Tema 8

Aplicaciones web. Tecnologías de programación. JavaScript, applets, servlets y servicios Web. Lenguajes de descripción de datos: HTML, XML y sus derivaciones. Navegadores y lenguajes de programación web. Lenguajes de script.

Guion-resumen

1. **Aplicaciones web**
2. **Tecnologías de programación**
3. **JAVA Script**
4. **Applets**
5. **Servlets**
6. **Servicios web**
7. **Lenguajes de descripción de datos: HTML**
8. **Lenguajes de descripción de datos: XML y sus derivaciones**
9. **Navegadores y lenguajes de programación web. Lenguajes de script**

1. Aplicaciones web

El principal lenguaje utilizado para la creación de páginas web es HTML (*HyperText Markup Language,* lenguaje de marcas con hipertexto). No se considera un lenguaje de programación sino únicamente se habla de "lenguaje de marcas". Las **marcas** o **etiquetas** son trozos de código que habitualmente delimitan un texto para asociarle un significado o unas características. Por ejemplo, el código: <h1>Página web</h1> indica que el texto "Página web" es un título importante en el documento. Los códigos colocados entre los signos < y > son las etiquetas. Las páginas web tienen habitualmente las extensiones .htm y .html.

HTML es un subconjunto de SGML (*Standard Generalized Markup Lenguage*, lenguaje de marcas estándar generalizado). Contiene etiquetas con un significado fijo (por ejemplo, siempre indica texto centrado. Se entremezclan información estructural y de presentación.

XML (*eXtensible Markup Lenguage*, lenguaje de marcas extensible) es también un derivado de SGML. Se trata de un metalenguaje utilizado para definir nuevos lenguajes. En XML, el significado de las etiquetas puede variar. Solo se define la estructura. Ejemplo: <nombre>Juan</nombre> <telefono>912345678</telefono> (no especificamos cómo se debe mostrar esta información). Otras características de XML son anidación de niveles y posibilidad de validación de documentos, para comprobar que su sintaxis es correcta.

El metalenguaje XML se ha utilizado para redefinir el lenguaje HTML, generando el lenguaje XHTML (*eXtensible HyperText Markup Lenguage*). XHTML se considera la evolución de HTML para la creación de páginas web.

El organismo encargado del desarrollo técnico de los estándares de la Web es W3C (World Wide Web Consortium, www.w3.org). En su web podemos encontrar las especificaciones o recomendaciones de los lenguajes HTML, XML y XHTML, entre otros.

El elemento principal de una página web es el enlace o link a otra página. Esta posibilidad de saltar de unas páginas a otras es lo que se conoce como navegación. En realidad, el nombre de web (telaraña) procede precisamente del entramado de enlaces entre unas páginas y otras. Una página web puede contener texto, elementos multimedia (principalmente imágenes), enlaces y objetos incrustados (como *scripts, applets de JAVA u objetos Flash*).

Los principales **formatos de imágenes** utilizados en páginas web son:

— **GIF** (*Graphics Interchange Format*). Formato propietario de Compuserve. Apropiado para pequeños iconos y dibujos, que utilizan colores sólidos, no degradados. 256 colores como máximo. Permite animaciones (una secuencia de imágenes GIF mostradas de forma circular) y transparencias (un color es marcado como transparente). Utiliza compresión sin pérdida.

— **JPEG o JPG** (*Joint Photographics Experts Group*, grupo de expertos fotográficos unidos). Formato abierto que soporta hasta 16 millones

de colores. Sus características son opuestas a GIF. Apropiado para fotografías. Se puede elegir el ratio de compresión: a más compresión, menor tamaño pero menor calidad de imagen.

— **PNG** (*Portable Network Graphics*, gráficos portables de red). Formato abierto, propuesto recientemente por W3C, para sustituir al formato propietario GIF. Utiliza compresión sin pérdida y admite 16 millones de colores. Permite guardar imágenes en modo entrelazado para que el visitante de la página vaya viendo la imagen progresivamente a medida que el navegador la carga (esto también es posible con GIF y JPEG). Soporta canales alfa, para especificar hasta 256 grados de transparencias. PNG no admite animaciones, en su lugar se está desarrollando el formato MNG (*Multiple image Network Graphics*, gráficos de red de imagen múltiple).

Para separar la estructura de la presentación de los documentos HTML, se recomienda utilizar **hojas de estilo CSS** (*Cascade style sheets*, hojas de estilo en cascada). Una hoja de estilos permite definir cuál será la apariencia de la estructura del documento. Por ejemplo, mediante CSS se puede definir que todas las etiquetas <h1> (título importante) tengan asociada las características de presentación "color rojo" y "centrado". Se puede utilizar una única hoja de estilos en todo un sitio web para economizar código y facilitar los posibles cambios de presentación futuros.

Las páginas utilizadas tradicionalmente en los servidores web son **páginas estáticas**: el servidor ofrece siempre el mismo contenido cuando el usuario las solicita. Estas páginas suelen tener extensión .htm o .html. Sin embargo, debido a las necesidades crecientes de los sitios web, se han desarrollado lenguajes específicos para **páginas dinámicas**. Cuando el usuario solicita la página, el servidor interpreta el código fuente de la página, genera un código HTML y lo devuelve al usuario. Las páginas dinámicas se utilizan habitualmente para realizar consultas a bases de datos y ofrecer así resultados dinámicos al usuario.

2. Tecnologías de programación

Los principales lenguajes de páginas dinámicas son:

— **ASP** (*Active Server Pages*, páginas activas de servidor). Lenguaje propietario de Microsoft. Programación basada en Visual Basic.

— **PHP** (*Hypertext Preprocessor*, preprocesador de hipertexto). Lenguaje abierto con sintaxis similar a C.

— **JSP** (*JavaServer Pages*, páginas JAVA de servidor). Sintaxis basada en JAVA.

Otras tecnologías relacionadas con las páginas web que debemos conocer son:

— **DHTML** (*Dynamic HTML, HTML dinámico*). Permite modificar las propiedades de presentación de un documento después de haberse cargado en el navegador del usuario. Esto es lo que se conoce como

"dotar de movimiento" a una página. Mediante DHTML podríamos hacer que una imagen se desplazara continuamente por la pantalla. Esto es posible mediante un código JavaScript que modifique continuamente las coordenadas de la imagen establecidas inicialmente en la hoja de estilos del documento.

— **VRML** (*Virtual Reality Markup/Modeling Language*, lenguaje de marcas/modelado de realidad virtual). Ofrece al usuario la posibilidad de realizar acciones dentro de escenarios tridimensionales de realidad virtual.

— **SMIL** (*Syncronized Multimedia Integration Language*, lenguaje de integración de multimedia sincronizado). Desarrollado por W3C, permite crear presentaciones utilizando un conjunto de objetos multimedia independientes. El programador puede definir la situación de los objetos y su comportamiento temporal. Está basado en XML.

HTML es un lenguaje estático, no tiene estado. Esto significa que las páginas servidas al usuario siempre serán las mismas, no pudiéndose adaptar a las preferencias o decisiones que haya tomado el usuario en páginas visitadas anteriormente. Cada página HTML funciona como un programa independiente, por lo que no existe el concepto de variable como medio para pasar información entre las páginas de un mismo sitio web. En realidad, la única interacción posible que ofrece HTML es a través de los enlaces y los formularios. Pero observemos que los formularios de HTML solo recogen información del usuario, no la tratan. HTML no es suficiente para procesar la información que el usuario ha escrito en un formulario.

Para aumentar las posibilidades de HTML, se han desarrollado diferentes tecnologías de programación web. Estas tecnologías se pueden clasificar en función de dónde se ejecuten.

— **En el servidor**. El servidor web ejecuta el código de programación y genera HTML. El navegador del cliente recibe únicamente HTML, por lo que no llega a ver la programación del servidor.

— **En el cliente**. El servidor envía un código de programación al cliente, sin entender lo que envía. El navegador del cliente ejecutará el código.

Una tecnología que tiene parte de cliente y parte de servidor son las *cookies*. Se trata de archivos con información del usuario que almacena el servidor web en el ordenador del usuario. La información que se acostumbra a almacenar en las cookies es aquella propia de cada usuario (sus preferencias de navegación por el sitio web, el identificador de la sesión del usuario, los artículos que ha comprado en un comercio virtual, su nombre, etc.). Las cookies tienen una caducidad definida por el sitio web. Esta información la genera y la guarda el servidor web para posteriormente consultarla. Se utiliza habitualmente para pasar información de unas páginas a otras de un mismo sitio web. El navegador del usuario utiliza un directorio para el almacenamiento de las cookies de los distintos sitios web que lo requieren. Un sitio web no debería poder consultar una cookie que no ha introducido.

2.1. Tecnologías en el lado servidor

Se ejecutan en el servidor las páginas dinámicas (ASP, PHP, JSP), los CGIs y los servlets.

La tecnología **CGI** (*Common Gateway Interface,* interfaz común de pasarela) permite ejecutar programas externos en el servidor web. Se utilizó frecuentemente para el acceso a bases de datos, tratamiento de formularios, buscadores, etc. El principal lenguaje utilizado para el desarrollo de CGIs es Perl. Los CGIs son programas independientes que habitualmente se invocan desde un formulario HTML (atributo action). Su mayor problema es que son poco eficientes cuando reciben muchas peticiones simultáneas. Esto es debido a que cada vez que un usuario ejecuta un CGI, se abre una instancia nueva del programa en el servidor, con el consiguiente consumo de memoria y CPU. En su lugar, se prefiere en la actualidad la utilización de páginas dinámicas o servlets.

Las páginas dinámicas incluyen código de programación mezclado entre el código HTML. Estos códigos se conocen con el nombre de scripts. El servidor web comienza leyendo la página desde el principio. Cuando el código es HTML, se lo envía tal cual al cliente. Sin embargo, cuando el código es un script de servidor, lo ejecuta y genera un código HTML que envía al cliente. La página HTML que recibe el cliente ha sido, por tanto, construida dinámicamente (en tiempo de ejecución) según su petición. Las aplicaciones de las páginas dinámicas son las mismas que en los CGIs: acceso a bases de datos, formularios, etc.

Los principales lenguajes de programación de páginas dinámicas son: **ASP** (Active Server Pages, páginas activas de servidor) de Microsoft; **PHP** (*Hypertext Preprocessor*, preprocesador de hipertexto), que es un proyecto de Apache Software Foundation y cuya utilización es libre y gratuita; y **JSP** (*JavaServer Pages*, páginas de servidor de JAVA), que es un proyecto del mundo JAVA, liderado por Sun Microsystems. ASP es propio del servidor web de Microsoft, Internet Information Server (IIS). PHP se acostumbra a utilizar en servidores Apache (Linux, Unix, etc.). JSP también es frecuente en entornos Unix. Para delimitar códigos ASP y JSP se utilizan los símbolos <% y %>. En cambio, para delimitar porciones de código PHP, se requieren los símbolos <? y ?>.

Por último, los **servlets** son la respuesta de la tecnología JAVA a los CGIs. Al igual que los CGIs, son programas completos, sin embargo, son más eficientes, potentes y portables. El servidor mantiene una máquina virtual en ejecución. Cada vez que un usuario invoca un servlet, se genera un nuevo hilo en la máquina virtual (no se abre una nueva instancia de programa).

Para la ejecución tanto de servlets como de JSPs, se requiere un software específico JAVA en el servidor como, por ejemplo, Tomcat, de la Apache Software Foundation.

2.2. Tecnologías en el lado cliente

Se ejecutan en el navegador del usuario los lenguajes de scripting (JavaScript, VBScript, JScript...) y, en general, el resto de tecnologías no citadas más arriba como, por ejemplo, DHTML, CSS, Applets de JAVA, VRML o SMIL.

Entre las tecnologías de scripting de cliente incluimos JavaScript (también conocido como LiveScript) de Netscape, VBScript y JScript de Microsoft, y ECMAScript que es un lenguaje basado en JavaScript con soporte para el estándar ECMA-262. Entre todas ellas, JavaScript es la tecnología más utilizada.

Al igual que en las páginas dinámicas, el autor de la página incorpora porciones de código de programación (scripts) intercalados en el código HTML. Sin embargo, la diferencia del scripting del cliente es que este se ejecuta en el propio navegador, por lo que los códigos fuente los recibe el navegador donde son interpretados línea a línea.

Los applets de JAVA se ejecutan en la máquina virtual del cliente. Se requiere que el navegador tenga instalada una máquina virtual de JAVA. Son programas completos que se traen compilados (archivos .class o bytecodes) desde el servidor web y se interpretan en el cliente.

2.3. Tecnologías JAVA

Resumimos las tecnologías web JAVA:

— En el servidor: JSP y Servlets.

— En el cliente: Applets, JavaScript y JScript.

— Scripts: JSP, JavaScript y JScript.

— Programas completos: Servlets, Servicios Web.

JavaScript en ocasiones se compara con otras tecnologías JAVA. Sin embargo, quizás su único parecido es la sintaxis del lenguaje. JavaScript es interpretado y JAVA, compilado (el código compilado luego se interpreta en la máquina virtual). JAVA es un lenguaje de propósito general y JavaScript únicamente funciona en un navegador web. JAVA, a diferencia de JavaScript, es orientado a objetos (incluye herencia, polimorfismo y encapsulación). JavaScript tiene muchas limitaciones, por lo que en la práctica se utiliza casi exclusivamente para validación de formularios y mejoras en la presentación visual de las páginas web.

Por último y para completar el listado de tecnologías JAVA, no nos olvidamos de JavaBeans. Se trata de una tecnología de componentes. Sus ventajas son la portabilidad, independencia de plataforma y reutilización de componentes. Mediante JavaBeans se pueden construir aplicaciones reutilizando componentes previamente escritos.

3. JAVA Script

Al igual que en VBScript puede ser utilizado dentro de una página Web utilizando el siguiente comando HTML.

- **El comando <SCRIPT>**

El código en JavaScript se escribe dentro del par <SCRIPT> como muestra el ejemplo.

```
<SCRIPT LANGUAGE="JavaScript">
<!--
  function MayorQue2(Dt){
    return Dt > 2;
  }
-->
</SCRIPT>
```

El atributo LANGUAGE indica al navegador el lenguaje en el que se ha escrito el script. No es necesario especificar el lenguaje porque es el lenguaje de script que los navegadores utilizarán por defecto. Todo indicado anteriormente para VBScript es también aplicable a JavaScript.

3.1. Tipos de datos en JavaScript

JavaScript es un lenguaje poco tipado porque posee pocos tipos de datos. Al declarar una variable no pertenece a ningún tipo concreto de datos, tiene un valor Unsigned. Al asignarle un valor pasará de un tipo de datos a otro dependiendo del valor que le asignemos. En función del contenido que tiene la variable, JavaScript asume que es numérico, cadena, booleano, objeto o nulo.

La siguiente tabla muestra los 5 diferentes tipos de datos de JavaScript.

Tipos	Descripción
Null	No contiene un dato válido o no inicializada.
Booleano	Contiene true (verdadero) o false (falso).
Numérico	Admite cualquier tipo de dato numérico, entero, coma flotante, positivo, negativo, etc.
String	Contiene una cadena de 2 millones de caracteres.
Object	Contiene un object.

Se pueden usar funciones de conversión o conversiones implícitas para pasar de un tipo a otro y la función typeOf() para conocer el tipo actual de una variable.

3.1.1. Funciones de conversión (conversión explícita de tipos)

parseFloat(..)	**Descripción**
	Convierte un String en un número en coma flotante si es posible.
	Sintaxis
	parseFloat(string)
	El string ha de ser una cadena válida o una cadena que posea un número válido, sino devolverá NaN.
	Notas
	Ejemplo:
	var v1= parseFloat("123.456");
	var v2=parseFloat("123ABC");
	var v3=parseFloat("ABC");
	alert(v1); // correcto. Devuelve 123.456
	alert(v2); // detecta un error en la primera letra. Devuelve 123. Solamente admite números del 0 al 9, el + y el -, el punto como separador decimal y la letra "e" para el exponencial de 10.
	alert(v3); // Incorrecto. Devuelve NaN.

parseInt (...)	**Descripción**
	Convierte un String en un número entero de cualquier base entre 2 y 36.
	Sintaxis
	parseInt(string, Base)
	Si no se especifica la base por defecto es base decimal.
	Notas
	Ejemplo:
	var v1= parseInt("123");
	var v2=parseInt("ABC",16);
	alert(v1); // Devuelve 123
	alert(v2); // Devuelve 2748 que es ABC en base Hexadecimal.

toString()	**Descripción** Retorna una cadena con la información de cualquier tipo de dato. **Sintaxis** **variable.toString()** **Notas** Esta función es llamada automáticamente cuando concatenamos una variable a una cadena.

3.1.2. La Función typeOf

Retorna una cadena que identifica el tipo de una variable. Su sintaxis es

typeOf(varname)

Algunos ejemplos son:

```
var check,saludo=”Hola”;
check = typeOf( saludo);
alert( check);    //retorna string
saludo=true;
check = typeOf( saludo);
alert( check);    //retorna boolean
```

3.2. Variables en JavaScript

Una variable es un espacio reservado en la memoria que es utilizado para guardar información durante la ejecución de un script. Una variable es referenciada por su nombre. Pueden declararse variables de un modo explícito utilizando la instrucción var o implícitamente solo con escribir su nombre y asignarle un valor:

```
var GradosFahrenheit; // Declaración explícita
GradosCentigrados=23;  //Declaración implícita
```

Las variables pueden ser declaradas en ámbito global fuera de cualquier función, o en ámbito local, dentro de una función. JavaScript es un lenguaje de programación que hace distinción entre Mayúsculas y Minúsculas.

Pueden declarase varias variables separándolas por comas (ejemplo: var Top, Bottom, Left, Right=23; //) e incluso asignarles valor a la hora de definirlas:

3.2.1. Restricciones en el nombre de las variables

— Deben comenzar con una letra.

— No pueden usar un punto.

— No pueden exceder 255 caracteres.

— No puede volver a utilizarse dentro del mismo ámbito.

3.2.2. Ámbito de las variables

El ámbito de una variable determina la parte del código desde el cual podremos acceder a la misma. Pueden declararse variables en dos ámbitos distintos:

— **Local**.

— **Global**.

Una variable **global** es aquella que se define fuera de cualquier función que compongan el script. Son accesibles desde cualquier parte del código. El tiempo de vida de estas variables es también global, se crean al cargar el script y se destruyen al descargar la página Web.

Una variable **local** a una rutina es aquella que es definida dentro de una función. Solo es accesible desde la función en la que han sido definida. El tiempo de vida de estas variables es local, se crean al cargar la función y se destruyen al finalizar el código de la misma.

3.2.3. Asignar valores a las variables

Mediante una operación de asignación. La variable siempre está definida en la parte izquierda y el valor asignado está en la parte derecha.

```
B = 200;
```

3.2.4. Variables array

Las matrices o arrays poseen tres sintaxis diferentes. Se puede declarar un array, indicar su tamaño o sus elementos. Las matrices en JavaScript se crean gracias a un objeto llamado Array para lo cual utilizaremos el operador **new**:

```
var proveedores = new Array();
```

A continuación asignaremos valores en las posiciones que queramos:

```
proveedores[0]="Arrakis";

proveedores[1]="Teleline";

proveedores[2]="CTV";
```

Las matrices comienzan a contar los elementos de la misma por 0.

El segundo sistema me permite especificar inicialmente el número de elementos de la matriz.

```
var proveedores = new Array(3);
```

El tercer sistema permite incluir los elementos en la matriz a la vez que se declara:

var proveedores = new Array("Arrakis","Teleline","CTV");

En último lugar podemos especificar el número de dimensiones de la matriz:

var proveedores = new Array(3)(3);

3.3. Operadores en JavaScript

JavaScript tiene un gran rango de operadores, incluyendo los aritméticos, de comparación, de asignación, lógicos y de concatenación.

- **Prioridad de operadores**

Cuando varios operadores son utilizados en la misma expresión, se utiliza la prioridad para resolverla. Podemos utilizar paréntesis para cambiar la prioridad y que unas partes de la expresión sean evaluadas antes que otras. Utilizando paréntesis las operaciones son ejecutadas resolviéndose de dentro hacia fuera y dentro de los paréntesis con la prioridad estándar.

Los operadores son evaluados en el siguiente orden:

Operadores	Precedencia
. () []	1
++ — - ¡ typeOf new delete void	2
* / %	3
+ -	4
<< >>	5
<<= >>=	6
>> ¡=	7
&	8
^	9
|	10
&&	11
||	12
?:	13
Asignación	14

Aritméticos		Comparaciones		Lógicos		Asignación	
Descripción	Símbolo	Descripción	Símbolo	Descripción	Símbolo	Descripción	Símbolo
Cambio de signo	-	Igualdad	==	Negación	!	Incremento	+=
Incremento	++	Distinto	!=	Y lógica	&&	Decremento	-=
Decremento	--	Menor que	<	O Lógica	||	Multiplicación	*=
Multiplicación y división	* , /	Mayor que	>			División	/=
División entera	\	Menor o igual que	<=			Módulo	%=
Suma y resta	+ , -	Mayor o igual que	>=				
Resto de división	%						

Cuando la multiplicación y la división, o la suma y la resta, aparecen juntas, son evaluadas de izquierda a derecha. La concatenación de cadenas no es un operador aritmético, pero la prioridad hace que se ejecute después de los operadores aritméticos y antes que los de comparación.

- **Operadores de comparación**

  Comparan expresiones. Su sintaxis es:

  resultado = expression1 comparisonoperador expression2

Operador	Descripción	Es true si	Es false si
<	Menor que	Expression1 < expression2	expression1 >= expression2
<=	Menor o igual que	Expression1 <= expression2	expression1 > expression2
>	Mayor que	Expression1 > expression2	expression1 <= expression2
>=	Mayor o igual que	Expression1 >= expression2	expression1 < expression2
==	Igual que	Expression1 = expression2	expression1 <> expression2
¡=	distinto	Expression1 <> expression2	expression1 = expression2

- **Operadores de concatenación**

operador + Suma dos números. Si uno de ellos o los dos son cadenas también concatena. Su sintaxis es:

result = expression1 + expression2;

- **Operadores lógicos**

operador && Conjunción lógica entre dos expresiones. Su sintaxis es:

resultado = expression1 && expression2;

operador ¡ Negación lógica de una expresión. Su sintaxis es:

result = ¡ expression;

operador || Disyunción lógica de dos expresiones. Su sintaxis es:

result = expression1 || expression2;

- **Operadores de asignación**

```
num += 2;  // num = num + 2
num -= 2;  // num = num - 2
num *= 2;  // num = num * 2
num /= 2;  // num = num / 2
num %= 2;  // num = num % 2
```

3.4. Usando condiciones

- **Controlando la ejecución del programa**

Las decisiones nos permiten ejecutar o no un conjunto de instrucciones dependiendo de si la condición es verdadera (true) o falsa (false). Existen dos instrucciones para poder hacerlo en JavaScript:

— If-else

— switch-case

- **Tomando decisiones usando if...else**

La instrucción "if...else" es utilizada para evaluar cuando una condición es cierta o falsa y dependiendo del resultado poder especificar un conjunto de

instrucciones a ejecutar. Usualmente la condición utiliza una operación de comparación. Las instrucciones de comparación pueden ser anidadas.

