) de cualquier forma en la que su software lo establezca, pero en muchos casos se usa para especificar un nombre de archivo, posiblemente precedido por nombres de directorio. Por ejemplo, en la ruta `/wiki/Vaca`, `wiki` sería un (seudo-)directorio y `Vaca` sería un (seudo-)nombre de archivo.
- La parte mostrada arriba como `?parámetro=valor` se conoce como porción de *consulta* (o también, porción de *búsqueda*). Puede ser omitido, puede haber una sola pareja parámetro-valor como en el ejemplo, o pueden haber muchas de ellas, lo cual se expresa como `?param=valor&otroParam=valor&...`. Las parejas parámetro-valor sólo son relevantes si el archivo especificado por la ruta no es una página Web simple y estática, sino algún tipo de página automáticamente generada. El software generador usa las parejas parámetro-valor de cualquier forma en que se establezca; en su mayoría transportan información específica a un usuario y un momento en el uso del sitio, como términos concretos de búsqueda, nombres de usuario, etc. (Observe, por ejemplo, de qué forma se comporta URL en la barra de direcciones de su navegador durante una búsqueda Google: su término de búsqueda es pasado a algún programa sofisticado en google.com como un parámetro, y el programa de Google devuelve una página con los resultados de la búsqueda.)
- La parte `#enlace`, por último, es conocida como *identificador de fragmento* y se refiere a ciertos lugares significativos dentro de una página; por ejemplo, esta página tiene enlaces internos hacia cada cabecera de sección a la cual se puede dirigir usando el ID de fragmento. Esto es relevante cuando una URL de una página ya cargada en un navegador permite saltar a cierto punto en una página larga. Un ejemplo sería un enlace que conduce a la misma página y al comienzo de la sección. (Observe que cambiará la URL en la barra de dirección del navegador).

3.1.2 URI y Referencias URI

URI es un **U**niform **R**esource **I**dentifier, identificador uniforme de recursos [3]. Texto corto que identifica unívocamente cualquier recurso (servicio, página, documento, dirección de correo electrónico, enciclopedia...) accesible en una red.

Normalmente un URI consta de dos partes:

- **Identificador** del método de acceso (protocolo) al recurso, por ejemplo http:, mailto:, ftp:
- **Nombre del recurso**, por ejemplo "//es.Wikipedia.org"

La especificación detallada se encuentra en la [RFC 2396] - Uniform Resource Identifiers (URI): Generic Syntax.

La principal diferencia entre la definición de URI y URL es que las URIs identifican a cualquier recurso pero no tienen por qué indicar como acceder al mismo. Las URL indican como acceder al recurso. Como puede verse, la definición de URI es más genérica, engloba a URL dentro de ella.

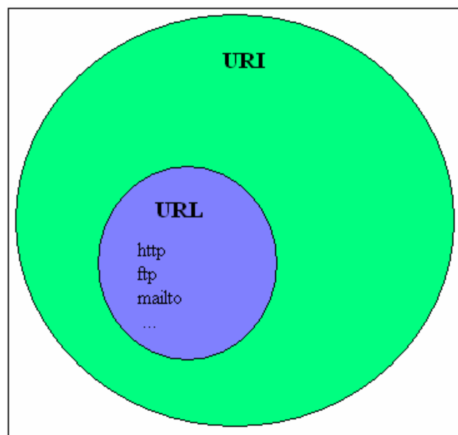

Figura 2: URI-URL

El término **referencia URI** se refiere a un caso particular de una URI, o una porción de éste, tal como es usada en un documento HTML, por ejemplo, para referirse a un recurso particular. Una referencia URI habitualmente se parece a una URL o a la parte final de una URL. Las referencias URI introducen dos nuevos conceptos: la distinción entre referencias *absolutas* y *relativas*, y el concepto de un identificador de fragmento.

Un **URL absoluto** es una referencia URI que es parecida a las URLs definidas arriba; empieza por un esquema seguido de dos puntos (":") y de una parte específica del esquema. Una **URL relativa** es una referencia URI que comprende sólo la parte específica del esquema de una URL, o de algún componente de seguimiento de aquella parte. El esquema y componentes principales se infieren del contexto en el cual aparece la referencia URL: el **URI base** (o **URL base**) del documento que contiene la referencia.