- **Ejecutar comando si la condición es true**

El siguiente ejemplo muestra una condición que solamente ejecuta instrucciones cuando la condición es cierta. No se escribe la parte del else.

```
function menorde5( aux){
   var num=aux;
   if( num < 5)
      return "Es menor que 5";
}
```

Si se quieren ejecutar más de una instrucción, han de incluirse dentro de un bloque con llaves

```
function AlertUser(value){
   if (value == 0){
      accion1;
      accion2;
      accion3;
   }
}
```

- **Ejecutando ciertas acciones si la condición es true y otras si es false**

Se puede usar "if...Else" para definir dos bloques de ejecución para los casos en que la condición sea cierta o falsa.

```
function AlertUser(value){
   if (value == 0){
      accion1;
      accion2;
      accion3;
```

```
    }
    else{
        accion4;
        accion5;
        accion6;
    }
}
```

- **Decidiendo entre varias alternativas**

Switch-case trabaja con una expresión que es evaluada una sola vez pero que luego es comparada en cada una de las instrucciones case de la estructura:

```
switch ( NotaExamen){
    case <5:
        alert("Suspenso");
        break;
    case<7:
        alert( "Aprobado");
        break;
    case 8,9:
        alert("Notable");
        break;
    case <=10:
        alert("Sobresaliente");
    default:
        alert("Nota no válida");
}
```

Default permite la ejecución de código en el caso de que ninguno de los case sea cierto.

Cada case ha de finalizar en un break para no pasar a ejecutar el código del siguiente case.

Los case se ejecutan en el orden indicado, del primero al último.

Las condiciones de los case indican igualdad (case 1), igualdad con varios números (case 3,4)

3.5. Bucles

Se usan bucles para repetir código.

Los bucles nos permiten ejecutar un conjunto de instrucciones repetidas veces. Uno de ellos nos permite ejecutar el bucle mientras que la condición sea falsa, otro hasta que sea cierta y otro un número determinado de veces.

Existen los siguientes bucles en JavaScript:

— **do-while:** se repite hasta que una condición sea falsa.

— **while:** se repite mientras que la condición sea cierta.

— **for:** usa un contador para repetirse un número determinado de veces.

- **Usando do-while**

Puede usar do-while para ejecutar un bloque un número indeterminado de veces. Se repite hasta que una condición sea falsa.

- **Repitiendo hasta que una condición sea falsa**

Usando while en el chequeo de la condición de do-while permite ejecutar una acción hasta que la condición sea falsa. El bucle evalúa la condición al final por lo que se ejecutará al menos una vez.

```
var contador=0;
do{
  alert( contador);
  contador++;
}while(contador<10);
```

- **Repetir un bucle mientras que la condición sea cierta**

While permite repetir mientras que la condición evaluada al principio sea cierta. Al evaluar la condición al principio puede no ejecutarse el bucle ninguna vez.

```
var contador=0;
while( contador<10){
 alert( contador);
   contador++;
}
```

- **Usando for**

Puede usar for para ejecutar un bucle un número determinado de veces. Utiliza una variable llamada contador cuyo valor se incrementa o decrementa en cada repetición del bucle y que marca el número exacto de veces que se ejecutará. Su sintaxis es:

```
for( inicialización; condición; incremento){}
```

El siguiente ejemplo repite el proceso 50 veces para los valores de x desde 1 a 50, incrementando x en +1 en cada paso del bucle.

```
var x
 for( x = 1;x<=50;x++){
    accion;
 }
```

El incremento puede ser positivo o negativo. Si es positivo incrementa y si es negativo decrementa. En el siguiente ejemplo "j" comienza valiendo 2 y pasa a valer 4, 6, 8 y 10 en cada paso del bucle.

```
for( j = 2;j<=10;j+=2) { ... }
```

El siguiente ejemplo tendrá para x los valores 16, 14, 12, 10, 8, 6, 4 y 2.

```
for( j = 16;j>=2;j-=2) { ... }
```

Puede utilizar break para finalizar el bucle en cualquier momento.

Puede utilizar continue para pasar a la siguiente iteración del bucle sin finalizar la actual.

3.6. Rutinas en JavaScript

En JavaScript solo existe la rutina function.

- **Function (Función)**

Function es un conjunto de instrucciones de Java Script. Esta función puede retornar un valor asignándolo al final a la palabra reservada return. Su sintaxis es:

```
function nombre_función (parámetros) {...}
```

Un ejemplo de aplicación es:

```
function multiplicador (p1, p2) {
        return p1 * p2;   //Devuelve el resultado de la multiplicación
}
```

Todo el código ejecutable debe ir situado dentro de las llaves. La instrucción return permite la finalización de la función. Puede estar situada en cualquier parte de una rutina. La finalización de una función devuelve el control a aquella que la llamó. Las funciones sí pueden ser utilizadas dentro de una expresión.

Dentro de una función pueden existir dos tipos de variables, aquellas que son definidas explícitamente en la función y aquellas que no. Las primeras son locales. Se crean al llamar a la función y se destruyen cuando finaliza la misma. No pueden ser utilizadas fuera de ella. Las segundas son las variables globales, que son comunes a todo el script. Permiten compartir información entre las funciones. Son accesibles desde cualquier función y son destruidas al final del script.

- **Utilizando funciones en código**

Una función siempre debe ser utilizada en la parte derecha de una asignación o en una expresión:

```
Temp = Celsius(fDegrees);
alert "The Celsius temperature is " + Celsius(fDegrees) + " degrees.";
```

4. Applets

4.1. ¿Qué es un applet?

Un applet es una mini-aplicación escrita en JAVA que se ejecuta en un browser (Netscape Navigator, Microsoft Internet Explorer...) al cargar una página HTML que incluye información sobre el applet a ejecutar por medio de los tags <APPLET>... </APPLET>.

Los ficheros de JAVA compilados (*.class) se descargan a través de la red desde un servidor web o servidor HTTP hasta el browser en cuya Máquina Virtual Java se ejecutan. Pueden incluir también ficheros de imágenes y sonido.

Aunque su entorno de ejecución es un browser, las applets se pueden probar sin necesidad de browser con la aplicación appletviewer del JDK de Sun.

4.2. Características de las applets

— Las applets no tienen un método main() con el que comience la ejecución. El papel central de su ejecución lo asumen otros métodos que se verán posteriormente.

— Todas las applets derivan de la clase java.applet.Applet. Las applets deben redefinir ciertos métodos heredados que controlan su ejecución: init(), start(), stop(), destroy().

— Se heredan otros muchos métodos de las super-clases de applet que tienen que ver con la generación de interfaces gráficas de usuario (AWT). Así, los métodos gráficos se heredan de Component, mientras que la capacidad de añadir componentes de interface de usuario se hereda de Container y de Panel.

• Las applets también suelen redefinir ciertos métodos gráficos: los más importantes son paint() y update(), heredados de Component y de Container; y repaint() heredado de Component.

• Las applets disponen de métodos relacionados con la obtención de información, como, por ejemplo: getAppletInfo(), getAppletContext(), getParameterInfo(), getParameter(), getCodeBase(), getDocumentBase(), e isActive().

4.3. Métodos de control

• **Método init()**

Se llama automáticamente al método init() en cuanto el browser o visualizador carga el applet. Este método se ocupa de todas las tareas de inicialización, realizando las funciones del constructor (al que el browser no llama).

• **Método start()**

El método start() se llama automáticamente en cuanto la applet se hace visible, después de haber sido inicializada. Se llama también cada vez que la applet se hace de nuevo visible después de haber estado oculta (por dejar de estar activa esa página del browser, al cambiar el tamaño de la ventana del browser, al hacer reload, etc.).

Es habitual crear threads en este método para aquellas tareas que, por el tiempo que requieren, dejarían sin recursos al Applet o incluso al browser. Las animaciones y ciertas tareas a través de Internet son ejemplos de este tipo.

• **Método stop()**

El método stop() se llama de forma automática al ocultar el applet (por haber dejado de estar activa la página del browser, por hacer reload o resize,

etc.). Con objeto de no consumir recursos inútilmente, en este método se suelen parar las threads que estén corriendo en la applet, por ejemplo para mostrar animaciones.

- **Método destroy()**

Se llama a este método cuando la applet va a ser descargada para liberar los recursos que tenga reservados (excepto la memoria). De ordinario no es necesario redefinir este método, pues el que se hereda cumple bien con esta misión.

4.4. Métodos para dibujar la applet

Las applets son aplicaciones gráficas que aparecen en una zona de la ventana del browser. Por ello, deben redefinir los métodos gráficos paint() y update(). El método paint() se declara en la forma:

public void paint(Graphics g)

El objeto gráfico "g" pertenece a la clase java.awt.Graphics, que siempre debe ser importada por la applet. Este objeto define un contexto o estado gráfico para dibujar (métodos gráficos, colores, fonts, etc.) y es creado por el browser.

Todo el trabajo gráfico del Applet (dibujo de líneas, formas gráficas, texto, etc.) se debe incluir en el método paint(), porque este método es llamado cuando la applet se dibuja por primera vez y también de forma automática cada vez que la applet se debe redibujar.

En general, el programador crea el método paint() pero no lo suele llamar. Para pedir explícitamente al sistema que vuelva a dibujar la applet (por ejemplo, por haber realizado algún cambio) se utiliza el método repaint(), que es más fácil de usar, pues no requiere argumentos. El método repaint() se encarga de llamar a paint() a través de update().

El método repaint() llama a update(), que borra todo pintando de nuevo con el color de fondo y luego llama a paint(). A veces esto produce parpadeo de pantalla o flickering. Existen dos formas de evitar el flickering:

- Redefinir update() de forma que no borre toda la ventana sino solo lo necesario.
- Redefinir paint() y update() para utilizar doble buffer.

4.5. Inclusión de applets en páginas HTML

Para llamar a un applet desde una página HTML se utiliza la tag doble <APPLET>...</APPLET>, cuya forma general es (los elementos opcionales aparecen entre corchetes[]):

<APPLET CODE=»miApplet.class» [CODEBASE=»unURL»] [NAME=»unName»]
WIDTH=»wpixels» HEIGHT=»hpixels»
[ALT=»TextoAlternativo»]>
[texto alternativo para browsers que reconocen el tag <applet> pero no pueden ejecutar el applet]
[<PARAM NAME=»MyName1» VALUE=»valueOfMyName1»>]
[<PARAM NAME=»MyName2» VALUE=»valueOfMyName2»>]
</APPLET>

El atributo NAME permite dar un nombre opcional al applet, con objeto de poder comunicarse con otras applets o con otros elementos que se estén ejecutando en la misma página. El atributo ARCHIVE permite indicar uno o varios ficheros Jar o Zip (separados por comas) donde se deben buscar las clases.

A continuación se señalan otros posibles atributos de <APPLET>:

- ARCHIVE=»file1, file2, file3». Se utiliza para especificar ficheros JAR y ZIP.
- ALIGN, VSPACE, HSPACE. Tienen el mismo significado que el tag IMG de HTML.

4.6. Paso de parámetros a una applet

Los tags PARAM permiten pasar diversos parámetros desde el fichero HTML al programa JAVA del applet, de una forma análoga a la que se utiliza para pasar argumentos a main().

Cada parámetro tiene un nombre y un valor. Ambos se dan en forma de String, aunque el valor sea numérico. El applet recupera estos parámetros y, si es necesario, convierte los Strings en valores numéricos. El valor de los parámetros se obtiene con el siguiente método de la clase applet:

String getParameter(String name)

La conversión de Strings a los tipos primitivos se puede hacer con los métodos asociados a los wrappers que JAVA proporciona para dichos tipo fundamentales (Integer.parseInt(String), Double.valueOf(String), ...).

En los nombres de los parámetros no se distingue entre mayúsculas y minúsculas, pero sí en los valores, ya que serán interpretados por un programa JAVA, que sí distingue.

El programador del Applet debería prever siempre unos valores por defecto para los parámetros del applet, para el caso de que en la página HTML que llama al applet no se definan.

El método getParameterInfo() devuelve una matriz de Strings (String[][]) con información sobre cada uno de los parámetros soportados por el applet: nombre, tipo y descripción, cada uno de ellos en un String. Este

método debe ser redefinido por el programador y utilizado por la persona que prepara la página HTML que llama al applet. En muchas ocasiones serán personas distintas, y esta es una forma de que el programador del applet de información al usuario.

4.7. Parametrizando una applet

Vamos a aprovechar este ejemplo, modificándolo un poco para indicarle desde el HTML qué archivos debe cargar, mediante parámetros. Nuestro HTML modificado será:

```
<HTML>
   <HEAD>
      <TITLE>Lolo 24 - Multimedia</TITLE>
   </HEAD>
   <BODY>
      <applet code=»Lolo24.class» width=150 height=200>
         <param name=»imagen» value=»javacero.gif»>
         <param name=»sonido» value=»tada.au»>
      </applet>
   </BODY>
</HTML>
```

Para leer estos parámetros desde la applet, usamos el método getParameter(nombre Parámetro), así que podemos modificar nuestra applet simplemente modificando un par de líneas:

archImagen = getParameter(«imagen»);

archAudio = getParameter(«sonido»);

Con esto hemos visto una gran parte de lo que es JAVA. No hemos profundizado demasiado en cada punto, pero hemos hecho ejemplos que funcionan para ilustrar cada cosa. Sin embargo, hemos dejado un punto importante y muy fuerte de JAVA, que es el de las comunicaciones entre aplicaciones y, especialmente, el uso de sockets y la programación de aplicaciones cliente-servidor.

Por cuestiones de seguridad, los applets son más limitadas que las aplicaciones JAVA locales. Las políticas de seguridad las manejan los browsers (no JAVA), y generalmente los límites que se imponen a las applets son:

— Un applet no puede cargar bibliotecas (libraries) ni definir métodos nativos.

— No puede leer o escribir normalmente archivos en el cliente que lo carga desde otro server.

— No puede establecer conexiones de red, salvo al servidor del que proviene.

— No puede arrancar programas en la máquina donde se está ejecutando.

— No puede leer ciertas propiedades del sistema.

— En las ventanas de las applets se indica que se trata de una applet.

Sin embargo, pueden:

— Reproducir sonidos.

— Pueden establecer conexiones con el servidor del que provienen.

— Pueden llamar fácilmente páginas HTML desde el browser.

— Pueden invocar métodos públicos de otros Applets de la misma página.

— Si se cargan desde la propia máquina (localmente) no tienen ninguna de las restricciones anteriores.

— Pueden seguir corriendo aunque se cambie de página en el browser.

En realidad, la especificación de JAVA permite que las applets lean archivos en otras máquinas dando la URL completa; sin embargo, los browsers no lo permiten. Veremos más adelante cómo intercambiar datos entre máquinas para poder ver un archivo del server, por ejemplo.

4.8. Sonidos en la applet

La clase applet y la interface AudioClips permiten utilizar sonidos en applets. Respecto a la carga de sonidos, por lo general es mejor cargar los sonidos en un thread distinto (creado en el método init()) que en el propio método init(), que tardaría en devolver el control y permitir al usuario empezar a interaccionar con el applet. Si el sonido no ha terminado de cargarse (en la thread especial para ello) y el usuario interacciona con la applet para ejecutarlo, la applet puede dar un aviso de que no se ha terminado de cargar.

4.9. Imágenes en la applet

Las applets admiten los formatos JPEG y GIF para representar imágenes a partir de ficheros localizados en el servidor. Estas imágenes se pueden cargar con el método getImage() de la clase applet, que puede tener las formas siguientes:

public Image getImage(URL url)

public Image getImage(URL url, String name)

Estos métodos devuelven el control inmediatamente. Las imágenes de cargan cuando se da la orden de dibujar las imágenes en la pantalla. El dibujo se realiza entonces de forma incremental, a medida que el contenido va llegando.

Para dibujar imágenes se utiliza el método drawImage() de la clase Graphics, que tiene las formas siguientes:

public abstract boolean drawImage(Image img, int x, int y,
Color bgcolor, ImageObserver observer)
public abstract boolean drawImage(Image img, int x, int y, int width, int height,
Color bgcolor, ImageObserver observer)

El primero de ellos dibuja la imagen con su tamaño natural, mientras que el segundo realiza un cambio en la escala de la imagen.

Los métodos drawImage() van dibujando la parte de la imagen que ha llegado, con su tamaño, a partir de las coordenadas (x, y) indicadas, utilizando bgcolor para los píxeles transparentes.

Estos métodos devuelven el control inmediatamente, aunque la imagen no esté del todo cargada. En este caso devuelve false. En cuanto se carga una parte adicional de la imagen, el proceso que realiza el dibujo avisa al ImageObserver especificado. ImageObserver es una interface implementada por applet que permite seguir el proceso de carga de una imagen.

5. Servlets

5.1. Introduccion

Los servlets son clases JAVA que amplian la funcionalidad de un servidor web mediante la generacion dinamica de paginas web. El motor de servlets administra la carga y descarga del servlet dirigiendo las peticiones a los mismos y enviando las respuestas a los clientes.

5.2. Ventajas a la hora de trabajar con los servlets

— Mejor rendimiento que otras tecnologias anteriores. El motor del servlet carga un solo ejemplar o instancia de la clase Servlet y le lanza peticiones a través de una serie de hilos (threads).

— Dados los problemas de compatibilidad de las applets JAVA debidos a su ejecucion del lado del cliente, los servlets se ejecutan en una maquina virtual en entorno servidor controlado y solo necesitan el protocolo http para su comunicación. Por tanto el cliente no necesita de ningun software adicional.

— Con los servlets podemos recordar detalles de peticiones previas del mismo cliente. A traves de la clase HttpSesion.

— El codigo en el cual se escriben los servlets es JAVA, con lo cual podemos disfrutar de toda su potencia asi como el acceso a datos dentro del servidor, restricciones, gesion de varios subprocesos, etc.

5.3. Ciclo de vida de un servlet

Cada servlet tiene el mismo ciclo de vida:

— Carga un servlet cuando lo solicitamos por primera vez. El servidor no puede recargar un servlet sin primero haber destruido el primero llamando al método destroy.

— Inicialización del servlet (init). Una vez que el servidor carga un servlet, ejecuta el método init del servlet. La inicialización se completa antes de manejar peticiones de clientes y antes de que el servlet sea destruido. Aunque muchos servlets se ejecutan en servidores multi-thread, no tienen problemas de concurrencia durante su inicialización. El servidor llama solo una vez al método init al crear la instancia del servlet, y no lo llamará de nuevo a menos que vuelva a recargar el servlet.

— Manejo de peticiones de cliente (service). Después de la inicialización, el servlet puede manejar peticiones de clientes. Estas respuestas son manejadas por la misma instancia del servlet por lo que hay que tener cuidado con acceso a variables compartidas por posibles problemas de sincronización entre requerimientos concurrentes.

— Eliminación del servlet (destroy). Los servlets se ejecutan hasta que el servidor los destruye, por cierre el servidor o bien a petición del administrador del sistema. Cuando un servidor destruye un servlet, ejecuta el método destroy del propio servlet. Este método solo se ejecuta una vez y puede ser llamado cuando aún queden respuestas en proceso por lo que hay que tener la atención de esperarlas. El servidor no ejecutará de nuevo el servlet hasta haberlo cargado e inicializado de nuevo.

5.4. Ejemplo de servlet

```
public class TAIServlet extends HttpServlet    {
    /**
     * Una sencilla página Web.
     */
    public void doGet (HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException
    {
    PrintWriter out;
    String          title = "Ejemplo de Servlet";
            response.setContentType("text/html");
```

```
        out = response.getWriter();
                out.println("<HTML><HEAD><TITLE>");
        out.println(title);
        out.println("</TITLE></HEAD><BODY>");
        out.println("<H1>" + title + "</H1>");
        out.println("<P>Ejemplo de Servlet.");
        out.println("<P>Para que este año apruebe fijo.");
        out.println("</BODY></HTML>");
        out.close();
       }
    }
```

5.5. Clases del servlet

Los servlets operan con un ciclo de vida fijo, que proporciona métodos de retrollamada a un motor de servlets para que se inicialicen, manejen las peticiones y los destruyan. La API proporciona dos modelos de subproceso: uno predeterminado consistente en subprocesos múltiples ejecutados en un solo ejemplar y el modelo de subproceso único alternativo.

Las clases e interfaces principales de la API servlet son:

— Servlet, interfaz que describe los métodos de retrollamada que se deben implementar.

— GenericServlet, clase que implementa los métodos de la interfaz Servlet.

— HttpServlet, clase específica para http de GenericServlet.

— ServletRequest, clase que encapsula la información sobre la petición del cliente.

— ServletResponse, clase que proporciona acceso a un flujo de salida para los resultados.

— ServletContext, interfaz que permite a un grupo de servlets interoperar entre si en una aplicación web.

5.6. Servlets y JSP

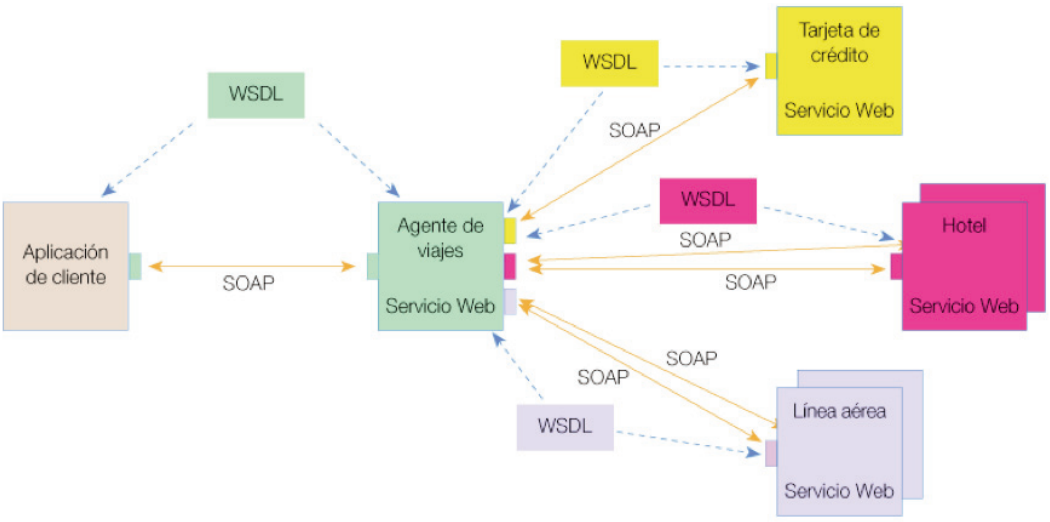
Una página JSP (JavaServer Pages) se ejecta en un componente del servidor llamado contenedor de JSP que las traduce a "servlets" JAVA equivalentes. Es decir, las paginas JSP son traducidas por este contenedor a su sintaxis Servlet correspondiente, teniendo las mismas ventajas. Quizás cabría destacar que las páginas JSP suelen ser mas sencillas de programar por parte de los desarrolladores, pero se obtiene menos potencia.

6. Servicios web

6.1. Introducción

Existen múltiples definiciones sobre lo que son los servicios web, lo que muestra su complejidad a la hora de dar una adecuada definición que englobe todo lo que son e implican. Una posible sería hablar de ellos como un conjunto de aplicaciones o de tecnologías con capacidad para interoperar en la web. Estas aplicaciones o tecnologías intercambian datos entre sí con el objetivo de ofrecer unos servicios. Los proveedores ofrecen sus servicios como procedimientos remotos y los usuarios solicitan un servicio llamando a estos procedimientos a través de la web.

Estos servicios proporcionan mecanismos de comunicación estándares entre diferentes aplicaciones, que interactúan entre sí para presentar información dinámica al usuario. Para proporcionar interoperabilidad y extensibilidad entre estas aplicaciones, y que al mismo tiempo sea posible su combinación para realizar operaciones complejas, es necesaria una arquitectura de referencia estándar.