Una referencia URI también puede estar seguida de un carácter de numeral ("#") y un puntero dentro del recurso referenciado por el URI en conjunto. Esto no es parte de la URI como tal, sino que es pensado para que el "agente usuario" (el navegador) lo interprete después que una representación del recurso ha sido recuperada. Por tanto, no se supone que sean enviadas al servidor en forma de solicitudes HTTP.

Ejemplos de URLs absolutos:

- `http://es.wikipedia.org/w/wiki.phtml?title=URL&action=history`
- `http://es.wikipedia.org/wiki/URL#Esquemas_en_URL`

Ejemplos de URLs relativos:

- `//en.wikipedia.org/wiki/Uniform_Resource_Locator`
- `/wiki/URL`
- `URL#Referencias_URI`

3.1.3 Diferenciación entre mayúsculas/minúsculas

De acuerdo al estándar actual, en los componentes `esquema` y `anfitrión` no se diferencian mayúsculas y minúsculas, y cuando se normalizan durante el procesamiento, deben estar en minúsculas. Se debe asumir que en otros componentes sí hay diferenciación. Sin embargo, en la práctica, en otros componentes aparte de los de `protocolo` y `anfitrión`, esta diferenciación es dependiente del servidor Web y del sistema operativo del sistema que albergue al servidor.

3.1.4 URLs en el uso diario

Un HTTP URL combina en una dirección simple los cuatro elementos básicos de información que son necesarios para recuperar un recurso desde cualquier parte de Internet:

- El protocolo que se usa para comunicar,
- El anfitrión (servidor) con el que se comunica,
- El puerto de red en el servidor para conectarse,
- La ruta al recurso en el servidor (por ejemplo, su nombre de archivo).

Un URL típico puede lucir como:

`http://es.wikipedia.org:80/wiki/Special:Search?search=tren&go=Go`

Donde:

- *http* es el protocolo,
- *es.wikipedia.org* es el anfitrión,
- *80* es el número de puerto de red en el servidor (siendo 80 el valor por omisión para el protocolo HTTP, esta porción puede ser omitida por completo),
- */wiki/Special:Search* es la ruta de recurso,
- *?search=tren&go=Go* es la cadena de búsqueda; esta parte es opcional.

Muchos navegadores Web no requieren que el usuario ingrese "`http://`" para dirigirse a una página Web, puesto que HTTP es el protocolo más común que se usa en navegadores Web. Igualmente, dado que 80 es el puerto por omisión para HTTP, usualmente no se especifica. Usualmente uno sólo ingresa una URL parcial tal como `www.wikipedia.org/wiki/Train`. Para ir a una página principal normalmente se entra sólo el nombre de anfitrión, como `www.wikipedia.org`.

Dado que el protocolo HTTP permite que un servidor responda a una solicitud redireccionando el navegador Web a una URL diferente, muchos servidores adicionalmente permiten a los usuarios omitir ciertas partes de la URL, tales como la parte "www.", o el carácter numeral ("#") de rastreo si el recurso en cuestión es un directorio. Sin embargo, estas omisiones técnicamente constituyen una URL diferente, de modo que el navegador Web no puede hacer estos ajustes, y tiene que confiar que el servidor responda con una redirección. Es posible para un servidor Web (pero debido a una extraña tradición) ofrecer dos páginas diferentes para URLs que difieren únicamente en un carácter "#".

Nótese que en `es.wikipedia.org/wiki/Tren`, el orden jerárquico de los cinco elementos es org (dominio genérico de nivel superior) - wikipedia (dominio de segundo nivel) - es (subdominio) - wiki - Train; es decir, antes del primer "/" se lee de derecha a izquierda, y después el resto se lee de izquierda a derecha.

3.1.5 El gran marco

El término URL también es usado fuera del contexto de la *World Wide Web*. Los servidores de bases de datos especifican URLs como un parámetro para hacer conexiones a éstos. De forma similar, cualquier aplicación cliente-servidor que siga un protocolo particular puede especificar un formato URL como parte de su proceso de comunicación.