6.2. Los servicios web

Un usuario (cliente dentro de los servicios web), a través de una aplicación, solicita información sobre un viaje que desea realizar haciendo una petición a una agencia de viajes que ofrece sus **servicios** a través de Internet. La agencia de viajes ofrecerá a su cliente (usuario) la información requerida. Para proporcionar al cliente la información que necesita, esta agencia de viajes solicita a su vez información a otros recursos (otros servicios web) en relación con el hotel y la línea aérea. La agencia de viajes obtendrá información de estos recursos, lo que la convierte a su vez en cliente de esos otros servicios web que le van a proporcionar la información solicitada sobre el hotel y la línea aérea. Por último, el usuario realizará el pago del viaje a través de la agencia de viajes que servirá de intermediario entre el usuario y el servicio web que gestionará el pago.

Toda la información está disponible para cualquier persona, en cualquier lugar, a través de cualquier dispositivo. Se busca un lenguaje común de intercambio de información aprovechando los estándares existentes en el mercado. Bajo este contexto nacen los servicios web basados en XML.

Estos servicios son componentes software que permiten a los usuarios usar aplicaciones de negocio que comparten datos con otros programas modulares, vía Internet. Son aplicaciones independientes de la plataforma que pueden ser fácilmente publicadas, localizadas e invocadas mediante protocolos web estándar, como XML, SOAP, UDDI o WSDL. El objetivo final es la creación de un directorio online de servicios, que pueda ser localizado de un modo sencillo y que tenga una alta fiabilidad.

La funcionalidad de los protocolos empleados es la siguiente:

— XML (eXtensible Markup Language): un servicio web es una aplicación web creada en XML.

— WSDL (Web Services Definition Language): este protocolo se encarga de describir el servicio web cuando es publicado. Es el lenguaje XML que los proveedores emplean para describir sus “Web Services”.

— SOAP (Simple Object Access Protocol): permite que programas que corren en diferentes sistemas operativos se comuniquen. La comunicación entre las diferentes entidades se realiza mediante mensajes que son rutados en un sobre SOAP.

— UDDI (Universal Description Discovery and Integration): este protocolo permite la publicación y localización de los servicios. Los directorios UDDI actúan como una guía telefónica de los “Web Services”.

Aunque la idea de la programación modular no es nueva, el éxito de esta tecnología reside en que se basa en estándares conocidos en los que ya se tiene una gran confianza, como el XML. Además, el uso de los servicios web aporta ventajas significativas a las empresas. El principal objetivo que se logra, es la interoperabilidad y la integración. Mediante estos servicios, las empresas pueden compartir servi-

cios software con sus clientes y sus socios de negocio. Esto ayudará a las compañías a escalar en sus negocios, reduciendo el coste en desarrollo y mantenimiento de software, y sacando los productos al mercado con mayor rapidez. La integración de aplicaciones hará posible obtener la información demandada en tiempo real, acelerando el proceso de toma de decisiones.

6.3. Seguridad

Actualmente, los servicios web están siendo ampliamente aceptados por las empresas para el desarrollo de software de uso interno. De este modo, los servicios pueden implementar toda su funcionalidad y permanecer seguros tras el cortafuegos de la compañía. Los desarrollos actuales no ayudan a la cooperación entre las empresas ya que no hay ningún estándar establecido sobre las técnicas de seguridad. Debido a la tecnología que es usada por los servicios web y, en concreto, al uso de SOAP, las técnicas de seguridad convencionales que se han venido usando en Internet, ya no son suficientes. Con SOAP, cada mensaje simple que se intercambia realiza múltiples saltos y es rutado a través de numerosos puntos antes de que alcance su destino final. Es por ello que los servicios web necesitan tecnologías que protejan los mensajes desde el principio hasta el final. Existen un conjunto de técnicas que se pueden usar para garantizar la seguridad a nivel de mensaje. Estas son:

— Encriptación XML: evita que los datos se vean expuestos a lo largo de su recorrido.

— Firma Digital XML: asocia los datos del mensaje al usuario que emite la firma, de modo que este usuario es el único que puede modificar dichos datos.

— XKMS y los certificados: XKMS (XML Key Management Specification) define servicios web que se pueden usar para chequear la confianza de un certificado de usuario.

— SAML y la autorización: SAML (Security Assertion Mark-up Language) hace posible que los servicios web intercambien información de autentificación y autorización entre ellos, de modo que un servicio web confíe en un usuario autentificado por otro servicio web.

— Validación de datos: permite que los servicios web reciban datos dentro de los rangos esperados.

— Además, también hay técnicas que permiten mantener la seguridad a otros niveles. La seguridad en UDDI permite autentificar todas las entidades que toman parte en la publicación de un servicio web: proveedor, agente y consumidor del servicio. De este modo, nadie podrá registrar servicios en el papel de un proveedor o hacer uso de ellos sin contar con los permisos adecuados.

6.4. Calidad

Actualmente ya existen en el mercado algunas herramientas específicamente diseñadas para medir la calidad de los servicios web, pero sigue siendo

necesaria una estandarización sobre este tema. Los resultados sobre la calidad de los diferentes servicios web, servirán como parámetro de comparación y ayudarán al consumidor a decantarse por un servicio u otro. Para que un servicio web se ejecute con corrección y satisfaga las expectativas creadas, aparte del precio, habrá que tener en cuenta una serie de parámetros como, por ejemplo, que los resultados obtenidos del mismo sean los esperados o que el entorno de uso sea amigable. Otro elemento a tener en cuenta es la integración. Aunque teóricamente los servicios web proporcionan conectividad con cualquier software de un modo transparente, cada proveedor de servicios puede adoptar soluciones diferentes que resultan más o menos adecuadas para el consumidor. Analizando la escalabilidad se comprobará el grado de modularidad y flexibilidad del servicio. Por último, también sería interesante analizar las características que ofrece el proveedor de servicios web. Actualmente no hay definidos estándares sobre este tema, pero la mayoría de las empresas ya está demandando algún tipo de acuerdo o contrato con los proveedores, de modo que se pueda garantizar la calidad y la fiabilidad de los servicios por los que se paga.

6.5. Estandarización

Los servicios web están basados en el estándar XML, que ha sido universalmente aceptado. Algunas de las empresas más importantes en el desarrollo de negocio electrónico como IBM, Intel, Microsoft u Oracle, han creado el WS-I: organización para la Interoperabilidad de los servicios web. El objetivo de dicha organización es la promoción de la estandarización de estos servicios de modo que se fomente la cooperación e interoperabilidad entre las compañías y los mercados.

6.6. Conceptos e ideas sobre servicios web

"Los servicios web son la revolución informática de la nueva generación de aplicaciones que trabajan colaborativamente en las cuales el software esta distribuido en diferentes servidores."

"Los servicios XML web son los bloques de construcción de la computación distribuida en el Internet. Se pueden crear soluciones al usar los múltiples servicios de XML Web desde varias fuentes que trabajan en conjunto-independientemente de dónde residan o cómo fueron implementadas."

"Un «Web Service» es un componente de software que se comunica con otras aplicaciones codificando los mensaje en XML y enviando estos mensaje a través de protocolos estándares de Internet tales como el Hypertext Transfer Protocol (HTTP). Intuitivamente un servicio web es similar a un sitio web que no cuenta con un interfaz de usuario y que da servicio a las aplicaciones en vez de a las personas; en vez de obtener solicitudes desde el navegador y retornar páginas web como respuesta, lo que hace es recibir solicitudes a través de un mensaje formateado en XML desde una aplicación, realiza una tarea y devuelve un mensaje de respuesta también formateado en XML."

Microsoft y otras empresas líderes están promocionando SOAP como estándar de los mensajes para estos servicios. Un mensaje SOAP se parece

mucho a una carta: es un sobre que contiene una cabecera con la dirección del receptor del mensaje, un conjunto de opciones de entrega (tal como la información de encriptación) y un cuerpo o body con la información o data del mensaje.

Microsoft y otros proveedores líderes promocionan los "Web Services" como un modelo de programación para la comunicación entre aplicaciones. Estas compañías piensan que la conexión de aplicaciones a través de la Internet mejorará la capacidad de las empresas para trabajar conjuntamente con sus socios de negocio, proveedores y clientes. Creando una capa de servicios web sobre una aplicación corporativa existente, las organizaciones podrán permitir que sistemas externos puedan invocar las funciones de la aplicación a través de Internet (o una intranet corporativa) sin tener que modificar la aplicación misma. Por ejemplo, varias compañías están hoy en día creando servicios que actúan como front end para aplicaciones de entrada de órdenes que están residentes internamente en un mainframe. Estas compañías permiten a los sistemas de compras de sus clientes enviar órdenes de compra a través de la Internet. Poner una capa de servicios web sobre las aplicaciones existentes es una solución muy interesante para integrar las aplicaciones desarrolladas por los diferentes departamentos y así reducir los costos de integración.

6.7. Requisitos de un servicio web

— **Interoperabilidad:** un servicio remoto debe permitir su utilización por clientes de otras plataformas.

— **Amigabilidad con Internet:** la solución debe poder funcionar para soportar clientes que accedan a los servicios remotos desde Internet.

— **Interfaces fuertemente tipadas:** no debería haber ambigüedad acerca del tipo de dato enviado y recibido desde un servicio remoto. Más aún, los tipos de datos definidos en el servicio remoto deben poderse corresponder razonablemente bien con los tipos de datos de la mayoría de los lenguaje de programación procedimentales.

— **Posibilidad de aprovechar los estándares de Internet existentes:** la implementación del servicio remoto debería aprovechar estándares de Internet existentes tanto como sea posible y evitar reinventar soluciones a problemas que ya se han resuelto. Una solución construida sobre un estándar de Internet ampliamente adoptado puede aprovechar conjuntos de herramientas y productos existentes creados para dicha tecnología.

— **Soporte para cualquier lenguaje:** la solución no debería ligarse a un lenguaje de programación particular JAVA RMI, por ejemplo, esta ligada completamente a lenguaje JAVA. Sería muy difícil invocar funcionalidad de un objeto JAVA remoto desde Visual Basic o PERL. Un cliente debería ser capaz de implementar un nuevo servicio web existente independientemente del lenguaje de programación en el que se halla escrito el cliente.

— **Soporte para cualquier infraestructura de componente distribuida:** la solución no debe estar fuertemente ligada a una infraestructura de

componentes en particular. De hecho, no se bebería requerir el comprar, instalar o mantener una infraestructura de objetos distribuidos, solo construir un nuevo servicio remoto utilizar un servicio existente. Los protocolos subyacentes deberían proporcionar un nivel base de comunicación entre infraestructura de objeto distribuidos existentes tales como DCOM y CORBA.

6.8. Bloques constructivos de servicios web

Descubrimiento UDDI, DISCO
Descripción WSDL, Esquema XML, Docs
Formato de Mensaje SOAP
Codificación XML
Transporte HTTP, SMTP y otros

— **Descubrimiento:** la aplicación cliente que necesita acceder a la funcionalidad que expone un servicio web necesita una forma de resolver la ubicación de servicio remoto. Se logra mediante un proceso llamado, normalmente descubrimiento (discovery). El descubrimiento se puede proporcionar mediante un directorio centralizado así como por otros métodos ad hoc. En DCOM, el servicio de descubrimiento lo proporciona el Administrador de control de servicios (SCM, Services Control Manager).

— **Descripción:** una vez que se ha resuelto el extremo de un servicio web dado, el cliente necesita suficiente información para interactuar adecuadamente con el mismo. La descripción de un servicio web implica meta datos estructurados sobre la interfaz que intenta utilizar la aplicación cliente así como documentación escrita sobre el servicio web incluyendo ejemplo de uso. Un componente DCOM expone meta datos estructurados sobre sus interfaces mediante una biblioteca de tipo (typelib). Los meta datos dentro de una typelib de componente se guardan en un formato binario propietario a los que se accede mediante una interfaz de programación de aplicación (API) propietaria.

— **Formato del mensaje:** para el intercambio de datos, el cliente y el servidor tienen que estar de acuerdo en un mecanismo común de codificación y formato de mensaje. El uso de un mecanismo están-

dar de codificación de datos asegura que los datos que codifica el cliente los interpretará correctamente el servidor. En DCOM los mensajes que se envían entre un cliente y un servidor tienen un formato definido por el protocolo DCOM Object RPC (ORPC).

— **Codificación:** los datos que se trasmiten entre el cliente y el servidor necesitan codificarse en un cuerpo de mensaje. Dcom utiliza un esquema de codificación binaria para serializar los datos de los parámetros que se intercambian entre el cliente y el servidor.

— **Transporte:** una vez se ha dado formato al mensaje y se han serializado los datos en el cuerpo del mensaje se debe transferir entre el cliente y el servidor utilizando algún protocolo de transporte. DCOM dispone de varios protocolos propietarios como TCP, SPX, NetBEUI y NetBIOS sobre IPX.

6.8.1. SOAP (Simple Object Access Protocol)

SOAP Version 1.2
Latest version of SOAP Version 1.2 specification: http://www.w3.org/TR/soap12
W3C Recommendation (Second Edition) 27 April 2007
SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)
W3C Recommendation 27 April 2007 http://www.w3.org/TR/soap12-part1/ http://www.w3.org/TR/soap12-part2/ http://www.w3.org/TR/soap12-part3/

Son las siglas de Simple Object Access Protocol. Este protocolo deriva del protocolo XML-RPC. SOAP; proporciona un mecanismo estándar para empaquetar mensajes. SOAP ha recibido gran atención debido a que facilita una comunicación del estilo RPC entre un cliente y un servidor remoto. Pero existen multitud de protocolos creados para facilitar la comunicación entre aplicaciones, incluyendo RPC de Sum, DCE de Microsoft, RMI de JAVA y ORPC de CORBA.

SOAP es un protocolo basado en XML, que permite la interacción entre varios dispositivos y que tiene la capacidad de transmitir información compleja. Los datos pueden ser transmitidos a través de HTTP, SMTP, etc. SOAP especifica el formato de los mensajes. El mensaje SOAP está compuesto por un **envelope** (sobre), cuya estructura está formada por los siguientes elementos: **header** (cabecera) y **body** (cuerpo).

- **Estructura de los mensajes**

SOAP es el primer protocolo de su tipo que ha sido aceptado prácticamente por todas las grandes compañías de software del mundo. Compañías que en raras ocasiones cooperan entre sí están ofreciendo su apoyo a este protocolo. Algunas de las mayores Compañías que soportan SOAP son Microsoft, IBM, SUN Microsystems, SAP y Ariba.

Algunas de las ventajas de SOAP son:

— **No esta asociado con ningún lenguaje:** los desarrolladores involucrados en nuevos proyectos pueden elegir desarrollar con el último y mejor lenguaje de programación que exista pero los desarrolladores responsables de mantener antiguas aflicciones heredadas podrían no poder hacer esta elección sobre el lenguaje de programación que utilizan. SOAP no especifica una API, por lo que la implementación de la API se deja al lenguaje de programación, como en JAVA, y la plataforma como Microsoft .Net.

— **No se encuentra fuertemente asociado a ningún protocolo de transporte:** la especificación de SOAP no describe como se deberían asociar los mensajes de SOAP con HTTP. Un mensaje de SOAP no es más que un documento XML, por lo que puede transportarse utilizando cualquier protocolo capaz de transmitir texto.

— **No está atado a ninguna infraestructura de objeto distribuido:** la mayoría de los sistemas de objetos distribuidos se pueden extender, y ya lo están alguno de ellos para que admitan SOAP.

— **Aprovecha los estándares existentes en la industria:** los principales contribuyentes a la especificación SOAP evitaron, intencionadamente, reinventar las cosas. Optaron por extender los estándares existentes para que coincidieran con sus necesidades. Por ejemplo, SOAP aprovecha XML para la codificación de los mensajes, en lugar de utilizar su propio sistema de tipo que ya están definidas en la especificación esquema de XML. Y como ya se ha mencionado SOAP no define un medio de trasporte de los mensajes; los mensajes de SOAP se pueden asociar a los protocolos de transporte existentes como HTTP y SMTP.

— **Permite la interoperabilidad entre múltiples entornos:** SOAP se desarrolló sobre los estándares existentes de la industria, por lo que las aplicaciones que se ejecuten en plataformas con dicho estándares pueden comunicarse mediante mensaje SOAP con aplicaciones que se ejecuten en otras plataformas. Por ejemplo, una aplicación de escritorio que se ejecute en un PC puede comunicarse con una aplicación del back-end ejecutándose en un mainframe capaz de enviar y recibir XML sobre HTTP.

SOAP proporciona un mecanismo estándar para empaquetar un mensaje. Un mensaje SOAP se compone de un sobre que contiene el cuerpo del mensaje y cualquier información de cabecera que se utiliza para describir le mensaje.

El elemento raíz del documento es el elemento envelope. El ejemplo contiene dos subelementos, body y header. Un ejemplo de SOAP valido también puede contener otros elementos hijo en el sobre.

— El sobre puede contener un elemento header opcional que contiene información sobre el mensaje.

— Un mensaje debe estar dentro de sobre de SOAP bien construido. Un sobre se compone de un único elemento envelope el sobre puede contener un elemento Header y puede contener un elemento body. Si existe, la cabecera debe ser el elemento hijo inmediato del sobre, con el cuerpo siguiendo inmediatamente a la cabecera.

— El cuerpo contiene la carga de datos del mensaje y la cabecera contiene los datos adicionales que no pertenecen necesariamente al cuerpo del mensaje.

— Además de definir un sobre de SOAP, la especificación de SOAP define una forma de codificar los datos contenidos en un mensaje. La codificación de SOAP proporciona un mecanismo estándar para serializar tipos de datos no definidos en la parte 1 de la especificación del esquema de XML.

— La especificación de SOAP también proporciona un patrón de mensaje estándar para facilitar el comportamiento de tipo RPC. Se emparejan dos mensajes de SOAP para facilitar la asociación de un mensaje de petición con un mensaje de respuesta.

— La llamada a un método y sus parámetros se serializan en el cuerpo del mensaje de petición en forma de una estructura. El elemento raíz tiene el mismo nombre que el método objetivo, con cada uno de los parámetros codificado como un subelemento.

— El mensaje de respuesta puede contener los resultados de la llamada al método o una estructura de fallo bien definida. Los resultados de la llamada a un método se serializan en el cuerpo de la petición como una estructura.

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-
envelope">
 <env:Header>
 <m:reserva xmlns:m="http://www./reserva"
env:role=http://www.w3.org/2003/05/soap-
envelope/role/next>
   <m:referencia>      lolo123   </m:referencia>
<m:fechaYHora>2007-05-18T13:20:00.000-05:00</m:fechaYHora>
</m:reserva>
<n:pasajero xmlns:n="http://www../empleados"
env:role=http://www.w3.org/2003/05/soap-envelope/role/next>
   <n:nombre>Pepe Ejemplo</n:nombre>
  </n:pasajero>
 </env:Header>
```

```
<env:Body>
  <p:itinerario  xmlns:p="http://www.../reserva/viaje">
   <p:ida>
     <p:salida>Madrid</p:salida>
     <p:llegada>Los Angeles</p:llegada>
     <p:fechaSalida>2007-12-14</p:fechasalida>
     <p:horaSalida>última hora de la tarde</p:horaSalida>
     <p:preferenciaAsiento>pasillo</p:preferenciaAsiento>
   </p:ida>
   <p:vuelta>
     <p:salida>Los Angeles</p:salida>
     <p:llegada>Madrid</p:llegada>
     <p:fechaSalida>2007-12-20</p:fechaSalida>
     <p:horaSalida>media-mañana</p:horaSalida>
     <p:preferenciaAsiento/>
   </p:vuelta>
  </p:itinerario>
  <q:alojamiento   xmlns:q="http://www.../reserva/hote-
les">
    <q:preferencia>ninguna</q:preferencia>
  </q:alojamiento>
 </env:Body>
</env:Envelope>
```

6.8.2. WSDL (Lenguaje de Descripción de Servicios Web) para la documentación de servicios web

- **Segunda versión convertida a recomendación: W3C Recommendation 26 June 2007 (http://www.w3.org/TR/wsdl20/)**

XML no basta para describir un servicio web, ya que este ha de asentarse también en unos patrones; estos patrones los proporciona el lenguaje de descripción WSDL. WSDL, permite que un servicio y un cliente establezcan un acuerdo en lo que se refiere a los detalles de transporte de mensajes y su contenido, a través de un documento procesable por dispositivos. WSDL representa una especie de contrato entre el proveedor y el que solicita. WSDL especifica la sintaxis y los mecanismos de intercambio de mensajes.

Supongamos que se ha creado un servicio Web Calculadora. Este servicio web expone los métodos sumar y restar. Ambos métodos aceptan dos enteros y devuelven un único entero con el resultado; sumar devuelve la suma de los dos enteros y restar devuelve su diferencia.

En un esfuerzo para describir cómo interacciona un cliente con el servicio web se define un esquema para los mensajes que se intercambiarán entre el cliente y el servidor. El esquema contiene una definición de un tipo de com-

plejo para los mensajes de petición y repuesta para los métodos sumar y restar. Recuerde que el objetivo último es que los desarrolladores no tengan que investigar en las definiciones del esquema intentando descifrar cómo interaccionar con el servicio web. En lugar de ello se quiere describir el servicio de forma que una herramienta pueda descifrarlo y crear un proxy por el cliente.

Además de la información que proporciona el esquema, ¿Qué más necesita conocer el cliente para invocar los métodos que expone el Servicio Web Calculadora? Como el cuerpo de un mensaje de SOAP puede contener cualquier cosa que no invalide el XML los mensajes de SOAP se pueden combinar para disponer de una amplia variedad de patrones de intercambio de mensajes. Los patrones de intercambio de mensajes para el Servicio Web Calculadora son bastante inmediatos pero una asociación formal entre los mensajes de petición Sumar y Restar y sus mensajes de respuesta asociados eliminarían cualquier posible ambigüedad.

Algunos servicios podrían aceptar una petición pero no enviar la respuesta correspondiente devuelta al cliente. Otros podrían solamente enviar mensajes al cliente. Además, el esquema no contiene información sobre cómo acceder al servicio web. Como SOAP es independiente del protocolo, se intercambiarán los mensajes entre el cliente y el servidor de numerosas formas. ¿Cómo se sabe si hay que enviar un mensaje mediante HTTP, SMTP o cualquier otro protocolo de transporte? Más aún, ¿cómo se sabe la dirección la que hay que enviar el mensaje?

El lenguaje de descripción de servicios web (WSDL) es un dialecto basado en XML sobre el esquema que describe un servicio web. Un documento WSDL proporciona la información necesaria al cliente para interaccionar con el servicio web. WSDL es extensible y se pude utilizar para describir, prácticamente, cualquier servicio de red, incluyendo SOAP sobre HTTP e incluso protocolos que no se basan en XML como DCOM sobre UDP.

Dado que los protocolos de comunicaciones y los formatos de mensajes están estandarizados en la comunidad Web, cada día aumenta la posibilidad e importancia de describir las comunicaciones de forma estructurada. WSDL afronta esta necesidad definiendo una gramática XML que describe los servicios de red como colecciones de puntos finales de comunicación capaces de intercambiar mensajes. Las definiciones de servicio WSDL proporcionan documentación para sistemas distribuidos y sirven como fórmula para automatizar los detalles que toman parte en la comunicación entre aplicaciones.

Los documentos WSDL definen los servicios como colecciones de puntos finales de red o puertos. En WSDL, la definición abstracta de puntos finales y de mensajes se separa de la instalación concreta de red o de los enlaces del formato de datos. Esto permite la reutilización de definiciones abstractas: mensajes, que son descripciones abstractas de los datos que se están intercambiando y tipos de puertos, que son colecciones abstractas de operaciones. Las especificaciones concretas del protocolo y del formato de datos para un tipo de puerto determinado constituyen un enlace reutilizable. Un puerto se define por la asociación de una dirección de red y un enlace reutilizable; una colección de puertos define un servicio.

El esquema para un determinado conjunto de elementos de extensibilidad se debe definir dentro de distintos espacios de nombres que WSDL. La definición de los propios elementos puede contener un atributo wsdl:requiered que indique un valor boolean si este atributo se establece a true en una definición de elementos; una asociación que haga referencia a ese conjunto concreto de electos de extensibilidad tiene que incluir dicho elemento.

Lo más habitual es que los elementos de extensibilidad se utilicen para especificar especificación de asociación. La especificación WSDL define el conjunto de elementos de extensibilidad para la asociación SOAP, HTTP GET, HTTP POS, MIME. Sin embargo, la especificación solo define las asociaciones para dos de los cuatro tipos de operaciones. Un sentido y petición repuesta.

El grupo de trabajo Web Services Description ha hecho público el Lenguaje descriptor de servicios (WSDL) Versión 2.0:

— Parte 1: Núcleo del Lenguaje.

— Parte 2: Patrones de Mensaje.

WSDL es un modelo en formato XML para describir servicios en red. El lenguaje permite separar etapas fundamentales de función abstracta y detalles concretos.

WSDL es un protocolo basado en XML que describe los accesos al servicio web. Podriamos decir que es el manual de operación de estos servicios, porque nos indica cuales son las interfaces que provee y los tipos de datos necesarios para la utilización del mismo. Veamos un ejemplo de un documento WSDL:

```
<?xml version="1.0"> <definitions>  <types>  ...  </types>  <message>  ...  </message>
  <portType>  ...  </portType>  <binding>  ...  </binding> </definitions>
```

Un documento WSDL utiliza los siguientes elementos en la definición de servicios de red:

<definitions>	Comienzo del documento, este tag agrupa a todos los demás.
<types>	Se definen los tipos de datos utilizados en el servicio web. Contenedor de definiciones del tipo de datos que utiliza algún sistema de tipos (por ejemplo XSD).
</types>	Fin de la definición de tipos.
<message>	Se definen los métodos y parámetros para realizar la operación. Cada message puede consistir en una o más partes (parámetros). Definición abstracta y escrita de los datos que se están comunicando.
</message>	Fin de la definición de los parámetros.
<portType>	Esta sección es la más importante, ya que se definen las operaciones que pueden ser realizadas, y los mensajes que involucran (por ejemplo el mensaje de petición y el de respuesta). Conjunto abstracto de operaciones admitidas por uno o más puntos finales.
</portType>	Fin de la definición de las operaciones y mensajes.
<port>	Punto final único que se define como la combinación de un enlace y una dirección de red.
<service>	Colección de puntos finales relacionados.
<binding>	Se definen el formato del mensaje y detalles del protocolo para cada portType. Especificación del protocolo y del formato de datos para un tipo de puerto determinado.
</binding>	Fin de la definición del formato del mensaje y detalles del protocolo para cada PortType.
</definitions>	Fin del documento WSDL

Ejemplo Parte 1 (Núcleo del Lenguaje):

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:description
    targetNamespace="http://example.org/TicketAgent.wsdl20"
    xmlns:xsTicketAgent="http://example.org/TicketAgent.xsd"
    xmlns:wsdl="http://www.w3.org/ns/wsdl"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-ins-
tance"
    xsi:schemaLocation="http://www.w3.org/ns/wsdl
http://www.w3.org/2007/06/wsdl/wsdl20.xsd">
```

```
    <wsdl:types>
        <xs:import schemaLocation="TicketAgent.xsd"
         namespace="http://example.org/TicketAgent.xsd" />
    </wsdl:types>

    <wsdl:interface name="TicketAgent">
        <wsdl:operation name="listFlights"
           pattern="http://www.w3.org/ns/wsdl/in-out">
            <wsdl:input element="xsTicketAgent:listFlightsRe-
quest"/>
            <wsdl:output element="xsTicketAgent:listFlights-
Response"/>
        </wsdl:operation>

        <wsdl:operation name="reserveFlight"
          pattern="http://www.w3.org/ns/wsdl/in-out">
            <wsdl:input element="xsTicketAgent:reserveFligh-
tRequest"/>
            <wsdl:output element="xsTicketAgent:reserveFligh-
tResponse"/>
        </wsdl:operation>
    </wsdl:interface>
</wsdl:description>
```

Ejemplo Parte 2 (Patrones de Mensaje):

```
<description>
  <binding name="xs:NCName" interface="xs:QName"?
type="xs:anyURI"
            whttp:contentEncodingDefault="xs:string"? >

    <fault ref="xs:QName"
            whttp:contentEncoding="xs:string"? >
    </fault>*

    <operation location="xs:anyURI"?
              whttp:contentEncodingDefault="xs:string"? >
      <input messageLabel="xs:NCName"?
             whttp:contentEncoding="xs:string"? />
```

```
        <output messageLabel="xs:NCName"?
                whttp:contentEncoding="xs:string"? />

      </operation>
    </binding>
  </description>
```

6.8.3. UDDI (Universal Description Discovery and Integration). Un registro global de servicios web

- **UDDI Version 3.0. UDDI Spec Technical Committee Specification. http://www.uddi.org/pubs/uddi-v3.00-published-20020719.htm**

Una vez definido el servicio web, necesitamos darlo a conocer a la comunidad para que sepan de su existencia. Definir cómo se dará a conocer el servicio web para que los clientes interesados puedan descubrirlo fácilmente y utilizarlo en sus aplicaciones. En la actualidad, ya existe un mecanismo de descubrimiento que cumple estos requisitos: UDDI *(Universal Description Discovery and Integration)*, una iniciativa del sector para hacer compatible el descubrimiento de servicios web con todo tipo de tecnologías y plataformas.

UDDI es un registro público diseñado para almacenar de forma estructurada información sobre empresas y los servicios que estas ofrecen. A través de UDDI, se puede publicar y descubrir información de una empresa y de sus servicios. Se pueden utilizar sistemas taxonómicos estándar para clasificar estos datos y poder encontrarlos posteriormente en función de la categorización. Lo más importante es que UDDI contiene información sobre las interfaces técnicas de los servicios de una empresa. A través de un conjunto de llamadas a API XML basadas en SOAP, se puede interactuar con UDDI tanto en tiempo de diseño como de ejecución para descubrir datos técnicos de los servicios que permitan invocarlos y utilizarlos. De este modo, UDDI sirve como infraestructura para una colección de software basado en servicios web.

Varias empresas, incluidas Microsoft, IBM, Sun, Oracle, Compaq, Hewlett Packard, Intel, SAP y unas trescientas más (para obtener un listado completo, consulte UDDI: Community [en inglés]), unieron sus esfuerzos para desarrollar una especificación basada en estándares abiertos y tecnologías no propietarias. A partir de la creación de esta infraestructura para servicios Web, los datos sobre estos servicios se pueden encontrar de forma sistemática y confiable en una capacidad universal totalmente independiente de proveedores. Se pueden llevar a cabo búsquedas categóricas precisas utilizando sistemas de identificación y taxonómicos extensibles. La integración de UDDI en tiempo de ejecución se puede incorporar a las aplicaciones. Como resultado, se fomenta el desarrollo de un entorno de software de servicios web.

Nota. *En Biología, un taxón (del griego Ù·ÍÈ˜, ordenamiento) es un grupo de organismos emparentados, que en una clasificación dada han sido agrupados, asignándole al grupo un nombre en latín, una descripción, y un "tipo", que si el taxón es una especie es un espécimen o ejemplar concreto. Cada descrip-*

ción formal de un taxón es asociada al nombre del autor o autores que la realizan, los cuales se hacen figurar detrás del nombre.

La información de UDDI se aloja en nodos de operador, empresas que se han comprometido a ejecutar un nodo público conforme a la especificación que rige el consorcio UDDI.org. Existen nodos públicos que se ajustan a la versión 1 de la especificación UDDI: Microsoft aloja uno e IBM el otro. Hewlett Packard se ha comprometido a alojar un nodo bajo la versión 2 de la especificación. Los operadores del "host" deben replicar datos entre ellos a través de un canal seguro, para conseguir la redundancia de la información en el registro UDDI. Se pueden publicar los datos en un nodo y descubrirlos en otro tras la réplica. Actualmente, la réplica se produce cada 24 horas. En el futuro, este intervalo entre réplicas se reducirá, ya que habrá más aplicaciones que dependan de los datos de UDDI.

Resulta importante observar que no existen requisitos de propietario respecto al modo en que el operador del host implementa su nodo. El nodo solo se debe ajustar a la especificación UDDI. El nodo público UDDI constituye un claro ejemplo de que el modelo de servicios web XML funciona en entornos heterogéneos.

UDDI es relativamente ligero; se ha diseñado como registro, no como depósito. La diferencia, aunque sutil, resulta esencial. Un registro redirige al usuario a recursos, mientras que un depósito solo almacena información. El registro Microsoft® Windows® puede servir de ejemplo: contiene las configuraciones y parámetros básicos pero, en última instancia, su función es la de dirigir la aplicación a un recurso o binario. Buscar un componente COM basándonos en su Id de programa nos conducirá a un Id de clase, que a su vez nos dirigirá a la ubicación del binario. UDDI se comporta de forma similar: como el registro de Windows, se basa en identificadores únicos globales (GUID) para garantizar la capacidad de búsquedas y determinar la ubicación de recursos. En última instancia, las consultas a UDDI conducen a una interfaz (un archivo .WSDL, .XSD, .DTD, etc.) o a una implementación (como un archivo .ASMX o .ASP) ubicadas en otro servidor.

OASIS ratifica UDDI-OASIS (Organization for the Advancement of Structured Information Standards) ha aprobado UDDI (Universal Description, Discovery and Integration), uno de los estándares claves en la arquitectura de servicios Web junto a XML), SOAP (Simple Object Access Protocol) y WSDL (Web Services Description Language).

UDDI es un modelo de directorios para servicios web, es una especificación para mantener directorios estandarizados de información acerca de estos servicios, sus capacidades, ubicación, y requerimientos en un formato reconocido universalmente. UDDI utiliza WSDL para describir las interfaces de los servicios web. Es un lugar en el cual podemos buscar cuales son los servicios web disponibles, una especie de directorio en el cual podemos encontrar los publicados y publicar los que desarrollemos.

6.8.4. WSDL y UDDI

WSDL se ha convertido en una pieza clave de la pila de protocolos de los servicios web. Por eso, es importante saber cómo colaboran UDDI y WSDL y por qué la idea de interfaces frente implementaciones forma parte de cada

protocolo. WSDL y UDDI se diseñaron para diferenciar claramente los metadatos abstractos y las implementaciones concretas. Para entender cómo funcionan WSDL y UDDI resulta esencial comprender las consecuencias de esta división.

Por ejemplo, WSDL distingue claramente los mensajes de los puertos: los mensajes (la sintaxis y semántica que necesita un servicio web) son siempre abstractos, mientras que los puertos (las direcciones de red en las que se invoca al servicio web) son siempre concretos. No es necesario que un archivo WSDL incluya información sobre el puerto. Un archivo WSDL puede contener simplemente información abstracta de interfaz, sin facilitar datos de implementación concretos, y ser válido. De este modo, los archivos WSDL se separan de las implementaciones.

Una de las consecuencias más interesantes de esto es que pueden existir varias implementaciones de una única interfaz WSDL. Este diseño permite que sistemas dispares escriban implementaciones de la misma interfaz, para garantizar así la comunicación entre ellos. Si tres empresas diferentes implementan el mismo archivo WSDL y una parte del software de cliente crea el código auxiliar/proxy a partir de esa interfaz, dicho software se podrá comunicar con las tres implementaciones con el mismo código de base, cambiando simplemente el punto de acceso.

6.9. SOA

La Arquitectura Orientada a Servicios (SOA), define un modelo de arquitectura software que está teniendo bastante aceptación para entornos de integración de aplicaciones. El concepto básico de esta arquitectura es el de servicio. Los nodos de un sistema distribuido ofrecen servicios al resto del sistema de una forma estandarizada. Un servicio es una función autocontenida y sin estado que acepta una o varias peticiones y devuelve una o varias respuestas a través de una interfaz bien definida. Los servicios no dependen del estado de otras funciones o procesos. SOA proporciona una metodología y un marco de trabajo para documentar las capacidades de negocio y puede dar soporte a las actividades de integración y consolidación.

Los objetivos iniciales de la plataforma SOA son contar con un contenedor de servicios fiable y de altas prestaciones para el desarrollo de servicios SOA y que sirva como base para el desarrollo de posteriores herramientas y servicios de descubrimiento y orquestación de servicios web. En un ambiente SOA, los nodos de la red hacen disponibles sus recursos a otros participantes en la red como servicios independientes a los que tienen acceso de un modo estandarizado. La mayoría de las definiciones de SOA identifican la utilización de servicios web (empleando SOAP y WSDL) en su implementación, no obstante se puede implementar una SOA utilizando cualquier tecnología basada en servicios.

Al contrario que las arquitecturas orientadas a objetos, las SOAs están formadas por servicios de aplicación débilmente acoplados y altamente interoperables. Para comunicarse entre sí, estos servicios se basan en una definición formal independiente de la plataforma subyacente y del lenguaje de programación (p.ej., WSDL). La definición de la interfaz encapsula (oculta) las particularidades de una implementación, lo que la hace independiente del fabricante, del lenguaje de pro-

gramación o de la tecnología de desarrollo (como Plataforma JAVA o Microsoft .NET). Con esta arquitectura, se pretende que los componentes software desarrollados sean muy reusables, ya que la interfaz se define siguiendo un estándar; así, un servicio C Sharp podría ser usado por una aplicación JAVA.

Los lenguajes de alto nivel como BPEL o WS-coordinación llevan el concepto de servicio un paso adelante al proporcionar métodos de definición y soporte para flujos de trabajo y procesos de negocio.

Término	Definición / Comentario
Servicio	Una función sin estado, auto-contenida, que acepta una(s) llamada(s) y devuelve una(s) respuesta(s) mediante una interfaz bien definida. Los servicios pueden también ejecutar unidades discretas de trabajo como serían editar y procesar una transacción. Los servicios no dependen del estado de otras funciones o procesos. La tecnología concreta utilizada para prestar el servicio no es parte de esta definición.
Orquestación	Secuenciar los servicios y proveer la lógica adicional para procesar datos. No incluye la presentación de los datos. Coordinación.
Sin estado	No mantiene ni depende de condición pre-existente alguna. En una SOA los servicios no son dependientes de la condición de ningún otro servicio. Reciben en la llamada toda la información que necesitan para dar una respuesta. Debido a que los servicios son "sin estado", pueden ser secuenciados (orquestados) en numerosas secuencias (algunas veces llamadas tuberías o pipelines) para realizar la lógica del negocio.
Proveedor	La función que brinda un servicio en respuesta a una llamada o petición desde un consumidor.
Consumidor	La función que consume el resultado del servicio provisto por un proveedor.

La metodología de modelado y diseño para aplicaciones SOA se conoce como análisis y diseño orientado a servicios. La arquitectura orientada a servicios es tanto un marco de trabajo para el desarrollo de software como un marco de trabajo de implantación. Para que un proyecto SOA tenga éxito los desarrolladores de software deben orientarse ellos mismos a esta mentalidad de crear servicios comunes que son orquestados por clientes, o middleware, para implementar los procesos de negocio. El desarrollo de sistemas usando SOA requiere un compromiso con este modelo en términos de planificación, herramientas e infraestructura.

6.10. Detalles sobre los servicios web

A diferencia de DCOM y CORBA, que son binarios, SOAP usa el código fuente en XML. Esto es una ventaja ya que facilita su lectura por parte de

humanos, pero también es un inconveniente dado que los mensajes resultantes son más largos. El intercambio de mensajes se realiza mediante tecnología de componentes.

SOAP es un marco extensible y descentralizado que permite trabajar sobre múltiples pilas de protocolos de redes informáticas. Los procedimientos de llamadas remotas pueden ser modelados en la forma de varios mensajes SOAP interactuando entre sí.

SOAP funciona sobre cualquier protocolo de Internet, generalmente HTTP, que es el único homologado por el W3C. SOAP tiene como base XML, con un diseño que cumple el patrón Cabecera-Desarrollo de diseño de software, como otros muchos diseños, por ejemplo HTML. La cabecera Header es opcional y contiene metadatos sobre enrutamiento (routing), seguridad o transacciones. El desarrollo Body contiene la información principal, que se conoce como carga útil (payload). La carga útil se acoge a un XML Schema propio.

6.11. Ventajas de los servicios web

— Aportan interoperabilidad entre aplicaciones de software independientemente de sus propiedades o de las plataformas sobre las que se instalen.

— Los servicios web fomentan los estándares y protocolos basados en texto, que hacen más fácil acceder a su contenido y entender su funcionamiento.

— Al apoyarse en HTTP, los servicios web pueden aprovecharse de los sistemas de seguridad firewall sin necesidad de cambiar las reglas de filtrado.

— Permiten que servicios y software de diferentes compañías ubicadas en diferentes lugares geográficos puedan ser combinados fácilmente para proveer servicios integrados.

— Permiten la interoperabilidad entre plataformas de distintos fabricantes por medio de protocolos estándar.

6.12. Inconvenientes de los servicios web

— Para realizar transacciones no pueden compararse en su grado de desarrollo con los estándares abiertos de computación distribuida como CORBA (Common Object Request Broker Architecture).

— Su rendimiento es bajo si se compara con otros modelos de computación distribuida, tales como RMI (Remote Method Invocation), CORBA, o DCOM (Distributed Component Object Model). Es uno de los inconvenientes derivados de adoptar un formato basado en texto. Y es que entre los objetivos de XML no se encuentra la concisión ni la eficacia de procesamiento.

— Al apoyarse en HTTP, pueden esquivar medidas de seguridad basadas en firewall cuyas reglas tratan de bloquear o auditar la comunicación entre programas a ambos lados de la barrera.

— Existe poca información de servicios web para algunos lenguajes de programación.

7. Lenguajes de descripción de datos: HTML

7.1. Introducción

HTML es el lenguaje utilizado para la creación de páginas web. Las siglas proceden de *HyperText Markup Language* (lenguaje de marcado de hipertexto). Los documentos escritos en un lenguaje de marcado están formados por datos y marcas (etiquetas). Los datos constituyen el contenido del documento y las marcas proporcionan el significado de los datos o bien, indican algún otro tipo de atributo. El término **hipertexto** se utiliza para referirse a un documento que incluye texto combinado con enlaces, los cuales redireccionan al usuario a otras partes del mismo o a otros documentos. En realidad, los documentos HTML no solo contienen texto y enlaces, también pueden contener otros formatos de información multimedia como imágenes, vídeos o sonidos.

Los documentos HTML se pueden crear con un simple editor de textos o utilizando un programa de diseño web como Microsoft FrontPage o Macromedia Dreamweaver. Estos programas generan automáticamente el código HTML a partir de la representación visual que va elaborando el diseñador.

Una vez creados, los documentos HTML se almacenan en el servidor web. Cuando un cliente solicita una página, el servidor envía el documento HTML correspondiente. Finalmente, el navegador del usuario lee el documento recibido y lo representa en pantalla. Un navegador, visualizador HTML o browser no es más que un programa capaz de representar documentos HTML, ya sean locales o estén en un servidor web remotos. Algunos de los navegadores más conocidos son: Microsoft Internet Explorer, Google Chrome, Firefox, Safari u Opera.

HTML es un subconjunto del metalenguaje SGML *(Standard Generalized Markup Language*, lenguaje de marcas generalizado estandarizado) para la creación de páginas web. Metalenguaje significa lenguaje para la creación de otros lenguajes. XML *(Extensible Markup Language*, lenguaje de marcas extensible) también es otro metalenguaje, basado en SGML, aunque mucho más sencillo. En XML se han excluido los elementos menos utilizados y más complejos de SGML.

La primera versión de HTML data de 1992. Esta versión disponía de un número muy reducido de etiquetas. Desde esta especificación inicial, el lenguaje ha ido evolucionando en distintas versiones hasta llegar a la versión 4.0 en diciembre de 1997, mejorada posteriormente por HTML 4.01 (diciembre de 1999). La siguiente versión a HTML 4.01 se considera XHTML 1.0 (enero de 2000) que es, en realidad, una reformulación de HTML 4 en el lenguaje XML. El organismo responsable del desarrollo de las recomendaciones HTML y del resto de estándares de la web es W3C (Consorcio para la World Wide Web).

El grupo WHATWG *(Web Hipertext Application Technology Working Group)*, formado por empresas como Apple, Opera y Mozilla, publicó en 2007 el borrador de HTML 5.0 y W3C liberó esta versión como estándar oficial en octubre de 2014. HTML 5.0 incorpora diversos cambios respecto a la versión 4.01, como nuevas etiquetas o la posibilidad de incorporar audio y vídeo en la web sin necesidad de plugins.

La mayor parte de ejemplos de este tema se refieren a la versión 4.0 por ser la versión preferida por los exámenes.

7.2 Elementos. Sintaxis de las etiquetas

Los documentos HTML son archivos en formato de texto que contienen información textual y etiquetas. La información se mostrará en el navegador del usuario según lo indiquen las etiquetas. Por ejemplo, se puede utilizar una etiqueta para destacar un párrafo con un determinado color o tipo de letra. Las etiquetas también se utilizan para incluir otros elementos adicionales en las páginas web como imágenes, sonidos, scripts o applets de JAVA.

Todas las etiquetas se encuentran delimitadas entre los símbolos < (menor que) y > (mayor que). Lo que esté fuera de estos símbolos son textos que, generalmente, se mostrarán en el navegador del usuario.

Un elemento de HTML está formado por una etiqueta de apertura, un contenido y una etiqueta de cierre. En ocasiones, se omiten partes del elemento. Al contenido del elemento se le aplicará lo que indique la etiqueta.

— **Etiqueta de apertura.** Tiene la sintaxis general de <nombreetiqueta atributos>. Los atributos indican propiedades adicionales de la etiqueta. Una etiqueta puede tener cero, uno o más atributos. El orden en el que se escriban los atributos no es relevante. Por ejemplo: <nombreetiqueta atributo1=”valor1” atributo2=”valor2”>. Como vemos, se utiliza el delimitador espacio para separar el nombre de la etiqueta del primer atributo y un atributo de otro. Para cada atributo se indica el valor que recibe entre comillas.

— **Etiqueta de cierre.** Su sintaxis es </nombreetiqueta>. Una etiqueta de cierre siempre comienza con la barra inclinada '/'. No incluye atributos.

Veamos varios ejemplos:

— Este texto aparecerá en negrita

— <h1 align=”center”>Título que aparecerá centrado</h1>

— <img src=”foto.jpg” width=”100” height=”200”>

Aspectos a tener en cuenta:

— HTML 4 no distingue entre mayúsculas y minúsculas en las etiquetas y atributos, por lo que <script> y <SCRIPT> serían equivalentes. Por el contrario, XHTML requiere que las etiquetas y atributos aparezcan en minúsculas.

— La utilización de comillas en los valores de los atributos de HTML 4 es opcional en ciertos casos, aunque se recomienda su utilización. La etiqueta <img src=foto.jpg width=100 height=200> sería correcta

sintácticamente. Es obligatoria la utilización de comillas cuando el valor de los atributos contenga un carácter diferente a números (0-9), letras (a-z y A-Z), guión (-), subrayado (_), punto (.) o dos puntos (:). Por ejemplo <img src=”foto del viaje.jpg”>. En XHTML es obligatorio el uso de comillas en todos los casos.

— En HTML 4, no es obligatorio cerrar algunas etiquetas (elementos vacíos). Por ejemplo, las etiquetas <img>,
, <hr>, <p>, <li> o <td>. En XHTML, por el contrario, es obligatorio cerrar todas las etiquetas. Esto se puede hacer de dos formas: abriendo y cerrando con la misma etiqueta (se coloca una barra al final para indicarlo:
) o bien, colocando la etiqueta de cierre a continuación de la de apertura (ejemplo:
</br>).