Ejemplo de una URL en una base de datos:

```
jdbc:datadirect:oracle://myserver:1521;sid=testdb
```

Si una página Web está en forma singular y más o menos permanentemente definida a través de una URL, ésta puede ser enlazada. Éste no siempre es el caso, por ejemplo., una opción de menú puede cambiar el contenido de un marco dentro de la página, sin que esta nueva combinación tenga su propia URL. Una página Web puede depender también de información almacenada temporalmente. Si el marco o página Web tiene su propia URL, esto no es siempre obvio para alguien que quiere enlazarse a ella: la URL de un marco no es mostrado en la barra de direcciones del navegador, y una página sin barra de dirección pudo haber sido producida. La URL se puede encontrar en el código fuente y/o en las "propiedades" de varios componentes de la página.
Aparte del propósito de enlazarse a una página o a un componente de página, puede ocurrir que se quiera conocer el URL para mostrar únicamente el componente, o superar restricciones tales como una ventana de navegador que no tenga barras de herramientas y/o que sea de tamaño pequeño y no ajustable.

Los servidores Web también tienen la capacidad de direccionar URLs si el destino ha cambiado, permitiendo a los sitios cambiar su estructura sin afectar los enlaces existentes. Este proceso se conoce como redireccionamiento de URLs.

3.2 UUID

Un ***Universally Unique Identifier*** [RFC 4122] es un identificador estándar usado en la construcción de Software, estandarizado por el Open Software Foundation (OSF) como parte del Distributed Computing Environment (DCE) [27]. La función de los UUIDs es permitir a los sistemas distribuidos identificar información única sin necesidad de un sistema de coordinación central. Así, cualquiera puede crear UUID y usarla para identificar algo con cierta confianza de que el identificador nunca se usará inintencionadamente por otra persona. La información etiquetada con UUIDs puede ser usada en bases de datos simples sin necesidad de resolver conflictos de nombres. El uso más extendido de este estándar son las Microsoft´s Globally Unique Identifiers (GUIDs) que implementan este estándar.

Un UUID es esencialmente un número de 16 octetos y su forma canónica se parece a algo como esto:

550e8400-e29b-11d4-a716-446655440000

Los UUIDs se documentan como parte de ISO/IEC 11578:1996 "Information technology -- Open System Interconection – Remote Procedura Call (RPC)".

El esquema de UUIDs fue criticado por no ser suficientemente opaco; revelaba la identidad del ordenador que generaba el UUID y el momento en el que lo hizo.

3.3 HTTP

El Protocolo de Transferencia de Hipertexto (Hypertext Transfer Protocol) es un sencillo protocolo cliente-servidor que articula los intercambios de información entre los clientes Web y los servidores HTTP. La especificación completa del protocolo HTTP 1/0 está recogida en el [RFC 1945]. Fue propuesto por Tim Berners-Lee, atendiendo a las necesidades de un sistema global de distribución de información como el World Wide Web.

3.3.1 Características y funcionamiento

Desde el punto de vista de las comunicaciones, está soportado sobre los servicios de conexión **TCP/IP**, y funciona de la misma forma que el resto de los servicios comunes de entornos UNIX: un proceso servidor escucha en un puerto de comunicaciones TCP

(por defecto, el 80), y espera las solicitudes de conexión de los clientes Web. Una vez que se establece la conexión, el protocolo TCP se encarga de mantener la comunicación y garantizar un intercambio de datos libre de errores.

HTTP se basa en sencillas operaciones de solicitud/respuesta. Un cliente establece una conexión con un servidor y envía un mensaje con los datos de la solicitud. El servidor responde con un mensaje similar, que contiene el estado de la operación y su posible resultado. Todas las operaciones pueden adjuntar un objeto o recurso sobre el que actúan; cada objeto Web es conocido por su URL.

Los recursos u objetos que actúan como entrada o salida de un comando HTTP están clasificados por su descripción MIME. De esta forma, el protocolo puede intercambiar cualquier tipo de dato, sin preocuparse de su contenido. La transferencia se realiza en modo binario, byte a byte, y la identificación MIME permitirá que el receptor trate adecuadamente los datos.

Las principales **características** del protocolo **HTTP** son:

- Toda la comunicación entre los clientes y servidores se realiza a partir de caracteres de 8 bits. De esta forma, se puede transmitir cualquier tipo de documento: texto, binario, etc., respetando su formato original.

- Permite la transferencia de objetos multimedia. El contenido de cada objeto intercambiado está identificado por su clasificación MIME.

- Existen tres verbos básicos (hay más, pero por lo general no se utilizan) que un cliente puede utilizar para dialogar con el servidor: **GET**, para recoger un objeto, **POST**, para enviar información al servidor y **HEAD**, para solicitar las características de un objeto (por ejemplo, la fecha de modificación de un documento HTML).