— Existen algunos atributos generales que son válidos para todas las etiquetas (como los atributos class o id). Sin embargo, hay otros que son específicos de cada etiqueta (como el atributo src en la etiqueta <img>).

Se pueden incluir comentarios dentro del código HTML. Estos comentarios no serán interpretados por el navegador, aunque sí serían vistos por el visitante si este consultase el código fuente de la página. Los comentarios se utilizan habitualmente para señalizar bloques o partes del documento en códigos HTML largos. Los comentarios se delimitan entre los símbolos <!— (menor que, admiración, guión, guión) y —> (guión, guión, mayor que). Pueden abarcar una o más líneas. Ejemplo:

— <!— Aquí comienza el menú principal —>

7.3. Estructura general de un documento HTML

Un código HTML está delimitado por las etiquetas <html> y </html>. Contiene dos secciones:

— **Cabecera.** Se delimita por las etiquetas <head> y </head>. Contiene información adicional del documento, como su título, su autor o la codificación internacional que se ha empleado, así como hojas de estilo, scripts u otros elementos. Cada documento, según las especificaciones de HTML 4, debe tener exactamente un título dentro de la sección de cabecera. El título se indica con el elemento <title>. El contenido de este elemento se mostrará habitualmente en la barra de título de la ventana del navegador.

— **Cuerpo del documento.** Se incluye entre las etiquetas <body> y </body>. Se corresponde con el contenido del documento, esto es, aquella información que se mostrará en el navegador del usuario. La sección de body es la más extensa del documento HTML.

A continuación mostramos la estructura general de un documento HTML:

```
<html>
     <head>
          <title>Este es el título de la página</title>
     </head>
     <body>
          Aquí se incluye el contenido de la página.
     </body>
</html>
```

Atributos que puede aparecer en la etiqueta <body>:

- **background**= archivo de imagen de fondo.
- **bgcolor**= color de fondo del documento.
- **text**= color del texto.
- **link**= color de los enlaces no visitados.
- **vlink**= color de los enlaces visitados.
- **alink**= color del enlace que está seleccionando el usuario.

Ejemplo:

```
<body background="fondo.jpg" text="black" link="blue"
vlink="red" alink="green">
```

7.4. Elementos de la sección de cabecera

Los siguientes elementos aparecen en la sección de cabecera:

A) Elemento: <title>

— **Explicación**: título del documento. El título del documento aparece habitualmente en el título de la ventana del navegador. No confundir esta etiqueta con los títulos que aparecen en el cuerpo del documento (<h1>, <h2>, etc...).

Ejemplo: `<title>Programación en Internet</title>`

B) Elemento: <meta>

— **Explicación**: información del documento. La información de estas etiquetas no se muestra al usuario en el navegador, sin embargo puede ser leída por programas específicos (motores de búsqueda,

por ejemplo) o consultando manualmente el código fuente de la página. Permite almacenar datos como el autor de la página, el programa de diseño web que se ha utilizado, la descripción del documento, palabras clave, etc.

Ejemplos: `<meta name="Description" content="página de ejemplo de HTML">`

<meta name="Keywords" content="oposiciones, tai, programación">

<meta http-equiv="Refresh" content="10;URL=http://www.adams.es"> (esta última etiqueta genera una redirección automática después de 10 segundos a la URL indicada)

C) Elemento: <style>

— **Explicación**: se utiliza para incluir una hoja de estilos CSS en el propio documento.

Ejemplo: `<style type="text/css"> strong {color: red;} a {text-decoration: none; } </style>`

D) Elemento: <link>

— **Explicación:** enlace en la sección de cabecera (al contrario que <a> que son enlaces en el cuerpo del documento). Uno de los usos más frecuentes de este elemento es para llamar a una hoja de estilo CSS ubicada en un documento externo.

Ejemplo: `<link href="hoja-de-estilos.css" type="text/css" rel="stylesheet">`

E) Elemento: <script>

— **Explicación:** código de un lenguaje de script, JavaScript normalmente, tanto almacenado en un documento externo como en el propio documento. El elemento <script> también puede aparecer en la sección del cuerpo del documento. La ventaja de incluir los scripts en la cabecera es que los mismos se cargarán antes de que comience a mostrarse el documento en el navegador del usuario. Por este motivo, es frecuente colocar en la cabecera las funciones o trozos de códigos que serán llamados desde otras partes del documento.

Ejemplo: `<script src="codigo-javascript.js" language="JavaScript" type="text/javas cript">`

7.5. Marcos (frames)

Lo habitual es que cada documento HTML se ubique en el área de trabajo de la ventana del navegador. Sin embargo, los marcos permiten dividir el área del navegador en varias partes y en cada una de ellas, desplegar un documento HTML diferente. Su uso más frecuente es la utilización de un marco

para mantener cierta información visible permanentemente (por ejemplo, una botonera con opciones) y otro marco con información variable.

Para generar marcos se requiere, por un lado, un documento HTML que defina la estructura de los marcos (frameset) y por otro, tantos documentos HTML como marcos se hayan definido.

El siguiente ejemplo genera una estructura con dos marcos. Cuando se invoque este código, el navegador dividirá su área de trabajo en dos columnas. La de la derecha ocupa un 10% de la anchura del navegador y la de la izquierda, un 90%. En la primera división aparecerá el documento indice.htm y en la segunda, portada.htm:

```
<html>
     <head>
          <title>Ejemplo de frames</title>
     </head>
     <frameset cols="10%,90%">
          <frame name="indice" src="indice.htm">
          <frame name="contenido" src="portada.htm">
     </frameset>
</html>
```

Se puede realizar una división por filas, cambiando el atributo cols de <frameset> por rows. Es posible también anidar un <frameset> dentro de otro <frameset> y generar así estructuras de marcos más complejas.

7.6. Elementos de formato de textos

Los elementos que veremos en los próximos apartados se ubican dentro de la sección del cuerpo del documento (<body>).

Comenzamos repasando los elementos relativos al formato de texto. A pesar del siguiente listado, las recomendaciones de diseño actuales instan a utilizar hojas de estilo en lugar de etiquetas de presentación. Esto es: en lugar de emplear etiquetas como <b> para indicar "negrita", se prefiere la utilización de otras etiquetas más generales, cuyas características de presentación concretas (tipos de letras, tamaños, negritas, colores, alineaciones, separación de párrafos, etc...) queden definidas en una hoja de estilo.

— <b> Texto en negrita.

— <i> Texto en cursiva.

— <u> Texto subrayado.

— <font> Tipo de letra. Permite definir el tipo de letra (atributo face), color (atributo color) y tamaño (atributo size).

—
 Salto de línea. No es necesaria etiqueta de cierre.

— <hr> Línea horizontal.

— <tt> Letra de máquina de escribir.

— Texto centrado.

— <pre> Texto preformateado. Se conservarán los espacios en blanco y los saltos de línea tal y como aparecen en el código fuente.

— <blackquote> Texto sangrado.

— <blink> Texto con parpadeo. Este elemento no está contemplado por el W3C, únicamente funciona en el navegador Netscape.

Los siguientes elementos añaden información estructural (aportan significado) a los documentos. En lugar de indicar cómo se deben mostrar, indican el tipo de texto que delimitan. Las características de presentación se definirán en una hoja de estilos. Por ejemplo: se puede escribir ejemplo y en la hoja de estilos, indicar que todos los elementos se muestren en color rojo.

— <p> Párrafo. Se puede indicar la alineación del párrafo mediante el atributo "align". No es necesaria etiqueta de cierre.

— <h1>, <h2>, <h3>, <h4>, <h5> y <h6> Títulos. <h1> es el título (epígrafe) más general, <h2> es un subtítulo de <h1> y así sucesivamente, hasta llegar a <h6>, que es el título de menor importancia.

— Texto con énfasis.

— Texto destacado (con mayor énfasis).

— <address> Dirección. Se utiliza para indicar la forma de contacto con los autores de un texto mostrado en la página.

— <cite> Cita o referencia a otras fuentes.

— <code> Código de ordenador.

— <dfn> Definición.

— <kbd> Teclado. Texto que debe introducir el usuario en un programa.

— <samp> Ejemplo de salida de un programa.

— <var> Variable de un programa.

Ejemplo:

```
<h1>Ceremonia de los Oscars</h1>

<p align="right"><font face="Arial,Helvetica" size="3"
color="blue"><em>El Señor de los Anillos: El Retorno del
Rey</em> fue la vencedora de la 76 edición de los
<i>Oscars</i>, ya que obtuvo nada menos que
<strong>11</strong> Oscars</font></p>
```

7.7. Referencias de caracteres

Las referencias de caracteres o referencias de entidad definen una serie de nombres para que los autores puedan referirse a caracteres especiales dentro de un documento. Comienzan por el símbolo "&" y terminan con un punto y coma. Algunos de ellos aparecen frecuentemente en documentos HTML:

— < representa el carácter menor que (<).

— > representa el carácter mayor que (>).

— & representa el carácter &.

— " representa las comillas dobles (").

Por ejemplo: el texto 5<2 se codificaría en HTML como 5<2 y de esta forma se evitaría la confusión con el símbolo de apertura de etiqueta.

Existen también referencias de caracteres para aquellos específicos de cada idioma como, por ejemplo: á (á), Â (Â), Ç (Ç), ñ (ñ).

7.8. Colores

Hemos visto que algunas etiquetas HTML incluyen atributos de color. En este tipo de atributos, así como en ciertas propiedades de las hojas de estilo, es necesario representar el color deseado con una notación apropiada. En este apartado estudiamos las notaciones admitidas:

— Nombre del color: black, white, red, blue, yellow, green, maroon, gray, cyan, purple, etc.

— Notación RGB. Se indican 3 bytes expresados en hexadecimal. El primer byte indica la cantidad de rojo; el segundo, la cantidad de verde y el tercero, la cantidad de azul. Ejemplos: #000000 (negro), #FFFFFF (blanco), #FF0000 (rojo), #D50000 (un rojo más oscuro), #FFFF00 (amarillo), #FFAC30 (una tonalidad naranja).

— Notación RGB abreviada. Similar al anterior, aunque solo se indica el primer carácter de cada byte. Por ejemplo: #ABC se corresponde en realidad con el color #AABBCC. La gama de colores obtenida utilizando la notación abreviada es menor que con la notación anterior, aunque suele ser más que suficiente.

7.9. Imágenes

Se pueden incluir imágenes en documentos HTML mediante el elemento <img>. Este elemento solo tiene etiqueta de apertura, no tiene ni contenido ni etiqueta de cierre. El siguiente ejemplo muestra la imagen del archivo foto.jpg con 200 píxeles de anchura y 300 de altura. Si no se especifican las dimensiones, se toman las de la imagen original.

```
<img src="foto.jpg" width="200" height="300">
```

7.10. Enlaces

Los enlaces (links, hiperenlaces, vínculos o hipervínculos) son el principal elemento de las páginas web. Sin enlaces no existiría la navegación. Un enlace permite referenciar otra página web u otra zona de la misma para que el usuario, haciendo clic en el mismo, salte al lugar indicado.

También se pueden referenciar imágenes o archivos para descargar.

Su sintaxis es:

```
<a href="localización de la página">Texto que aparecerá en el enlace</a>.
```

Ejemplos:

- `<a href="portada.htm">Ir a la portada</a>`
- `<a href="http://www.adams.es">Centro de Estudios Adams</a>`
- `<a href="http://www.adams.es"><img src="logotipo-adams.jpg"></a>`
- `<a href="archivos/tema.pdf">Descarga del tema</a>`

Obsérvese que para indicar la localización del recurso, se emplean atributos diferentes en las etiquetas <img> y <a> (href en <a> y src en <img>).

7.11. Listas

HTML dispone de estructuras para crear listas de información. Las listas están compuestas por elementos (items). Podemos ver las listas de HTML como las numeraciones y viñetas de un procesador de textos.

Se contemplan tres tipos de listas:

— **Lista ordenada (etiqueta <ol>)**. Muestra los elementos numerados (1, 2, 3, ...; A, B, C, ...; i, ii, iii, etc.).

— **Lista no ordenada (etiqueta <ul>)**. Muestra los elementos anteponiéndoles un mismo icono (disk, circle o square) a todos ellos.

— **Lista de definición (etiqueta <dl>)**. Muestra un listado de definiciones. Cada elemento está formado por el término que se define y su definición.

Los elementos de las listas ordenadas y no ordenadas se indican con la etiqueta <li>. No es obligatorio cerrarla.

En el caso de las listas de definición, el elemento que se define se indica con la etiqueta <dt> y su definición, con la etiqueta <dd>. Tampoco es obligatorio cerrar estas dos etiquetas.

Ejemplo:

```
<ul type="circle">
     <li>JAVA</li>
     <li>C++</li>
     <li>Eiffel</li>
</ul>
```

7.12. Tablas

Las tablas permiten mostrar información organizada en filas y columnas. La intersección de una fila con una columna se denomina celda. El elemento HTML que genera una tabla es <table>. Dentro de una tabla, se definen tantas filas como sean necesarias con el elemento <tr>. Y dentro de una fila, se indican las celdas que lo componen (columnas) mediante elementos <td>. No es obligatorio cerrar las etiquetas <td> y <tr>.

El siguiente ejemplo representa una tabla con 2 filas y 3 columnas:

```
<table>
     <tr>
          <td>Celda 1.1</td>
          <td>Celda 1.2</td>
          <td>Celda 1.3</td>
     </tr>
     <tr>
          <td>Celda 2.1</td>
          <td>Celda 2.2</td>
          <td>Celda 2.3</td>
     </tr>
</table>
```

7.13. Formularios

Los formularios HTML permiten recoger información del usuario y enviársela a un programa para que la procese. Los formularios están compuestos por elementos de formulario como, por ejemplo, cuadros de texto o listas desplegables. Todos los elementos de un mismo formulario deben estar contenidos en un elemento <form>. La sintaxis de este elemento es:

```
    <form method="método de envío" action="localización
del programa">
```

El atributo method puede contener dos valores: get si los parámetros escritos por el usuario deben enviarse en la URL o post, si deben enviarse de forma oculta. El atributo action indica el programa que procesará los datos escritos por el usuario. Este programa suele ser un archivo CGI o una página dinámica (ASP, PHP o JSP). Ejemplo:

```
<form method="post" action="buscador.asp">
    <!-- Aquí se incluyen los elementos del formulario -->
</form>
```

Elementos del formulario:

— **Cuadro de texto** (single-line text input). Recoge números o un texto breve (de una línea). Ejemplo: <input type="text" name="apellidos" size="10" maxlength="30">.

— **Botones de radio** (radio buttons). Presentan varias opciones, de las cuales solo se puede escoger una de ellas. Ejemplo: <input type="radio" name="color" value="rojo"> <input type="radio" name="color" value="amarillo">.

— **Casilla de verificación** (checkboxes). Presenta una opción que se puede activar o no independientemente del resto de opciones. Ejemplo: <input type="checkbox" name="tiene_coche" value="si">.

— **Lista desplegable** (menus). Ofrece un conjunto de opciones. Se utiliza en lugar de los botones de radio cuando el número de opciones es elevado.

  Ejemplo:

```
<select name="edad"><option value="1">20 años o
menos</option>
<option value="2">entre 21 y 60</option>
<option value="3">más de 60 años</option></select>
```

— **Área de texto** (multi-line text input). Recoge un texto de varias líneas. Ejemplo: <textarea name="comentarios">Aquí puede ir el texto</textarea>.

— **Botones** (buttons). Muestra un botón que desencadenará una acción cuando el usuario lo pulse. Ejemplo: <input type="button" value="Calcular total">. Existen dos botones predefinidos, uno para enviar el formulario completo (submit) y otro para borrar lo que el usuario haya escrito (reset). Ejemplo: <input type="submit" value="Enviar"> <input type="reset" value="Borrar">.

8. Lenguajes de descripción de datos: XML y sus derivaciones

8.1. Introducción

El objetivo XML no es la aplicación de estilos ni de formatos sobre un documento creado con un contenido, sino que estamos ante un lenguaje pensado para estructurar documentos. De lo dicho podemos deducir que cualquier documento consta de tres partes bien diferenciadas:

— **Contenido.** Es el texto escrito en si. Este contenido puede estar escrito con cualquier editor de texto plano. Por supuesto es lo que interesa al usuario, no obstante este contenido, falto de formatos y de una estructura, representa por si solo algo difícil de hacer vistoso y por supuesto difícil de catalogar e indexar.

— **Formato.** Se le pone una cara bonita al texto plano anterior, usando formatos de fuentes, párrafos, colores, márgenes, etc. Hoy en dia es mejor recurrir a formatear nuestros documentos usando estilos, como son las hojas de estilo en cascada (CSS).

— **Estructura.** Cualquier documento contiene partes claramente diferenciadas, como son títulos, subtítulos, niveles, numeraciones, etc. La estructura de un documento esta definida por la forma en cómo se distribuyen los elementos que lo componen. Podemos hacer representaciones gráficas de estas estructuras a través de árboles. XML está diseñado para trabajar con la estructura de los documentos (así como los campos, definidos en la vista diseño de una tabla de una base de datos, definen la estructura de la tabla).

8.1.1. Lenguajes de marcas

Las marcas son códigos que indican a un programa cómo debe tratar su contenido. La forma que ideó IBM en la década de los 60 para pasar información de un sistema a otro sin necesidad de perder el formato indicado se basaba en tratar las marcas como texto accesible desde cualquier sistema, texto plano, código ASCII. Y la norma se denominó GML (General Modeling Language).

Más tarde GML pasó a manos de ISO y se convirtió en SGML (ISO 8879), Standart Generalized Markup Language. Esta norma es la que se aplica desde entonces a todos los lenguajes de marcas, cuyos ejemplos más conocidos son el HTML y el RTF.

Los lenguajes de marcas no son equivalentes a los lenguajes de programación. Son sistemas complejos de descripción de información, normalmente documentos, que si se ajustan a SGML, se pueden controlar desde cualquier editor ASCII. Las marcas más utilizadas suelen describirse por textos descriptivos encerrados entre signos de menor (<) y mayor (>), siendo lo más usual que exista una marca de principio y otra de final.

La diferencia fundamental entre HTML y XML es que mientras el HTML es un lenguaje, XML es un metalenguaje. De hecho es un subconjunto de SGML, una versión reducida y totalmente compatible diseñada específicamente para la web. Todos los documentos XML se ajustan a las normas de SGML.

Según el documento "Extensible Markup Language (XML) 1.0 W3C Recommendation", los documentos XML deben ser fáciles de crear, legibles por personas (no solo por computadoras) y razonablemente claros. También debe ser fácil escribir programas de tratamiento de documentos XML y se debe reducir al mínimo el número de elementos opcionales.

8.1.2. Vista general de XML

Cuando trabajamos con XML, estamos trabajando con estructuras de documentos. Cuando accedemos a la información (datos) que XML estructura necesitamos de algún visualizador, como puede ser un navegador (browser), pero al mismo tiempo necesitamos también un analizador (parser) que no nos dejara acceder a los contenidos del documento hasta que no esté "bien formado". Se dice que un documento está "bien formado" cuando cumple las normas básicas de XML. Se dice que un documento XML es "válido" cuando además de cumplir las normas básicas también se cumplen las normas impuestas en un DTD (Definición de Tipo de Documento).

XML permite al usuario manejar los elementos como mejor le convenga, cosa que el código HTML no permite. Además con XML el usuario podrá ordenar los datos o actualizarlos en tiempo real. XML también permite realizar un estándar de almacenamiento estructurado. Un error que se suele cometer es considerar a XML un HTML extendido. XML es el estándar de Extensible Markup Language, es un conjunto de reglas para definir etiquetas semánticas que nos organizan un documento en diferentes partes. XML es un metalenguaje que define la sintaxis utilizada para definir otros lenguajes de etiquetas estructurados.

En teoría HTML es un subconjunto de XML especializado en presentación de documentos para la web, mientras que XML es un subconjunto de SGML especializado en la gestión de información para la web. En la práctica XML contiene a HTML aunque no en su totalidad. La definición de HTML contenido totalmente dentro de XML y por lo tanto que cumple la especificación SGML, es XHTML (Extensible Hypertext Markup Language).

XML fue creado al amparo del Word Wide Web Consortium (W3C) organismo que vela por el desarrollo de WWW partiendo de las amplias especificaciones de SGML. Su desarrollo se comenzó en 1996 y la primera versión salió a la luz el 10 de febrero de 1998. La primera definición que apareció fue: "sistema para definir validar y compartir formatos de documentos en la web."

Respecto a sus objetivos son:

— XML debe ser directamente utilizable sobre Internet.

— XML debe soportar una amplia variedad de aplicaciones.

— XML debe ser compatible con SGML.

— Debe ser fácil la escritura de programas que procesen documentos XML.

— El número de características opcionales en XML debe ser mínima.

— Los documentos XML deben ser legibles por humanos y lo más claros posible.

— El diseño de XML debe ser preparado rápidamente.

— El diseño de XML debe ser formal y conciso.

— Los documentos XML deben ser fácilmente creables.

— La concisión en las marcas XML es de mínima importancia.

Estamos ante una herramienta para definir estructuras de datos susceptibles de ser procesadas por una gran variedad de aplicaciones para realizar un eficiente intercambio electrónico de datos, es un lenguaje orientado a identificar estructuras de datos en un documento. La especificación XML define la manera estándar de cómo hay que realizar el marcado de expresiones en un documento no estructurado, para que con dicho marcado se defina una determinada estructura de datos. Como ya hemos comentado, un documento XML es un documento que puede ser leído y entendido por una persona y a la vez puede ser procesado por un sistema para extraer información.

8.1.3. Tipos de documentos XML

Para que un documento sea compatible con XML tiene que cumplir dos condiciones: Ha de ser un documento conforme *(well formed)* y válido *(valid)*.

El hecho de imponer condiciones es importante para que XML se comporte efectivamente como una base universal para estructurar la información. El documento será conforme cuando cumpla una serie de normas que veremos más adelante.

Para evitar errores se ha creado el sistema DTD o (*Document Type Definition*, Definición de Tipos de Documento). En un documento DTD se indica qué códigos contiene el documento XML, en qué orden aparecen, cómo se anidan unos dentro de otros y si pueden estar vacíos o no, entre otras cosas. Los documentos que se ajustan a su DTD son válidos.

En el caso de HTML existe un DTD universal en el que se definen con claridad las etiquetas HTML ya conocidas y sus contenidos. Para XML, cada aplicación puede generar sus propios tipos de datos y códigos, con lo que es preciso generar un DTD junto con cada documento o aplicación. Una empresa puede utilizar su propio DTD.

Por tanto los documentos XML pueden construirse para ser documentos válidos o para ser documentos bien formados. Si hablamos de un documento válido, es que este documento está asociado a un conjunto de reglas que definen su estructura lógica. El documento se certifica conforme a estas reglas que en conjunto reciben el nombre de "Definición de tipo de documento" (DTD). Cuando hablamos de un documento bien formado, manifestamos que este sigue las reglas de sintaxis especificadas para el lenguaje XML, pero no dispone de reglas de certificación asociadas. Un documento bien formado puede ser muy simple, todo lo que necesita contener son elementos de datos.

8.1.4. XML familia de tecnologías

Aunque una de las principales funciones con las que nace sería suceder al HTML, separando la estructura del contenido y permitiendo el desarrollo de vocabularios modulares, compatibles con cierta unidad y simplicidad del lenguaje (objetivo que se viene desarrollando a través de la especificación XHTML), tiene otras aplicaciones entre las que destaca su uso como estándar para el intercambio de datos entre diversas aplicaciones o software con lenguajes privados como en el caso del SOAP.

Al igual que el HTML, se basa en documentos de texto plano en los que se utilizan etiquetas para delimitar los elementos de un documento. Sin embargo, XML define estas etiquetas en función del tipo de datos que está describiendo y no de la apariencia final que tendrán en pantalla o en la copia impresa, además de permitir definir nuevas etiquetas y ampliar las existentes.

Son varios los vocabularios desarrollados en XML con el fin de ampliar sus aplicaciones. Podemos considerar fundamentales: **XHTML, XSL-FO y XSLT, XLink, XPointer y Schema**. Estos serán estudiados en apartados posteriores. Pero, además, existen también versiones para usos específicos, ya que XML facilita incluso la accesibilidad y esta adaptado para cambiar la salida de texto a voz y traducir a Braille. Otras versiones destacables son:

— **MathML**. Fórmulas matemáticas, utilizando etiquetas extensibles que permiten que las representaciones matemáticas sean visualizadas por el navegador que lo admita.

— **SVG**. Gráficos vectoriales. Utilizando elementos que definan formas, rutas y texto, permite que se escriba codigo en lugar de crear grandes mapas de bits.

— **RSS**. Sindicación de noticias

— **XBRL**. Partes financieros

— **SMIL**. Lenguaje de Integración Multimedia Sincronizada, puede utilizarse para crear contenido web interactivo. Permite combinar audio, video y gráficos que pueden enviarse al navegador y pueden representarse rápidamente.

— **WML**. Aplicaciones para móviles para aplicaciones inalámbricas (Wireless)

— Por supuesto las organizaciones o empresas pueden crearse su propia versión de este lenguaje extensible y personalizarlo para su negocio.

  • Si los utiliza un video club para guardar la información de las películas, este lenguaje se podría llamar PeliculasML.

  • Si lo utiliza un hospital para almacenar diagnósticos de los pacientes, este lenguaje se podría llama HospitalML.

***XML** es un método para introducir datos estructurados en un fichero de texto. Cuando pensamos en "datos estructurados" pensamos en cosas tales como hojas de cálculo, libretas de direcciones, parámetros de configuración, transacciones financieras, dibujos técnicos, etc. Los programas que producen*

esta clase de datos a menudo también los guardan en disco, por lo que pueden usar tanto un formato binario como un formato texto. El último formato te permite, si es necesario, ver los datos sin el programa que los ha producido. XML consiste en una serie de reglas, pautas, convenciones para planificar formatos texto para tales datos, de manera que produzcan archivos que sean fácilmente generados y leídos (por un ordenador) que son inequívocos, y que evitan escollos comunes como la falta de extensibilidad, falta de soporte para la internacionalización o localismo, y la dependencia de una determinada plataforma.

XML define cuales son las "tags" (etiquetas) y "atributos", pero alrededor de XML hay una creciente serie de módulos opcionales que ofrecen colecciones de etiquetas y atributos, o pautas para especificar tareas. Existen:

— **Xlink** que describe una manera estándar de añadir hiper-enlaces a un archivo XML.

— **XFragments** son sintaxis para apuntar a partes de un documento XML.

— **XPointer**. Lenguaje de Direccionamiento XML, es un lenguaje que permite el acceso a la estructura interna de un documento XML, esto es, a sus elementos, atributos y contenido

— **CSS**, el lenguaje de hojas de estilo, se puede aplicar a XML igual que a HTML.

— **XSL** es el lenguaje avanzado para explicitar hojas de estilo. Está basado en **XSLT**, un lenguaje de transformación a menudo útil también fuera de XSL, para reordenar, añadir o borrar etiquetas y atributos. XSL-FO.–es un metalenguaje de marcado (denominado también lenguaje de anotaciones o de etiquetas) que nos permite definir lenguajes de marcado adecuados a usos determinados–.

— El **DOM** es una serie de funciones estándar llamadas para manipular archivos XML (y HTML) desde un lenguaje de programación.

— **XML Namespaces** es una especificación que describe cómo puedes asociar una URL con cada etiqueta y atributo en un documento XML, si bien, para qué se utiliza la URL depende de la aplicación que lea la URL.

— **XML Schemas 1** y **2** ayuda a los desarrolladores a definir precisamente sus propios formatos basados en XML.

— **XPath**. Lenguaje de Rutas XML, es un lenguaje para acceder a partes de un documento XML. Es un lenguaje (basado en xml) que permite seleccionar subconjuntos de un documento xml. La idea es parecida a las expresiones regulares para seleccionar partes de un texto sin atributos (plain text). Xpath permite buscar y seleccionar teniendo en cuenta la estructura jerárquica del XML.

— **XQL**. Lenguaje de Consulta XML, es un lenguaje que facilita la extracción de datos desde documentos XML. Ofrece la posibilidad de realizar consultas flexibles para extraer datos de documentos XML en la web.

Como vemos, se pueden crear infinitos lenguajes a partir del XML. Para especificar cada uno de los usos de XML, o lo que es lo mismo, para especificar cada uno de los sublenguajes que podemos crear a partir de XML, se utilizan unos lenguajes propios.

Son unos lenguajes que sirven para definir otros lenguajes, es decir, son metalenguajes. Los definen especificando qué etiquetas podemos o debemos encontrarnos en los documentos HTML, en qué orden, dentro de qué otras, además de especificar los atributos que pueden o deben tener cada una de las etiquetas.

Hay dos metalenguajes con los que definir los lenguajes que podemos obtener a partir de XML, el DTD y el XML Schema. El DTD, Definition Type Document, tiene una sintaxis especial, distinta de la de XML. Para evitar el DTD, que tiene una sintaxis muy especial, se intentó encontrar una manera de escribir en XML la definición de otro lenguaje XML. Se definió entonces el lenguaje XML Schema.

Un detalle importante de señalar a la hora de hablar de los DTD o XML Schema es que estos lenguajes también permiten comprobar la integridad de los datos en cualquier momento. Los metalenguajes de XML sirven para tomar un documento XML y comprobar que los datos que él incluye son válidos, comprobando si lo que tenemos en el XML concuerda con lo que tendríamos que tener. Eso lo podemos hacer al leer el documento, si no son validos se saca un mensaje de error y se detiene el proceso del documento.

8.1.5. Principales características de XML

— Es una arquitectura más abierta y extensible. No se necesita versiones para que puedan funcionar en futuros navegadores. Los identificadores pueden crearse de manera simple y ser adaptados en el acto en Internet/Intranet por medio de un validador de documentos (parser).

— Mayor consistencia, homogeneidad y amplitud de los identificadores descriptivos del documento con XML (los RDF Resource Description FrameWork), en comparación a los atributos de la etiqueta <META> del HTML.

— Integración de los datos de las fuentes. Se podrá hacer el intercambio de documentos entre las aplicaciones tanto en el propio PC como en una red local o extensa.

— Datos compuestos de múltiples aplicaciones. La extensibilidad y flexibilidad de este lenguaje nos permitirá agrupar una variedad amplia de aplicaciones, desde páginas web hasta bases de datos.

— Gestión y manipulación de los datos desde el propio cliente Web.

— Los motores de búsqueda devolverán respuestas más adecuadas y precisas, ya que la codificación del contenido web en XML consigue que la estructura de la información resulte más accesible.

— Se desarrollarán de manera extensible las búsquedas personalizables y subjetivas para robots y agentes inteligentes. También conllevará que los clientes web puedan ser más autónomos para desarrollar tareas que actualmente se ejecutan en el servidor.

— Se permitirá un comportamiento más estable y actualizable de las aplicaciones web, incluyendo enlaces bidireccionales y almacenados de forma externa.

— El concepto de hipertexto se desarrollará ampliamente (permitirá denominación independiente de la ubicación, enlaces bidireccionales, enlaces que pueden especificarse y gestionarse desde fuera del documento, hiperenlaces múltiples, enlaces agrupados, atributos para los enlaces, etc. Creado a través del Lenguaje de enlaces extensible (XLL).

— Exportabilidad a otros formatos de publicación (papel, web, CD-ROM, etc.). El documento maestro de la edición electrónica podría ser un documento XML que se integraría en el formato deseado de manera directa.

— La gramática usada en HTML es fija y no ampliable. En XML es extensible.

— La estructura en HTML es monolítica y en XML es jerárquica.

— El estilo en HTML viene dado por CSS o por las propias etiquetas, en XML por CSS o por XSL, en SGML por DSSSL.

— HTML no es exportable, XML si lo es.

— HTML no permite validación, XML puede validarse, SGML es obligatorio.

8.1.6. Estructura del XML

El *metalenguaje* XML consta de cuatro especificaciones (el propio XML sienta las bases sintácticas y el alcance de su implementación):

DTD *(Document Type Definition)*: definición del tipo de documento. Es, en general, un archivo que encierra una definición formal de un tipo de documento y , a la vez, especifica la estructura lógica de cada documento. Define tanto los elementos de una página como sus atributos. El DTD del XML es opcional. En tareas sencillas no es necesario construir una DTD, entonces se trataría de un documento "bien formado" *(well-formed)* y si lleva DTD será un documento “validado” *(valid)*.

XSL *(eXtensible Stylesheet Language)*: define o implementa el lenguaje de estilo de los documentos escritos para XML. Desde el verano de 1997 varias empresas informáticas como Arbortext, Microsoft e Inso vienen trabajando en una propuesta de XSL (antes llamado “xml-style”) que presentaron a W3C. Permite modificar el aspecto de un documento. Se puede lograr múltiple columnas, texto girado, orden de visualización de los datos de una tabla, múltiples tipos de letra con amplia variedad en los tamaños. Este estándar está basado en el lenguaje de semántica y especificación de

estilo de documento (DSSSL, Document Style Semantics and Specification Language, ISO/IEC 10179) y, por otro lado, se considera más potente que las hojas de estilo en cascada (CSS), usado en un principio con el lenguaje DHTML. "Se espera que el CSS sea usado para visualizar simples estructuras de documentos XML (actualmente se ha conseguido mayor integración en XML con el protocolo CSS2 (Cascading Style Sheets, nivel 2) ofreciendo nuevas formas de composición y una más rápida visualización) y, por otra parte, XSL pueda ser utilizado donde se requiera más potencia de diseño como documentos XML que encierran datos estructurados (tablas, organigramas, etc.)".

XLL *(eXtensible Linking Language)*: define el modo de enlace entre diferentes enlaces. Se considera que es un subconjunto de HyTime *(Hipermedia/Timed-based structuring Language* o Lenguaje de estructuración hipermedia/basado en el tiempo, ISO 10744) y sigue algunas especificaciones del TEI *(Text Encoding Initiative* o Iniciativa de codificación de texto).

Desde marzo de 1998 el W3C trabajó en los enlaces y direccionamientos del XML. Provisionalmente se le renombró como Xlink y a partir de junio se le denomina XLL. Este lenguaje de enlaces extensible tiene dos importantes componentes: Xlink y el Xpointer. Va más allá de los enlaces simples que solo soporta el HTML. Se podrá implementar con enlaces extendidos. Jon Bosak establece los siguientes mecanismos hipertextuales que soportará esta especificación:

— Denominación independiente de la ubicación.

— Enlaces que pueden ser también bidirecccionales.

— Enlaces que pueden especificarse y gestionarse desde fuera del documento a los que se apliquen (esto permitirá crear en un entorno intranet/extranet un banco de datos de enlaces en los que se puede gestionar y actualizar automáticamente.

— Hiperenlaces múltiples (anillos, múltiples ventanas, etc.).

— Enlaces agrupados (múltiples orígenes).

— Transclusión (el documento destino al que apunta el enlace aparece como parte integrante del documento origen del enlace).

— Se pueden aplicar atributos a los enlaces (tipos de enlaces).

8.1.7. Documentos XML

Como XML es un metalenguaje, es posible crear códigos o etiquetas (tags) a gusto del usuario para aplicaciones concretas. Ya no solo es posible definir el formato del texto o la posición de los datos y las imágenes, sino que cada parte de la información puede ser identificada con un sistema propio de etiquetas.

Ejemplo:

```
<?xml version="1.0"?>
<empresa>
    <departamento>
          <empleado>
                <nombre>Mickey Mouse</nombre>
                <sueldo>900</sueldo>
                <provincia>Madrid</provincia>
          <empleado>
          <empleado>
                <nombre>Donald</nombre>
                <sueldo>950</sueldo>
                <provincia>Pontevedra</provincia>
          </empleado>
     </departamento >
    <departamento>
          <empleado>
                <nombre>Mynie</nombre>
                <sueldo>1100</sueldo>
                <provincia>Barcelona</provincia>
          <empleado>
          <empleado>
                <nombre>Deysi</nombre>
                <sueldo>850</sueldo>
                <provincia>Sevilla</provincia>
          </empleado>
     </departamento >
</empresa >
```

La idea que subyace bajo el XML es la de crear un lenguaje muy general que sirva para muchas cosas. El HTML está diseñado para presentar información directamente a los humanos, es un lenguaje complicado de procesar para los programas informáticos. El HTML no es bueno porque no indica lo que está representando, se preocupa principalmente del formato. El XML describe el contenido de lo que etiqueta.

Esto permite, por ejemplo, realizar motores de búsqueda mucho más eficaces, lo que nos permitirá un acceso más rápido y eficiente a la información.

La potencia de esta forma de trabajar radica en que estamos etiquetando e identificando el contenido, olvidándonos en un principio por la forma de presentarlo. Mediante una XSL podremos transformar un document XML en otro XML (por ejemplo en HTML) o convertirlo a un formato de impresión: RTF, PDF, etc. El XML supone una revolución porque permite la comunicación entre las máquinas.

8.1.8. Parsers XML

Un parser o procesador de XML es la herramienta principal de cualquier aplicación XML. Mediante este parser no solo podremos comprobar si nuestros documentos están bien formados o válidos, sino que también podremos incorporarlos a nuestras aplicaciones, de manera que estas puedan manipular y trabajar con documentos XML. Actualmente hay muchos y para todos los lenguajes y plataformas: JAVA, C, Phyton, Visual Basic, Perl, Tcl, Delphi, etc. El XML contribuye con datos independientes de la plataforma (documentos y datos portables). Todas las grandes compañías ya han elaborado sus propios procesadores de XML. Y existen muchos más gratuitos.

8.2. XML

XML *(eXtensible Markup Language* o Lenguaje de Marcado Extensible) viene a ser una versión reducida de SGML, en el cual se basa, y está especialmente diseñado para la definición de estructuras de documentos y el almacenamiento de datos. Podemos decir que XML se puede utilizar para el desarrollo de dos tipos de aplicaciones:

— Aplicaciones de datos.

— Aplicaciones de documento.

Las recomendaciones XML 1.0 del W3C definen los documentos XML como un tipo de objetos de datos que están formados físicamente por unidades de almacenamiento, denominadas entidades, cuyo contenido puede ser datos analizados y no analizados. Los datos analizados (PCDATA, Parsed Character Data) están compuestos por datos de caracteres (character data) y marcas (markup). Las marcas son símbolos especiales utilizados por el lenguaje XML para indicar que el texto que viene a continuación debe ser procesado, ya que se trata de una etiqueta, marcas < y >, o una referencia, marca &. Las marcas permiten establecer la estructura lógica y de almacenamiento del documento. Los datos no analizados o datos de caracteres (CDATA, Carácter Data) representarían datos de contenido textual que no han de ser analizados por el procesador XML.

XML, además, proporciona un mecanismo para imponer restricciones al almacenamiento de los datos y a la estructura lógica del documento. Una unidad de almacenamiento XML se puede considerar como documento XML si está bien formado. Un documento XML bien formado puede además ser válido si cumple una serie de restricciones definidas en una DTD o Schema.

XML es, por lo tanto, un lenguaje diseñado para trabajar con datos y estructuras; podríamos definir, teniendo en cuenta lo visto anteriormente, a los documentos XML como "objetos de datos cuya estructura está bien formada".

Las aplicaciones destinadas a trabajar con documentos XML han de incorporar un módulo de software especial denominado Procesador XML, que permitirá leer el documento XML y acceder a su estructura y contenido (en caso de que dicho documento esté como mínimo bien formado).

Por tanto los documentos XML, teniendo en cuenta el grado de restricción utilizado para establecer la estructura y contenido de sus datos, pueden ser de dos tipos: bien formados y válidos o validados.

8.2.1. Documentos XML bien formados

Los documentos XML se basan en una estructura en forma de árbol en la cual los elementos se encuentran distribuidos jerárquicamente. En esa estructura siempre existe un elemento raíz (root) que representa el elemento documento (o entidad documento).

Las seis **normas básicas** que debe cumplir todo documento XML para que sea considerado como bien formado y, por lo tanto, su estructura sea correcta son:

1. Un documento XML solamente puede tener un elemento raíz (root). La entidad documento está representada por el denominado elemento documento (Document). Es el nodo raíz a partir del cual se irán añadiendo el resto de entidades o elementos (nodos hijo) que conforman el documento.

2. Todos los elementos con contenido de un documento XML han de tener etiquetas de cierre, es decir, siempre que se utilice la etiqueta de un determinado elemento, hemos de utilizar también su etiqueta de cierre: <etiqueta> ... </etiqueta>. Debemos tener en cuenta que las etiquetas vacías utilizan una sola etiqueta que incluye su propio cierre por medio del carácter "/" situado al final: <etiqueta/>.

3. Las etiquetas de distintos elementos no se pueden solapar o superponer, es decir, no se puede mezclar el contenido de un elemento con el de otro.

4. Las etiquetas de elementos anidados se deben cerrar en el orden correcto, es decir, no se pueden mezclar las etiquetas de elementos de orden superior (elementos padre) con los de orden inferior (elementos hijo).

5. Los atributos de los elementos XML deben ir siempre entre comillas; se pueden utilizar comillas dobles ("") o simples (´).

6. No se pueden utilizar los caracteres <,> o & en el texto del contenido de un elemento, ya que representan marcas del lenguaje XML; se ha de recurrir a las entidades XML < > o &.

8.2.2. Documentos XML válidos

Además de ser documentos bien formados, utilizan un mecanismo de descripción de documentos, ya sea DTD o Schema, en el cual se definen los elementos que se han de usar y el modo de hacerlo (sintaxis), así como la estructura que han de mantener.

Un documento XML basado en DTD *(Document Type Definition* o Definición de Tipo de Documento) utiliza una declaración DOCTYPE en el

prólogo del documento indicando cuál es el elemento documento (elemento raíz) y la definición del resto de los elementos que se pueden utilizar en ese tipo de documentos.

Por tanto, en función de si lleva asociada una DTD o no, podemos diferenciar dos tipos de documentos XML:

— **Válidos**, aquellos que siguen las reglas de una DTD especifica.

— **Bien formados** (well-formed), que no tienen necesariamente una DTD asociada, pero siguen las reglas del XML al pie de la letra.

Evidentemente, los documentos válidos son bien formados.

8.2.3. Estructura básica de un documento XML

La estructura lógica de un documento XML esta definida por declaraciones, elementos, comentarios, referencias e instrucciones de procesamiento. Todo documento XML consta de dos partes: el prólogo y el cuerpo del documento.

- **Prólogo**

Contiene datos con meta información utilizada por el procesador XML y generalmente, está formado por:

— La línea de declaración de XML.

— Instrucciones de procesamiento.

— Declaración DOCTYPE.

— Comentarios.

La línea de declaración de documento XML

Es una instrucción de procesamiento y la primera línea que debe aparecer en un documento XML. Usar la línea de declaración de documento XML es opcional; no obstante, puede prevenir contra posibles errores producidos al intentar abrir el documento. Si se utiliza, será la primera línea del código en el prólogo del documento y es una instrucción similar a la siguiente:

Parámetros

— **Version**: indica la versión XML en la cual se basa del documento. En caso de utilizarse la línea de declaración de documento XML, este parámetro es de uso obligatorio.

— **Encoding**, tipo de codificación utilizado por los caracteres del documento.

— **Standalone**, para especificar si se utiliza una DTD externa o no. En caso de utilizarse estos dos últimos parámetros, han de indicarse en ese orden. Posibles valores del atributo standalone:

  • **Yes:** la especificación DTD se encuentra dentro del propio documento; no se permite el uso de DTD externa.

  • **No:** la especificación DTD se encuentra en un documento externo.

En caso de que el parámetro standalone sea "no", se pueden utilizar conjuntamente DTD interna y externa.

• **Cuerpo del documento**

Representado por el elemento raíz (objeto o entidad documento) y el resto de los elementos y contenido:

8.2.4. Tipos de datos XML

Como ya hemos visto, el texto de un documento XML está formado por una mezcla de datos de caracteres y marcas. Este texto es analizado por el procesador de XML; para comprobar la estructura y buena formación del documento, no obstante, pueden existir fragmentos que no se desee que pasen dicho análisis. Podemos decir, por lo tanto, que un documento XML puede contener datos que han de ser analizados han de ser marcados de una manera especial, es decir, utilizando unas determinadas marcas del lenguaje XML.

Datos no analizados

Son datos incluidos dentro del documento XML y que son ajenos a la normativa de XML, es decir, no tienen por qué ser analizados para comprobar su estructura o sintaxis por el procesador de XML y son enviados tal como están escritos, como simples datos de caracteres (CDATA). Pertenecen a este grupo:

— Comentarios: <! - - comentario - - >

— Secciones CDATA:<![CDATA[esto es una sección CDATA]]>

— Instrucciones de procesamiento (PI, Proccessing Instruction)

Datos analizados

Por defecto, todo el contenido de un documento XML ha de ser analizado por el procesador de XML al abrir el documento para comprobar su buena formación. Todos los datos analizados (PCDATA) son de tipo texto y se dividen, como ya hemos visto, en datos carácter y marcas.

Datos carácter

Son cualquier cadena de caracteres, excepto los utilizados como delimitadores de marcas, caracteres < >, ampersand (&), utilizado por las entidades y los caracteres apóstrofo (´) y comillas (“) utilizados en los valores de atributos. En caso de querer utilizar estos caracteres como texto, se hará como caracteres escapados, mediante referencias entre los símbolos ampersand (&) y punto y coma (;), pudiéndose utilizar dos métodos:

— Referencias numéricas por código, con la sintaxis &#valor; (numeración decimal o &#xvalor; (numeración hexadecimal).

— Referencia a entidades por nombre como, por ejemplo, < (<), & (&).

Marcas

Los símbolos de marcas utilizadas en un documento XML pueden ser:

— **De etiquetas**:

- Marca de comienzo de etiqueta(<) y marca de fin de etiqueta(>).
- Marcas de etiquetas de elementos vacíos (</>).

— **De referencias**:

- Referencias a entidades: &NombreEntidad;.
- Referencias a códigos de caracteres: &#xCódigo; o &#Código;.

— **De valores de los atributos**:

- Apóstrofo (´) o comilla simple: ´valor´.
- Comillas dobles (“): “valor”.

— **Delimitadores: utilizados en listas de elementos de contenido, generalmente:**

- Coma (,).
- Barra inclinada (|).

Existen además marcas especiales para establecer:

— Declaraciones de tipo de documento <!DOCTYPE ElementoRaiz>

— Comentarios:<!- - comentario - - >

— Secciones CDATA: <![CDATA[.....datos no analizados.....]]>

— Instrucciones de procesamiento:<?... instrucciones....?>

Atributos especiales XML

Las etiquetas de los elementos XML pueden utilizar una serie de atributos especiales definidos para este lenguaje y que todo procesador XML debe implementar.

— **Xml:space**

Sirve para establecer si se han de preservar los espacios utilizados por el contenido de un determinado elemento o no. En caso de utilizarse este atributo en un DTD ha de declararse de tipo enumerado y su único valor posible es default y preserve. La declaración para este atributo en un DTD es, por lo tanto, xml:space(default|preserve).