- Cada operación **HTTP** implica una conexión con el servidor, que es liberada al término de la misma. Es decir, en una operación se puede recoger un único objeto. En la actualidad se ha mejorado este procedimiento, permitiendo que una misma conexión se mantenga activa durante un cierto periodo de tiempo, de forma que sea utilizada en sucesivas transacciones. Este mecanismo, denominado *HTTP Keep Alive*, es empleado por la mayoría de los clientes y servidores modernos.

- No mantiene estado. Cada petición de un cliente a un servidor no es influida por las transacciones anteriores. El servidor trata cada petición como una operación totalmente independiente del resto.

- Cada objeto al que se aplican los verbos del protocolo está identificado a través de la información de situación del final de la URL.

Cada vez que un cliente realiza una petición a un servidor, se ejecutan los siguientes pasos:

1. Un usuario accede a una **URL**, seleccionando un enlace de un documento **HTML** o introduciéndola directamente en el campo Dirección del cliente Web.

2. El cliente Web descodifica la **URL**, separando sus diferentes partes. Así identifica el protocolo de acceso, la dirección DNS o IP del servidor, el posible puerto opcional (el valor por defecto es 80) y el objeto requerido del servidor.

3. Se abre una conexión **TCP/IP** con el servidor, llamando al puerto TCP correspondiente.

4. Se realiza la petición. Para ello, se envía el comando necesario (**GET**, **POST**, **HEAD**,…), la dirección del objeto requerido (el contenido de la URL que sigue a la dirección del servidor), la versión del protocolo HTTP empleada (casi siempre HTTP/1.0) y un conjunto variable de información, que incluye datos sobre las capacidades del navegador, datos opcionales para el servidor, etc.

5. El servidor devuelve la respuesta al cliente. Consiste en un código de estado y el tipo de dato MIME de la información de retorno, seguido de la propia información.

6. Se cierra la conexión **TCP**. Si no se utiliza el modo *HTTP Keep Alive*, este proceso se repite para cada acceso al servidor **HTTP**.

A continuación se muestra un esquema gráfico de este proceso en la Figura [3]:

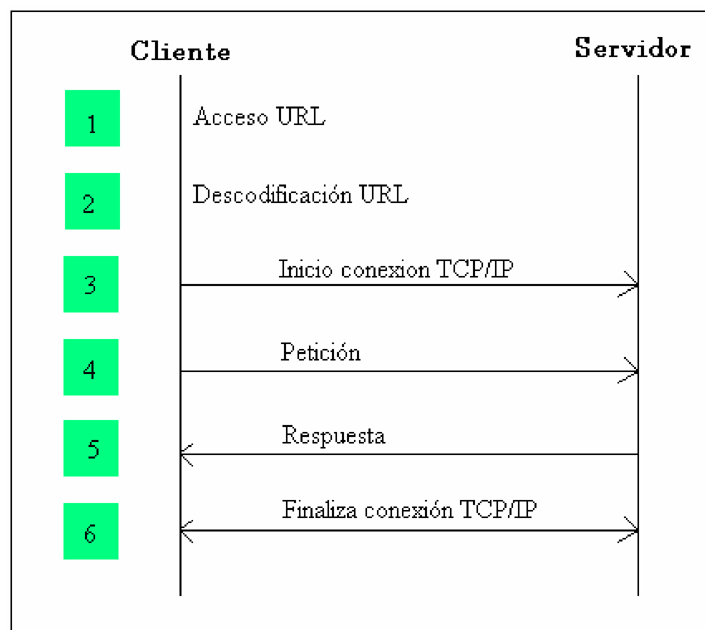

Figura 3: Petición HTTP

El diálogo con los servidores HTTP se establece a través de mensajes formados por líneas de texto, cada una de las cuales contiene los diferentes comandos y opciones del protocolo. Sólo existen dos **tipos de mensajes**, uno para realizar peticiones y otro para devolver la correspondiente respuesta. La estructura general de los dos tipos de mensajes se puede ver en el siguiente esquema de la Figura [4]:

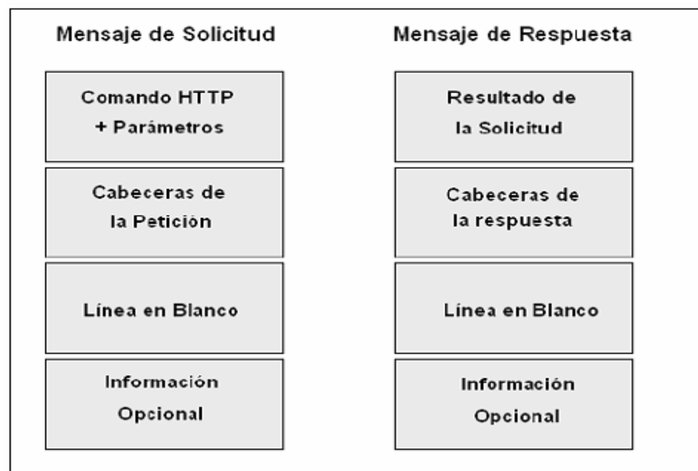

Figura 4: Mensaje HTTP para solicitud/respuesta

La primera línea del mensaje de solicitud contiene el comando que se solicita al servidor **HTTP**, mientras que en la respuesta contiene el resultado de la operación que es un código numérico que permite conocer el éxito o fracaso de la operación. Después aparece, para ambos tipos de mensajes, un conjunto de cabeceras (obligatorias y opcionales), que condicionan y matizan el funcionamiento del protocolo.

La separación entre cada línea del mensaje se realiza con un par *CR-LF* (retorno de carro más nueva línea). El final de las cabeceras se indica con una línea en blanco, tras la cual se pueden incluir los datos transportados por el protocolo, por ejemplo, el documento HTML que devuelve un servidor.

3.3.2 Comandos del protocolo HTTP

El protocolo *HTTP* consta de los siguientes comandos que permiten realizar las acciones pertinentes: *GET, HEAD, POST, PUT, DELETE, LINK, UNLINK***.** A continuación se va a explicar detenidamente cada uno de ellos:

- **GET**, se usa para recoger cualquier tipo de información del servidor. Se utiliza siempre que se pulsa sobre un enlace o se teclea directamente a una URL. Como resultado, el servidor HTTP envía el documento correspondiente a la URL.

  Este comando puede ir acompañado de una cabecera con los siguientes parámetros:

- ***Accept***: indica los formatos de archivo que puede soportar el cliente.
- ***Accept-Charset***: indica los conjuntos de caracteres que acepta.
- ***Accept-Encoding***: el tipo de codificación que acepta.
- ***Accept-Language***: el lenguaje en el que se hará la respuesta de la cabecera.
- ***Authorization***: clave para tener acceso a lugares restringidos donde se requiere nombre de contraseña y el formato de autorización empleado.
- ***Connection***: especifica las opciones requeridas para una conexión.
- ***Date***: representa la fecha y la hora a la que se envió el comando.
- ***From***: campo opcional que incluye la dirección de correo electrónico del usuario del cliente.
- ***Host***: especifica la dirección del host del cliente y el puerto de conexión que ha empleado.
- ***If-Modified-Since***: condición de que sólo se responda a la solicitud si la fecha de última modificación del objeto es superior a la indicada en su parámetro.
- ***If-UnModified-Since***: sólo se responde a la solicitud si la fecha de última modificación del objeto no ha cambiado a partir de la fecha indicada en su parámetro.
- ***Pragma***: para incluir directivas de implementación.
- ***Proxy-Authorization***: para identificarse a un Proxy.
- ***Referer***: contiene la URL de la página que le ha dado acceso a la que está solicitando. Esto puede interesarle a los autores de páginas Web para saber en que lugares existen enlaces de su página.
- ***Range***: establecer un rango de Bytes del contenido de la respuesta.
- **Transfer-Encoding**: indica el tipo de codificación aplicada del contenido que responderá el servidor.
- ***Upgrade***: especifica qué protocolos suplementarios soporta el cliente, si también soportara FTP u otros.
- ***User-Agent***: especifica cual es el nombre del cliente y su versión. Tiene un segundo parámetro corresponde al sistema operativo donde corre el cliente.
- ***Via***: sirve para obtener la ruta de routers que ha recorrido el mensaje.

- **HEAD**, que es igual que el comando GET, y puede usar las mismas cabeceras, pero este comando le pide al servidor que le envíe sólo la cabecera como respuesta. Es utilizado por los gestores de caché de páginas o los servidores proxy, para conocer cuándo es necesario actualizar la copia que se mantiene de un fichero.

- **POST**, que tiene un funcionamiento idéntico al comando HEAD, pero además, este comando envía datos de información al servidor, normalmente originaria de un formulario Web para que el servidor los administre o los añada a una base de datos.