— **Xml:lang**

Sirve para especificar el lenguaje utilizado por los contenidos y los valores de atributos de cualquier elemento en un documento XML. Los valores de este atributo son identificadores de lenguajes definidos en el documento "Etiquetas para la Identificación de Lenguajes" [IETF RFC 1766].

Marcas XML

En los documentos XML podemos encontrar las siguientes marcas utilizadas por este lenguaje:

— **Etiquetas**: < > </ > y < />

Las etiquetas representan elementos XML; su nombre se encierra entre la marca de comienzo de etiqueta (<) y la marca de fin de etiqueta (>) para la etiqueta inicial y las marcas </ y > para la etiqueta final. En caso de que sean elementos vacíos, las marcas de etiquetas son < (marca inicial) y /> (marca final).

Sintaxis:

- Elemento con contenido: <etiqueta>... contenido..</etiqueta>.
- Elemento sin contenido:<etiqueta/>.

— **Referencias**: &

Como ya hemos visto, pueden existir referencias a caracteres (por código) o a entidades (por nombre). El valor utilizado como referencia va entre las marcas ampersand (&) y punto y coma (;).

En caso de ser una referencia a caracteres, se utilizará además el símbolo de almohadilla (#) antes del valor del código si va en notación decimal &#Valor; ejemplo: €. Se pueden utilizar también valores en hexadecimal; en este caso se pondrá una equis (x) tras el símbolo # y delante del valor hexadecimal &#xValor; ejemplo: €. Si es una referencia a una entidad, simplemente se ha de poner el nombre de dicha entidad &NombreEntidad; ejemplo:&.

Al encontrarse una referencia, el procesador XML lo que hace es insertar el carácter o expandir la entidad referenciada; debemos tener en cuenta que una entidad no tiene por qué ser un carácter simplemente, sino que puede ser incluso un documento completo o archivo externo.

— **Comentarios**: <!–– - - >

Los comentarios se sitúan entre las marcas <!—(inicio de comentario) y - -> (final de comentario) de forma similar a los utilizados por SGML y HTML. Los comentarios incluyen datos que no serán analizados por el procesador de XML. Generalmente son utilizados para documentar el código del documento XML y pueden incluir textualmente todo tipo de caracteres, incluidos los utilizados como marcas. No están permitidas las cadenas de caracteres de dos guiones (––) y la secuencia utilizada para finalización de comentario (- - >)

— **Secciones CDATA**: <[CDATA[“...”]]>

El contenido de estas secciones va entre las cadena de inicio “<[CDATA[“ y finalización”]]>”.

Al igual que los comentarios, sirven para insertar en el documento XML datos que no han de ser analizados por el procesador XML. Su contenido, como su nombre indica, es texto (CDATA, Character Data o Datos de Caracteres) en el cual se puede utilizar cualquier tipo de carácter, incluso los usados para marcas. No está permitido el uso de la secuencia de caracteres “]]>” usado para indicar la finalización de la sección CDATA.

Generalmente se utilizan estas secciones para incluir scripts, estilos, bloques de código HTML, documentación y comentarios en los cuales se incluyen los caracteres de marcas ya que, dentro de la sección, son interpretados como datos carácter y no como marcas de etiquetas.

— **Declaraciones de tipo de documento**: <!DOCTYPE...>

Las declaraciones de tipo de documento (DOCTYPE) se utilizan para declarar documentos XML válidos, es decir, que el documento utilizará una definición de tipo de documento o DTD.

Los parámetros utilizados en una declaración DOCTYPE, por lo tanto, sirven para establecer cuál es el elemento raíz del documento y la DTD que se ha de utilizar para efectuar su validación. La línea de declaración DOCTYPE debe aparecer justamente antes del primer elemento utilizado.

La sintaxis genérica de una declaración DOCTYPE es la siguiente:

Sintaxis:<!DOCTYPE ElementoRaíz [DTD]>

Ejemplo: <!DOCTYPE empleado SYSTEM=”empleado.dtd”>

El parámetro elementoRaíz representa al elemento documento; este elemento es único, debe existir obligatoriamente en todo documento XML y es la base (raíz) de toda la estructura del documento; así, por ejemplo, en el documento biblioteca con la siguiente estructura:

```
<empleado>
      <nombre>Popeye</nombre>
      <apellidos> El Marino</apellidos>
</empleado>
```

El elemento raíz (entidad documento) está representado por las etiquetas <empleado> </empleado>.

El parámetro DTD sirve para indicar la Definición de Tipo de Documento en la cual se basa el documento; el valor de este parámetro puede ser:

- El código de la propia DTD entre corchetes: [contenido del DTD]. La declaración DOCTYPE para el ejemplo anterior sería, por lo tanto:

```
<!DOCTYPE empleado[
            <!ELEMENT empleado (nombre, apellidos)>
            <!ELEMENT nombre (#PCDATA)>
            <!ELEMENT apellidos (#PCDATA)>
      ]>
```

- SYSTEM “archivo.dtd”, si se utiliza una DTD externa almacenada en un archivo independiente en el mismo equipo. Si hemos guardado en el mismo directorio que el documento XML la DTD vista en el ejemplo anterior con el nombre de “empleado.dtd”, la declaración DOCTYPE sería:

```
<!DOCTYPE empleado SYSTEM “empleado.dtd”>
```

- PUBLIC “ID” “URI”, si se utiliza una DTD externa almacenada en un archivo independiente en un equipo existente en la red Internet. ID hace referencia a un identificador público utilizado por el DTD y URI es el localizador, la dirección donde se encuentra la DTD.

8.2.5. Instrucciones de procesamiento: <? ... ?>

Las instrucciones de procesamiento (IP, *Processing Instructions)* van situadas entre las marcas >? Y ¿> y sirven para pasar información que no ha de ser analizada por el procesador de XML. Pueden ir situadas en cualquier

lugar del documento, aunque, por lo general, van entre las líneas del prólogo. Se denominan instrucciones de procesamiento porque suelen contener instrucciones para determinadas aplicaciones; por ello, comienzan con el nombre identificador de la aplicación destino (PITarget) a la cual va dirigida dicha instrucción.

Las instrucciones de procesamiento generalmente se utilizan para efectuar enlaces con entidades externas o recursos que van a ser utilizados por el documento XML, tales como las hojas de estilos CSS y XSL. Ejemplo de instrucciones de procesamiento utilizadas con hojas de estilo:

```
<?xml-stylesheet type="text/css"href="estilos.css"?>
<?xml-stylesheet type="text/xsl"
href="transformaciones.xsl"?>
```

Sintaxis genérica para instrucciones de procesamiento utilizadas con hoja de estilo es:

```
<?xml-stylesheet type="tipo" href="uri" [title="nombre"]?>
```

8.2.6. Normas para el uso de etiquetas XML

1) Las etiquetas (tags) aparecen marcadas por un carácter de inicio > (menor que) y otro de finalización > (mayor que). Ejemplo:<etiqueta>.

2) Las etiquetas XML deben comenzar por una letra o espacio de subrayado ("_"); no pueden comenzar por <xml (nombre de espacio) ni caracteres numéricos. Pueden ir en mayúsculas, minúsculas o la combinación de ambas. Los nombres de las etiquetas son sensibles a mayúsculas y minúsculas, por lo cual <ETIQUETA>, <Etiqueta> y <etiqueta> representan elementos distintos. Por convención se recomienda el uso de minúsculas, pero, en caso de utilizar etiquetas HTML en documentos XML, se deberían escribir en mayúsculas.

3) La etiquetas pueden ser nombres compuestos que han de separarse por un guión, punto o carácter de subrayado, pero no se admiten los espacios. Ejemplo: <etiqueta-compuesta>. Se recomienda el uso de mayúsculas iniciales y no utilizar los símbolos antes citados: <EtiquetaCompuesta>.

4) Los documentos XML han de utilizar siempre para cada elemento con contenido una etiqueta de apertura y otra de cierre. Ambas utilizan el mismo nombre, pero en la de finalización va precedida por el símbolo/. Ejemplo: <etiqueta>contenido>/etiqueta>.

5) Pueden existir elementos "vacíos", sin etiqueta de cierre; en este caso se ha colocar el carácter / al final de ella. Ejemplo: <Elemento-Vacio />. Debemos tener en cuenta que, en realidad, una etiqueta

"vacía" puede contener datos, los cuales podrían ser considerados como elementos individuales dentro del documento, pero estos son utilizados como atributos en la etiqueta vacía. Así, si tenemos un elemento empleado, con los elementos nombre y apellidos: empleado (nombre, apellidos), podríamos utilizarlo de la siguiente forma:

En la misma etiqueta como atributos:

```
<empleado nombre="Popeye" apellidos="El Marino"/>
```

En etiquetas independientes como elementos independientes:

```
<empleado>
      <nombre>Popeye</nombre>
      <apellidos> El Marino</apellidos>
</empleado>
```

6) Las etiquetas de los elementos con contenido también pueden utilizar atributos.

Los valores de los atributos siempre han de ir entre comillas, ya sean simples (apóstrofos) o dobles. En caso de utilizar apóstrofos o comillas dobles dentro de la cadena de texto dada como valor para el atributo, se han de combinar ambas; en cado de utilizarse del mismo tipo, se han de usar las referencias a entidades ' o ".

- Sintaxis general para etiquetas de elementos con contenido:

```
<etiqueta[atributo="valor"]>contenido </etiqueta>
```

- Sintaxis general para etiquetas de elementos sin contenido:

```
<etiqueta[atributo="valor"]/>
```

Ejemplo:

```
  <!— Comentario —>
  <ejemplo>
< dedicatoria para="familia" > Ojana significa familia, estaremos unidos siempre</ dedicatoria >
<imagen src="imagen.gif"/>
<tai>
<![CDATA[
       Vamos a aprobar todos este año.
] ]>
</tai>
  </ejemplo>
```

Un documento XML es un conjunto de cadenas de caracteres, en el que, al igual que en el HTML, podemos diferenciar **dos tipos de construcciones: el marcado y los datos de carácter**. El texto incluido entre los caracteres menor que **<** y mayor que **>** o entre los signos **&** y **;** es el marcado. Son exactamente las partes del documento que tiene que entender el procesador de XML. El marcado entre los signos **<** y **>** se denominan etiqueta. El resto no son más que datos de carácter, que se corresponde con lo que sería el contenido del documento: es decir, la parte imprimible de este.

En el caso de **elementos con contenido**, las etiquetas de comienzo se componen del símbolo menor que **<**, el nombre del tipo de elemento, los atributos si los tiene y el símbolo mayor que **>**. Mientras que las etiquetas de fin se componen del símbolo menor que seguido de contrabarra **</**, el nombre del tipo del elemento y el símbolo mayor que **>**.

En el caso de ser un **elemento vacío**, solo hay una etiqueta de elemento vacío que se forma del símbolo menor que **<**, el nombre del tipo de elemento, los atributos si los tiene y se cierra con el símbolo **/>**. Es importante destacar este tipo de elementos, ya que hasta ahora en el SGML y, por tanto en el HTML entendido como aplicación SGML, los elementos vacíos solo se representaban con una etiqueta de inicio.

A diferencia del SGML, no es necesario que un documento XML esté asociado a una DTD.

8.3. CSS

La técnica más sencilla de publicar documentos XML en Internet consiste en aplicar en este tipo de documentos las denominadas Hojas de Estilo en Cascada (CSS, *Cascading Style Sheets)* también veremos otras posibilidades basadas en tecnologías propiamente XML como son la utilización de XSL y XSLT para su publicación.

No debemos olvidar que XML no es un lenguaje diseñado para la publicación de documentos y, por lo tanto, ha de recurrir al uso de alguna otra tecnología para poder mostrar los contenidos de los documentos. La técnica XML-CSS permite al diseñador de documentos utilizar al máximo su capacidad creativa. Con XML controlará la estructura y elementos (etiquetas) que van a ser usados en el documento y con CSS definirá de forma totalmente personal qué estilos serán utilizados por cada uno de esos elementos.

La definición de las normas o reglas de estilo sigue la misma sintaxis:

```
Selector { propiedad1: valor1 [; propiedad2: valor2:…] }
```

La principal característica de las hojas de estilo CSS utilizadas por los documentos XML es que el nombre del selector del estilo debe coincidir con el nombre del elemento XML sobre el cual se va a aplicar, es decir, los nombres de los selectores siempre serán nombres de elementos utilizados en el documento XML.

```
Título{
      Font-family:Comic Sans MS;
```

```
        Font-size: 16pt;
        Color:red;
        Text-align:center;
        Font-weight:bold
}
```

El enlace con las hojas de estilos se efectúa desde los documentos XML, por medio de una instrucción <¿xml-stylesheet...>.

La sintaxis genérica es la siguiente:

```
<¿xml-stylesheet href="estilos.css" type="text/css"?>
```

Ya que XML no es un lenguaje diseñado para la publicación de documentos podemos utilizar otras tecnologías (hojas de estilo) existentes para ese fin. Los estilos que deseemos aplicar a los elementos de los documentos XML, han de ser definidos por medio de las hojas de estilo en cascada (CSS). Los estilos definidos pueden ser aplicados directamente a los elementos existentes en el documento XML. La aplicación de estilos también se puede efectuar utilizando las tecnologías XSL y XSL-FO. XML permite definir etiquetas personalizadas cuya principal característica sea la de aplicar formato o estilos a determinados elementos del documento.

Las **hojas de estilo en cascada** (CSS) son un lenguaje formal usado para definir la presentación de un documento estructurado escrito en HTML o XML (y por extensión en XHTML). El W3C *(World Wide Web Consortium)* es el encargado de formular la especificación de las hojas de estilo que servirá de estándar para los agentes de usuario o navegadores.

CSS no es nada nuevo, ya se podía utilizar con HTML y se creó en un intento de separar la forma del contenido en HTML. En XML también podemos utilizar las CSS, y se utilizan de una manera muy similar a cómo se utilizan en HTML, por lo menos los atributos de estilo que podemos aplicar son los mismos y sus posibles valores también. La idea que se encuentra detrás del desarrollo de CSS es separar la estructura de un documento de su presentación.

Por ejemplo, el elemento de HTML <H1> indica que un bloque de texto es un encabezamiento y que es más importante que un bloque etiquetado como <H2>. Versiones más antiguas de HTML permitían atributos extra dentro de la etiqueta abierta para darle formato (como el color o el tamaño de fuente). No obstante, cada etiqueta <H1> debía disponer de esa información si se deseaba un diseño consistente para una página, y además, una persona que lea esa página con un navegador pierde totalmente el control sobre la visualización del texto.

Cuando se utiliza CSS, la etiqueta <H1> no debería proporcionar información sobre cómo va a ser visualizado, solamente marca la estructura del documento. La información de estilo separada en una hoja de estilo, especifica cómo se ha de mostrar <H1> : color, fuente, alineación del texto, tamaño, y otras características no visuales.

La información de estilo puede ser adjuntada tanto como un documento separado o en el mismo documento HTML. En este último podrían definirse estilos generales en la cabecera del documento o en cada etiqueta particular mediante el atributo style.

Las ventajas de utilizar CSS (u otro lenguaje de estilo) son:

— Control centralizado de la presentación de un sitio web completo con lo que se agiliza de forma considerable la actualización del mismo.

— Los navegadores permiten a los usuarios especificar su propia hoja de estilo local que será aplicada a un sitio web remoto, con lo que aumenta considerablemente la accesibilidad. Por ejemplo, personas con deficiencias visuales pueden configurar su propia hoja de estilo para aumentar el tamaño del texto o remarcar más los enlaces.

— Una página puede disponer de diferentes hojas de estilo según el dispositivo que la muestre o incluso a elección del usuario. Por ejemplo, para ser impresa, mostrada en un dispositivo móvil, o ser leída por un sintetizador de voz.

— El documento HTML en sí mismo es más claro de entender y se consigue reducir considerablemente su tamaño.

Hay varias versiones: CSS1, CSS2 y CSS3. Los navegadores modernos los implementan bastante bien, aunque existen pequeñas diferencias de implementación según marcas y versiones de los navegadores.

8.4. DTD

Siglas de **Document Type Definition**. Un DTD (Definición de tipo de documento) es una colección de reglas usadas con el propósito de identificar un tipo o clase de documento. El DTD realiza las siguientes tareas:

— Define todos los elementos (nombres de etiquetas) que pueden aparecer en el documento.

— Define las relaciones establecidas entre los distintos elementos.

— Suministra información adicional que puede ser incluida en el documento: Atributos, Entidades y Notaciones.

— Aporta comentarios e instrucciones para su procesamiento.

El DTD es una definición en un documento SGML o XML que especifica restricciones en la estructura del mismo. El DTD puede ser incluido dentro del archivo del documento, pero normalmente se almacena en un fichero ASCII de texto separado. La sintaxis de los DTD's para SGML y XML es similar pero no idéntica.

La definición de un DTD especifica la sintaxis de una aplicación de SGML o XML, que puede ser un estándar ampliamente utilizado como XHTML o una aplicación local.

Un DTD describirá típicamente cada elemento admisible dentro del documento, los atributos posibles y (opcionalmente) los valores de atributo permitidos para cada elemento. Es más, describirá los anidamientos y ocurrencias de elementos. La mayoría de DTD se componen generalmente de definiciones de ELEMENT y definiciones de ATTLIST.

Ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ejemplo [
    <!ELEMENT ejemplo ( dedicatoria| imagen |
    tai)>
    <!ELEMENT dedicatoria (#PCDATA)>
    <!ATTLIST dedicatoria
             para CDATA #REQUIRED>
    <!ELEMENT imagen EMPTY>
    <!ATTLIST imagen
             src CDATA #REQUIRED>
    <!ELEMENT tai (#PCDATA)>
]>
<!— Comentario —>
<ejemplo>
< dedicatoria para="familia" > Ojana significa familia, estare-
mos unidos siempre</ dedicatoria >
<imagen src="imagen.gif"/>
<tai>
<![CDATA[
       Vamos a aprobar todos este año.
] ]>
</tai>
</ejemplo>
```

Esto es un ejemplo de documento XML con DTD incorporada, prestamos especial interés en las siguientes características:

— **Utilizamos nuestras propias etiquetas**. Y es que en XML no estamos trabajando con etiquetas predefinidas. Nosotros podemos crearnos nuestro propio lenguaje de etiquetas en función de nuestras necesidades.

— **La sintaxis es estricta**. Ya no vale dejar de entrecomillar los atributos o utilizar las mayúsculas y minúsculas sin ningún control. La especificación XML determina claramente una serie de reglas que especifican cuando un documento está bien formado.

— **La utilización de una DTD**. En HTML, a pesar de ser una aplicación SGML, no era obligatorio utilizarlas y aunque para trabajar con XML tampoco será necesario, sí que será recomendable. Posiblemente no acompañen al documento XML en su distribución, pero resultan muy útiles en la elaboración y validación de los documentos.

— **Los elementos vacíos**. Son los elementos del tipo <img>, <hr>, etc. de HTML, en los que no existe etiqueta final al no tener contenido. Ahora, en el XML, la propia etiqueta de inicio llevará una contrabarra al final que los identificará.

— Es posible guardar el componente DTD en un archivo de texto llamado *ejemplo.dtd* y el fichero xml a un fichero de texto denominado de forma diferente.

Crear documentos XML válidos, es decir, documentos XML basados en la utilización de una definición de tipo de documento (DTD). Se denominan documentos XML válidos a aquéllos que, además de estar bien formados (estructura del documento correcta), siguen las normas sintácticas establecidas en un determinado DTD en el que se indican cuáles son los elementos, atributos, etc., que se deben utilizar y de qué manera.

Las DTD se basan en el lenguaje SGML, es decir, utilizan instrucciones definidas en ese lenguaje, pero existen otros lenguajes de modelado para la creación de documentos de validación y que están basados en XML; son los denominados esquemas (Schema), entre los cuales tenemos XML Schema.

8.4.1. Declaraciones usadas en un DTD

Las posibles declaraciones son cuatro y permiten declarar todos los objetos que pueden ser utilizados por un documento y cómo. Se dividen en:

— Estructura física del documento:

  • Declaración de entidad (ENTITY)

  • Declaración de notación (NOTATION)

— Estructura lógica del documento:

  • Declaración de elemento (ELEMENT)

  • Declaración de atributos (ATTLIST)

Veamos cada uno de ellos en detalle:

8.4.2. Declaración de elementos

En los DTD declaramos los elementos que se han de utilizar en el documento XML por medio de <!ELEMENT>; esta es la forma de establecer los nombres de etiquetas usadas y cuál será su contenido.

La sintaxis genérica para la declaración de elementos es:

```
<!ELEMENT nombreElemento (contenido)>
```

— NombreElemento representa a la etiqueta utilizada por un determinado elemento.

— (Contenido) tipo de dato que puede incluir ese elemento; va entre paréntesis. Aquellos elementos que carezcan de contenido se declaran como EMPTY (vacío).

Contenido de los elementos

— Un texto (datos de caracteres). Datos de caracteres analizados.

— Mixto con texto y otros elementos: los elementos de la lista de contenidos, entre paréntesis, han de ir separados por barra vertical "|". Además, se ha de poner el indicador de número de apariciones de ese elemento en el documento mediante el símbolo de asterisco * colocado fuera de los paréntesis, el cual se utiliza para establecer que esos elementos pueden aparecer cero o varias veces.

Cuando el contenido son otros elementos:

1. Los elementos de la lista han de ir separados por comas "," si todos ellos pueden ser utilizados y siguiendo un determinado orden.

2. Los elementos de la lista han de ir separados por la barra inclinada "|" (OR) si cualquiera de ellos puede ser utilizado.

8.4.3. Declaración de entidad

La entidad que sirve de punto de entrada al procesador XML es la entidad documento (o elemento raíz); desde esta entidad se puede hacer referencia a otras entidades ya sean internas (definidas en el propio documento) o externas. Esas entidades pueden a su vez hacer referencia a otras entidades. El conjunto de todas ellas da lugar al documento XML completo.

Una entidad puede ser desde un simple carácter especial, euro, por ejemplo, hasta archivos externos almacenados en disco de cualquier tipo: documentos de texto, imágenes, sonidos, etc. Se pueden establecer tres tipos de clasificaciones para las entidades:

— Teniendo en cuenta si la DTD está definida dentro del propio documento XML o en fichero externo, pueden ser entidades internas y entidades externas.

— Teniendo en cuenta el nivel de visibilidad o acceso a ella: entidades generales y entidades de parámetro.

— Teniendo en cuenta si han de ser analizadas por el procesador de XML o no: entidades analizadas y entidades no analizadas.

A) Entidades internas

Pueden estar definidas en cualquier lugar del documento XML o en una DTD interna. Para hacer referencia a una entidad interna en el documento XML, tiene que haber sido definida previamente. Las entidades internas son todas entidades analizadas y, por lo tanto, han de ser de tipo texto.

```
<!ENTITY nombre 'valor´>
```

La referencia a una entidad interna en el documento XML se efectúa por nombre entre las marcas ampersand (&) y punto y coma (;) Ejemplo:&fecha;

B) Entidades externas

Están definidas en archivos externos almacenados en el propio ordenador o en otro distinto de una red. Existen dos métodos de acceso a la entidades externas SYSTEM o PUBLIC, tal como se muestra a continuación:

— SYSTEM identificador_de_sistema.

Tras la palabra SYSTEM se especifica el identificador de sistema, un URI *(Universal Resource Identifier* o Identificador de Recursos Universal) o dirección de la entidad que se desea utilizar.

— PUBLIC identificador_público identificador_de_sistema.

Tras la palabra PUBLIC, se ha de indicar un identificador público, que generalmente hace referencia a una organización o documento estándar, y un identificador de sistema.

C) Entidades generales

Pueden estar definidas en el propio documento XML, en una DTD interna o en una DTD externa. Si hacen referencia a un objeto existente en el propio documento, un objeto XML, suelen ser utilizadas como macros y son expandidas por el procesador XML; reciben el nombre de entidades generales internas.

Las entidades generales que hacen referencia a un objeto o recurso no XML, tales como archivos de imágenes, sonidos, etc., se las denomina entidades generales externas.