Posteriormente se han definido algunos comandos adicionales, que sólo están disponibles en determinadas versiones de servidores HTTP. La última versión de HTTP, denominada 1.1, recoge estas y otras novedades, que se pueden utilizar, por ejemplo:

- **PUT**, actualiza información sobre un objeto del servidor. Es similar a POST, pero en este caso, la información enviada al servidor debe ser almacenada en la URL que acompaña al comando. Así se puede actualizar el contenido de un documento.

- **DELETE**, elimina el documento especificado del servidor.

- **LINK**, crea una relación entre documentos.

- **UNLINK**, elimina una relación existente entre documentos del servidor.

Estos métodos pueden usarse, por ejemplo, para editar las páginas de un servidor Web trabajando de manera remota.

Cuando el servidor envía la respuesta al cliente, le envía la información solicitada y una cabecera con una serie **campos**, estos campos son:

- ***Age***: tiempo transcurrido desde que se creó la respuesta.

- ***Allow***: especifica qué comandos puede utilizar el cliente respecto al objeto pedido, como pueden ser GET o HEAD.

- ***Expires***: fecha en la que caducará el objeto adjunto.

- ***Last-Modified***: fecha de la última modificación del objeto.

- ***Location***: se usa para redirigir la petición a otro objeto. Informa sobre la dirección exacta del objeto al que se ha accedido.

- ***Proxy-Authenticate***: indica el esquema de autentificación en caso de que un Proxy lo requiera.

- ***Public***: da una lista de los comandos que reconoce el servidor.

- ***Retry-After***: si no puede ofrecer un objeto provisionalmente, indica la fecha para que se vuelva a hacer la petición.

- ***Server***: especifica el tipo y versión del servidor.

- ***Vary***: indica que hay mas de una posible respuesta a la petición pero solo elige una.

- ***Warning***: específica información adicional sobre el estado de la respuesta.

- ***WWW-Autenticate***: usado cuando se accede a un lugar restringido, sirve para informar de las maneras de autentificarse que soporta el objeto del servidor.

Hay otros **parámetros de información de cabeceras**, que son válidos tanto para mensajes del cliente como para respuestas del servidor. Estas cabeceras son:

- ***Content-Type***: descripción MIME de la información que contiene el mensaje para dar el tratamiento conveniente a los datos que se envían.

- ***Content-Length***: longitud en Bytes de los datos enviados.

- ***Content-Encoding***: especifica el formato en el que están codificados los datos enviados.

- ***Date***: fecha local de la operación. Las fechas deben incluir la zona horaria en que reside el sistema que genera la operación. Por ejemplo: *Sunday, 12- Dec-96 12:21:22 GMT+01*. No existe un formato único en las fechas; incluso es posible encontrar casos en los que no se dispone de la zona horaria correspondiente, con los problemas de sincronización que esto produce. Los formatos de fecha a emplear están recogidos en las [RFC 1036] y [RFC 1123].

- ***Pragma***: Permite incluir información variada relacionada con el protocolo HTTP en la solicitud o respuesta que se está realizando. Por ejemplo, un cliente envía un *Pragma: no-caché* para informar de que desea una copia nueva del recurso especificado.

Ante cada transacción con un servidor HTTP, éste devuelve un **código numérico** que informa sobre el resultado de la operación, como primera línea del mensaje de respuesta. Estos códigos aparecen en algunos casos en la pantalla del cliente, cuando se produce un error.

Los **códigos** de estados están clasificados en **cinco categorías**:

- ***1xx***: mensajes informativos. En HTTP/1.0 no se utilizan, y están reservados para un futuro uso.

- ***2xx***: mensajes asociados con operaciones realizadas correctamente.

- ***3xx***: mensajes de redirección, que informan de operaciones complementarias que se deben realizar para finalizar la operación.

- ***4xx***: errores del cliente; el requerimiento contiene algún error, o no puede ser realizado.

- ***5xx***: errores del servidor, que no ha podido llevar a cabo una solicitud.

La lista de códigos es la que podemos ver en la Tabla [1]

Código	***Comentario***	***Descripción***
200	OK	Operación realizada satisfactoriamente.
201	Created	La operación ha sido realizada correctamente, y como resultado se ha creado un nuevo objeto, cuya URL de acceso se proporciona en el cuerpo de la respuesta. Este nuevo objeto ya está disponible. Puede ser utilizado en sistemas de edición de documentos.
202	Accepted	La operación ha sido realizada correctamente, y como resultado se ha creado un nuevo objeto, cuya URL de acceso se proporciona en el cuerpo de la respuesta. El nuevo objeto no está disponible por el momento. En el cuerpo de la respuesta se debe informar sobre la disponibilidad de la información.
204	No Content	La operación ha sido aceptada, pero no ha producido ningún resultado de interés. El cliente no deberá modificar el documento que está mostrando en este momento.
301	Moved Permanently	El objeto al que se accede ha sido movido a otro lugar de forma permanente. El servidor proporciona, además, la nueva URL en la variable *Location* de la respuesta. Algunos browsers acceden automáticamente a la nueva URL. En caso de tener capacidad, el cliente puede actualizar la URL incorrecta.
302	Moved Temporarily	El objeto al que se accede ha sido movido a otro lugar de forma temporal. El servidor proporciona, además, la nueva URL en la variable *Location* de la respuesta. Algunos browsers acceden automáticamente a la nueva URL. El cliente no debe modificar ninguna de las referencias a la URL errónea.
304	Not Modified	Cuando se hace un GET condicional, y el documento no ha sido modificado, se devuelve este código de estado.
400	Bad Request	La petición tiene un error de sintaxis y no es entendida por el servidor.
401	Unauthorized	La petición requiere una autorización especial, que normalmente consiste en un nombre y clave que el servidor verificará. El campo *WWW-Autenticate* informa de los protocolos de autentificación aceptados para este recurso.

403	Forbidden	Está prohibido el acceso a este recurso. No es posible utilizar una clave para modificar la protección.
404	Not Found	La URL solicitada no existe.
500	Internal Server Error	El servidor ha tenido un error interno, y no puede continuar con el procesamiento.
501	Not Implemented	El servidor no tiene capacidad, por su diseño interno, para llevar a cabo el requerimiento del cliente.
502	Bad Gateway	El servidor, que está actuando como Proxy o pasarela, ha encontrado un error al acceder al recurso que había solicitado el cliente.
503	Service Unavailable	El servidor está actualmente deshabilitado, y no es capaz de atender el requerimiento.

Tabla 1 Lista de códigos de estado http

3.4 XML

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 se podría 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 tampoco habrá de estar atado 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.

3.4.1 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>Cubo Velazquez</apellidos>
	<direccion>c/Luis Chamizo, 14</direccion>
</ficha>
```

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>
```

- 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.

3.4.2 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/~acubo/DTD/ficha.dtd">
```

  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>Cubo Vlezquez</apellidos>
        <direccion>c/Luis Chamizo, 14</direccion>
</ficha>
```

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 existe en un **DTD** la declaración:

```
<!ELEMENT ficha (nombre+, apellidos+, direccion*, foto?, telefono*|fax*)>
```

Se sabrá 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 el indicado en la Figura [5]

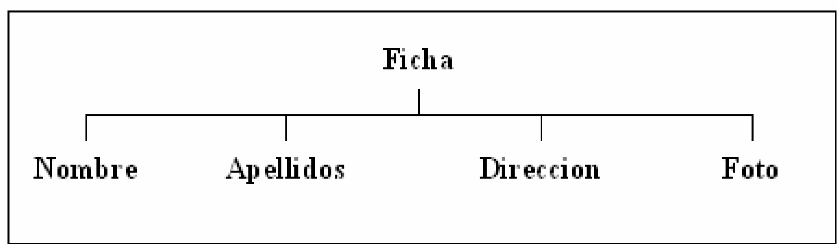

Figura 5: Jerarquía del documento ejemplo XML

Se observa que se trata de un documento muy sencillo, con una profundidad de 2 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 cada vez que se usa el elemento en cuestión declarar este atributo, 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>
```

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**.

3.4.3 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"
```

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/~acubo/DTD/ficha.dtd >
```

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) " >
```

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

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

3.4.4 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.

3.5 Conclusiones

En este capítulo se han expuesto los conocimientos básicos que son necesarios para poder abordar REST y SOAP de manera eficaz. Se han estudiado los conceptos de localizador de recursos (URL y UUID), protocolo de transferencia (HTTP) y lenguaje de marcas para describir y estructurar los datos (XML). En el siguiente capítulo se verá la arquitectura de Servicios Web SOAP y el lenguaje de descripción de Servicios WSDL. Este estudio será necesario para poder realizar comparaciones entre REST y SOAP.