```
<!ENTITY nombre 'definicion_de_la_entidad'>
```

La referencia a una entidad de tipo general en el documento XML se efectúa por nombre, entre las marcas ampersand (&) y punto y coma (;) Ejemplo: &tai;.

D) Entidades de párametro

Las entidades de parámetro se utilizan solamente en las DTD. Para su declaración se ha de usar el símbolo de tanto por ciento (%) tras <!ENTITY separado por un espacio. Este tipo de entidades generalmente son utilizadas para definir determinados datos, o listas de ellos, que han de ser utilizados de forma repetida en la DTD para así no tener que escribirlos varias veces.

```
<!ENTITY % parámetro ´definicion_de_la_entidad´>
```

E) Entidades analizadas

El contenido de estas entidades es analizado por el procesador de XML y debe ser siempre de tipo XML. Las entidades analizadas se suelen utilizar para compartir texto entre varios documentos. Pueden ser internas o externas.

F) Entidades no analizadas

Son entidades externas de cualquier tipo: sonidos, imágenes, documentos de texto, etc., y el acceso a ellas se efectúa mediante los métodos SYSTEM o PUBLIC.

8.4.4. Declaración de notación

Estas declaraciones permiten definir un nombre de notación y un identificador externo por medio del cual los procesadores XML puedan localizar la aplicación auxiliar capaz de procesar los datos existentes en un determinado archivo. Las notaciones también permiten establecer la aplicación que debe ejecutar una determinada instrucción de procesamiento.

```
<!NOTATION nombre PUBLIC | SYSTEM [identificador_público] identificador_de_sistema>
```

8.4.5. Declaración de atributos

Estas declaraciones permiten definir el nombre del atributo, tipo de dato que utilizará y valor por defecto, si lo tuviera, para un determinado elemento ya definido en la DTD.

```
<!ATTLIST elemento nombre tipo valor>
```

— **Elemento:** nombre del elemento para el cual se define la lista de atributos.

— **Nombre:** nombre del atributo o referencia a una entidad de parámetro con la lista de atributos ya definida en la DTD.

— **Valor:** se refiere al valor por defecto asignado al atributo; puede ser un dato literal o una palabra clave, predefinida en XML (véase apartado sobre valores por defecto de los atributos).

Tipos de atributos

— CDATA

— TIPOS ENUMERADOS

— NOTATION

• **Valores por defecto de los atributos**

La asignación de un valor por defecto a un atributo se efectúa mediante un literal. En las declaraciones de lista de atributos se pueden utilizar también determinadas palabras clave para indicar si es obligatorio (REQUIRED) asignar un valor al atributo, o bien, la asignación de valor es opcional (IMPLIED).También se puede asignar un valor constante (FIXED) al atributo en su declaración.

Declaración del tipo de documento <!DOCTYPE...>

Cuando un documento XML, va a utilizar una DTD para ser validado, ha de incluir una línea de prólogo en la cual se declara qué tipo de documento es, o sea, qué DTD se debe utilizar para efectuar la validación, por medio de una instrucción <!DOCTYPE...>

La sintaxis genérica de la línea de instrucción DOCTYPE es:

```
<!DOCTYPE elementoRaiz DTD>
```

— **ElementoRaíz (root):** se refiere al nombre del elemento que sirve como raíz de la estructura del árbol del documento.

— **DTD:** mediante este parámetro especificamos la DTD que va ser utilizada por el documento.

8.4.6. Tipos de DTD

Las DTD se clasifican en dos tipos: internas y externas.

— DTD INTERNA

La DTD se encuentra declarada en el propio documento XML.

```
<!DOCTYPE elementoRaíz [ declaración_de_elementos]>.
```

— DTD EXTERNA

Las DTD se pueden guardar en archivos independientes con extensión, dtd; Podemos entonces utilizar dos métodos: SYSTEM o PUBLIC:

- SYSTEM: utilizaremos este método para acceder a DTD's que se encuentren en nuestro propio equipo.
- PUBLIC: utilizamos este método para acceder a DTD's que se encuentran en otros ordenadores de una red.

```
<!DOCTYPE artículo SYSTEM "articulo.dtd">
```

El motor XML incluido en Internet Explorer incorpora un parser para validación de documentos XML basados en DTD.

8.5. XML SCHEMA

También llamado "Esquema", se trata de un documento de definición estructural al estilo de los DTD, que además cumple con el estándar XML. Los documentos Schema (usualmente con extensión XSD) se concibieron como un sustituto de los DTD teniendo en cuenta los puntos débiles de estos y la búsqueda de mayores y mejores capacidades a la hora de definir estructuras para los documentos XML, como la declaración de los tipos de datos.

Un ejemplo de documento Schema vacío podría ser el siguiente:

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

<xsd:schema
xmlns:xsd="http://www.w3.org/2001/XMLSchema" ver-
sion="0.1" xml:lang="es">

</xsd:schema>
```

Ejemplo de XSD:

Partamos del siguiente archivo xml que llamaremos futbol.xml:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<futbol xmlns="futbol.xsd">
      <equipo>
            <nombre> Lolo </nombre>
            <tfo tipo="fijo"> 91 777 77 77 </tfo>
            <mail> lolo@lolo.com </mail>
            <fondos> 66666.33 ? </fondos>
            <socios> 989898 </socios>
            <direccion de="trabajo">
                  <calle plaza="no"> Un poco mas arriba</calle>
                  <provincia> Madrid </provincia>
                  <ciudad> Mostoles </ciudad>
                  <cp> 28777 </cp>
            </direccion>
            <puntos> 25 </puntos>
            <pais division="1">
                  <CEE> España </CEE>
            </pais>
            <codigo> H5E24</codigo>
      </equipo>
      <equipo>
                  <nombre> Popo </nombre>
            <tfo tipo="fijo"> 91 777 77 77 </tfo>
            <mail> popo@popo.com </mail>
            <fondos> 66666.33 ? </fondos>
            <socios> 989898 </socios>
            <direccion de="particular">
                  <calle plaza="no"> Un poco mas abajo</calle>
                  <provincia> Barcelona </provincia>
                  <ciudad> Marsella </ciudad>
                  <cp> 22777 </cp>
            </direccion>
            <puntos> 23 </puntos>
            <pais division="1">
                  <Internacional> España </Internacional>
            </pais>
            <codigo> A5E25</codigo>
      </equipo>
      …
      ….
</futbol>
```

Vamos a crear ahora un esquema que valide el anterior XML. Le llamaremos por supuesto futbol.xsd:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" ver-
sion="1.0" xml:lang="es"?>
      <!—Esquema para validar documentos de futbol—>
      <xsd:element name="futbol" type="datosfutbol"/>

      <xsd:complexType name="datosfutbol">
            <xsd:sequence>
      <xsd:element name="equipo" type="datosequipo" minOc-
curs="1" maxOccurs="unbounded"/>
            </xsd:sequence>
      </xsd:complexType>

      <xsd:complexType name="datosequipo">
            <xsd:sequence>
                  <xsd:element name="nombre" type="xsd:string"/>
                  <xsd:element name="tfo" type="xsd:string"
minOccurs="1" maxOccurs="5"/>
                  <xsd:complexType>
                        <xsd:simpleContent>
                              <xsd:extension base="xsd:string">
                                    <xsd:attribute name="tipo"
type="xsd:string"/>
                              </xsd:extension>
                        </xsd:simpleContent>
                  </xsd:complexType>
                  <xsd:element name="fondos" type="xsd:double"/>
                  <xsd:element name="mail" type="xsd:string"/>
                  <xsd:element name="socios" type="xsd:posi-
tiveInteger"/>
                  <xsd:element name="direccion" type="datosdirec"/>
                  <xsd:element name="puntos" type="xsd:short"/>
                  <xsd:element name="pais" type="datospais"/>
                  <xsd:element name="codigo"
type="datoscodigo"/>
            </xsd:sequence>
      </xsd:complexType>
```

```
      <xsd:complexType name="datosdirec">
            <xsd:sequence>
                  <xsd:element name="calle" type="datoscalle"/>
                  <xsd:element name="provincia"
type="datosprovincia"/>
                  <xsd:element name="ciudad" type="xsd:string"/>
                  <xsd:element name="cp" type="datoscp"/>
                  </xsd:sequence>
            <xsd:attribute name="de" type="xsd:string"
use="required"/>
      </xsd:complexType>

      <xsd:simpleType name="datosprovincia">
            <xsd:restriction base="xsd:string">
                  <xsd:enumeration value="Madrid"/>
                  <xsd:enumeration value="Barcelona"/>
                  <xsd:enumeration value="Bilbao"/>
                  <xsd:enumeration value="Valencia"/>
            </xsd:restriction>
      </xsd:simpleType>

      <xsd:complexType name="datoscalle">
                  <xsd:simpleContent>
                        <xsd:extension base="xsd:string">
                              <xsd:attribute name="plaza"
                              type="xsd:string"/>
                        </xsd:extension>
                  </xsd:simpleContent>
      </xsd:complexType>

      <xsd:simpleType name="datoscp">
            <xsd:restriction base="xsd:short">
                  <xsd:minInclusive value="0"/>
                  <xsd:maxExclusive value="99999"/>
            </xsd:restriction>
      </xsd:simpleType>
```

```
      <xsd:complexType name="datospais">
            <xsd:sequence>
                  <xsd:choice>
                        <xsd:element name="CEE"
type="xsd:string"/>
                        <xsd:element name="Internacional"
type="xsd:string"/>
                  </xsd:choice>
            </xsd:sequence>
            <xsd:attribute name="division" type="xsd:string"
use="required"/>
      </xsd:complexType>

      <xsd:simpleType name="datoscodigo">
            <xsd:restriction base="xsd:string">
                  <xsd:pattern value="[A-Z]{1}-\d{1}-[A-Z]{1}-
\d{2}"/>
            </xsd:restriction>
      </xsd:simpleType>
</xsd:schema>
```

8.6. XSL

XSL *(eXtensible Style Language* o Lenguaje de Estilos Extensible) es un lenguaje basado en XML y especialmente diseñado para la publicación de documentos XML.

Las hojas de estilo XSL se almacenan en archivos independientes, con la extensión (*.xsl) y son enlazadas desde el documento XML que las utiliza de forma similar a como se enlazaban las hojas de estilo CSS.

XSL no es un lenguaje para definir estilos, como en el caso de las hojas de estilo CSS o XSL-FO, sino un lenguaje que permite aplicar determinados formatos ya definidos a documentos XML. Otra de sus características es la posibilidad de efectuar filtrado de información seleccionando los elementos y contenido que deseamos mostrar en la página generada.

Es una familia de lenguajes basados en el estándar XML que permite describir cómo la información contenida en un documento XML cualquiera debe ser transformada o formateada para su presentación en un medio específico. Este lenguaje no se limita a definir qué estilo aplicar a cada elemento del documento XML. Además se pueden realizar pequeñas instrucciones típicas de los lenguajes de programación y la salida no tiene porqué ser un documento HTML, sino que además podría ser de otros tipos, cualquiera que podamos necesitar como un documento escrito en WML (para WAP), un documento de texto plano u otro documento XML.

XSL resulta mucho más potente que CSS y de hecho es mucho más adecuado utilizarlo. Una de sus principales ventajas la vemos a continuación. Si tenemos un documento XML que queremos que se visualice en múltiples dispositivos distintos será imprescindible utilizar XSL. En este esquema tendríamos un solo documento XML y un documento XSL para cada dispositivo que queramos incluir, por ejemplo para un navegador Netscape, otro para Internet Explorer, otro para un móvil de una marca y otro para un móvil de otra marca. Si mañana aparece un nuevo dispositivo, por muy particular que sea, solo necesitaremos crear un documento XSL para que nuestros XML se puedan visualizar en él.

Esta familia está formada por tres lenguajes:

— XSLT (siglas de *Extensible Stylesheet Language Transformations*, lenguaje de hojas extensibles de transformación) (XSL *Transform* o Transformaciones XSL), que permite convertir documentos XML de una sintaxis a otra (por ejemplo, de un XML a otro o a un documento HTML). Es pues una tecnología utilizada para la aplicación de formatos a los documentos XML.

— XSL-FO (lenguaje de hojas extensibles de formateo de objetos) (XSL *Formatting Objects* u Objetos para Formato de XSL), que permite especificar el formato visual con el cual se quiere presentar un documento XML, es usado principalmente para generar documentos PDF. Es pues una tecnología utilizada para definir estilos de forma similar a las hojas de estilo CSS.

— XPath, o XML Path Language, es una sintaxis (no basada en XML) para acceder o referirse a porciones de un documento XML.

Para poder visualizar los resultados de aplicar los formatos y estilos a un documento XML, es necesario que el navegador incorpore un motor XSLT (XSL, Transform), el cual se encarga de efectuar las transformaciones de los estilos definidos y generar la presentación adecuada. Con Internet Explorer 5.5 se incorpora MSXML3.

Mediante XSL y XSL-FO se puede generar la salida de los contenidos de un archivo XML en cualquier formato de publicación: texto plano, HTML, XHTML, WML, PDF (utilizado por Acrobat Reader), RTF, etc.

Las hojas de estilo XSL, se basan en la definición de plantillas (templates). Cada una de las plantillas definidas ha de estar relacionada con algún elemento existente en el documento; sobre dicho elemento se aplicarán las transformaciones y formatos utilizados en la plantilla. La relación plantilla-elemento se establece por medio del parámetro match existente en toda la plantilla y en la cual se asignará la ruta (Xpath) que el elemento XML tiene en la estructura del documento.

El elemento (nodo) raíz del documento, el inicio de la ruta de acceso, es referenciado por el símbolo de barra inclinada (“/).

Para acceder a todos los nodos del documento se puede utilizar “//”.

Para acceder a todos los nodos situados por debajo de uno determinado, se utilizará el carácter comodín *.

- **Tipos de planillas**

En toda hoja de estilos XSL existirá una plantilla raíz, match="/", que estará relacionada con el elemento raíz del documento XML; establece, por lo tanto, cómo ha de efectuarse la transformación genérica que se utilizará en el documento.

- **Creación de plantillas XSL**

La declaración de una plantilla se efectúa por medio de la instrucción XSL <xsl:template>, en la cual se establece la ruta de acceso al elemento relacionado con la plantilla por medio del parámetro match.

- **Instrucciones XSL básicas**

1. Ejecución de las transformaciones definidas en las plantillas.

   Para indicar al motor XSL que aplique todas laas transformaciones definidas en el documento o las utilizadas por defecto, se usa la instrucción <xsl:apply-templates/>.

2. Acceso a contenidos del documento.

   Para indicar, dentro de la plantilla, que se utilice el dato (texto) del elemento XML relacionado con la plantilla, utilizaremos la instrucción <xsl:value-of select=".">.

- **Creación de hojas de estilo XSL**

1º Los documentos XSL comienzan con una línea de prólogo.

```
<?xml version="1.0"?>
```

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

2º La siguiente línea es la línea de declaración de hoja de estilo; sirve para identificar al documento como una hoja de estilos y representa su inicio. Comienza con una instrucción <xsl:stylesheet seguida del parámetro xmlns:xsl (xml NameSpace) por medio del cual estableceremos el nombre de espacio (referencia a la DTD utilizada), en el cual se encuentra definida la versión de XSL que utilizaremos en nuestra hoja de estilos.

```
<xsl:stylesheet xmlns:xsl="URL">
```

3º Los documentos XSL se han de cerrar siempre con la etiqueta </xsl:stylesheef>.

4º Por convención, los archivos de documentos XSL, se guardan con la extensión .xsl.

- **Enlace de hojas de estilo XSL**

El enlace de una hoja de estilos XSL, desde un archivo XML, se efectúa mediante una instrucción de proceso (PI, Processing Intruction). Las instrucciones de proceso están representadas por etiquetas de tipo vacío (sin etiqueta de cierre, solamente utilizan atributos) que comienzan con **<¿** y se cierran con **?>**.

```
<¿xml-stylesheet href="url" type="text/xsl"?>
```

- **Plantillas XSL utilizadas para conversión a texto**

Plantilla básica para mostrar los datos de un documento XML con XSL.

Selección de la información que vamos a mostrar <xsl:value-of select="dato"/>.

Bucles para selección de información <xsl:for-each select="dato">.

8.6.1. XSLT

Al igual que XML, XSLT es un lenguaje de programación. Forma parte de la trilogía transformadora de XML, compuesta por las CSS (*Cascading Style Sheets,* hojas de estilo en cascada), que permite dar una apariencia en el navegador determinada a cada una de las etiquetas XML; XSLT (XML *Stylesheets Language for Transformation*, o lenguaje de transformación basado en hojas de estilo); y XSL:FO (*Formatting Objects*, objetos de formateo), o transformaciones para fotocomposición, o, en general, para cualquier cosa que no sea XML, como por ejemplo HTML "del viejo" o PDF (el formato de Adobe).

XHTML sí es XML, sigue un DTD (varios, en realidad), y solo admite documentos "bien formados". HTML no lo es, aunque pude convertirse fácilmente en XHTML usando utilidades tales como Tidy.

XSLT es pues, un lenguaje que se usa para convertir documentos XML en otros documentos XML; puede convertir un documento XML que obedezca a un DTD a otro que obedezca otro diferente, un documento XML bien formado a otro que siga un DTD o, lo más habitual, convertirlo a formatos finales, tales como WML (usado en los móviles WAP) o XHTML.

Los programas XSLT están escritos en XML, y generalmente, se necesita un procesador de hojas de estilo, o stylesheet processor para procesarlas, aplicándolas a un fichero XML.

El estilo de programación con las hojas XSLT es totalmente diferente a los otros lenguajes a los que estamos acostumbrados (tales como C++ o Perl), pareciéndose más a lenguajes tales como el AWK, o a otros lenguajes funcionales, tales como ML o Scheme.

En la práctica, eso significa dos cosas:

— **No hay efectos secundarios**. Una instrucción debe de hacer lo mismo cualquier que sea el camino de ejecución que llegue hasta ella.

— **La programación está basada en reglas**: cuando ocurre algo en la entrada, se hace algo en la salida.

Lo que consiguen las hojas de estilo es separar la información (almacenada en un documento XML) de su presentación, usando en cada caso las transformaciones que sean necesarias para que el contenido aparezca de la forma más adecuada en el cliente. Es más, se pueden usar diferentes hojas de estilo, o incluso la misma, para presentar la información de diferentes maneras dependiendo de los deseos o de las condiciones del usuario.

XSLT o XSL Transformaciones es un estándar de la organización W3C que presenta una forma de transformar documentos XML en otros e incluso a formatos que no son XML. Las hojas de estilo **XSLT** realizan la transformación del documento utilizando una o varias reglas de plantilla: unidas al documento fuente a transformar, esas reglas de plantilla alimentan a un procesador de **XSLT**, el cual realiza las transformaciones deseadas colocando el resultado en un archivo de salida o, como en el caso de una pagina web, directamente en un dispositivo de presentación, como el monitor de un usuario.

Actualmente, **XSLT** es muy usado en la edición Web, generando páginas HTML o XHTML. La unión de XML y **XSLT** permite separar contenido y presentación, aumentando así la productividad. XSLT o XSL Transformations es la parte más importante del lenguaje XSL. La función de XSLT es la de transformar documentos XML en documentos XHTML u otros documentos XML. El W3C es el encargado de la definición de especificación XSLT.

XSLT se basa en XPath para realizar la búsqueda de información a través del documento XML. XPath son cadenas que son expresiones regulares, las cuales hacen referencia a alguna estructura dentro del documento XML.

El proceso de transformación se basa en plantillas. Dichas plantillas identifican una estructura a partir de la cual realizar la transformación (con XPath), así como las acciones a realizar con dicha estructura: recorrerla, obtener el dato de la etiqueta, el valor de alguno de sus atributos, contar cuantos elementos tiene la etiqueta anidados,...

Además, para poder aplicar las transformaciones, necesitaremos asociar el documento de transformación al documento XML receptor de la misma. A diferencia del lenguaje HTML, donde cada una de sus etiquetas lleva asociada una representación gráfica, el XML identifica datos, los cuales no tienen representación gráfica asociada.

Cuando definimos una tabla en HTML (la etiqueta <table>), sabemos que las herramientas que interpreten el documento HTML, normalmente los navegadores web, pintarán la tabla. De una forma u otra visualizaremos la tabla en nuestra pantalla.

Si tenemos un documento XML, donde podemos tener definida la etiqueta <libro>, esta no tendrá ninguna representación gráfica asociada. Es por ello que si visualizamos nuestro documento XML con alguna herramienta,

esta, mostrará el contenido de la etiqueta, pero sin ninguna representación. Es en este punto donde entra el lenguaje XSLT. Y es que este lenguaje permite transformar el susodicho documento XML en otro formato, el resultado de la transformación será el que lleve la representación gráfica.

8.6.2. XSL-FO (Objetos de Formateo)

Mediante los objetos de formateo (Formatting Objects –FO–) y sus propiedades podemos describir cómo se van a visualizar los componentes de un documento. Con estos objetos definimos:

— Las características de la página.

— Los párrafos.

— Las listas.

— Las tablas.

— Los enlaces.

— etc.

La especificación **XSL indica el vocabulario XML que define estos objetos de formateo.**

El siguiente código es un pequeño ejemplo de fichero XSL-FO:

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

<!—         Hola.fo     —>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <fo:layout-master-set>
      <fo:simple-page-master master-name="simple"
             page-height="29.7cm"          page-width="21cm"
             margin-top="1cm"             margin-bottom="2cm"
             margin-left="2.5cm"          margin-
right="2.5cm">
       <fo:region-body margin-top="3cm"/>
       <fo:region-before extent="3cm"/>
       <fo:region-after extent="1.5cm"/>
    </fo:simple-page-master>
  </fo:layout-master-set>

  <fo:page-sequence master-name="simple">
    <fo:flow flow-name="xsl-region-body">
```

```
      <fo:block font-size="18pt"
            font-family="sans-serif"
            line-height="24pt"
            space-after.optimum="15pt"
            text-align="center"
            padding-top="3pt">
        Mi primer XSL-FO
      </fo:block>

      <fo:block font-size="12pt"
                font-family="sans-serif"
                line-height="15pt"
                space-after.optimum="3pt"
                text-align="justify">
                Hola este es mi primer XSL-FO.
      </fo:block>
    </fo:flow>
  </fo:page-sequence>
</fo:root>
```

En el que podemos observar:

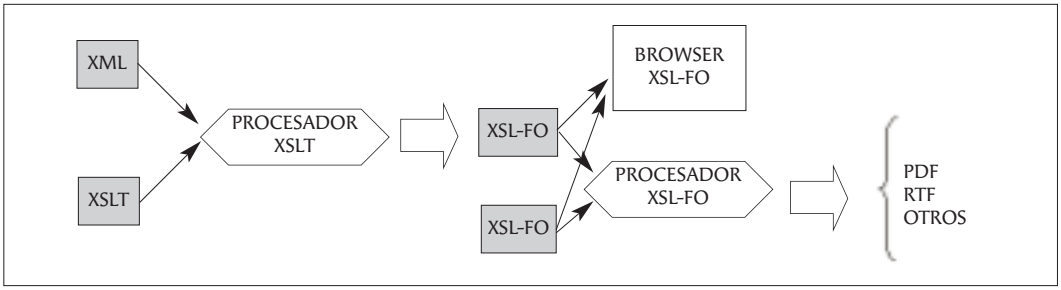
— Que se trata de un vocabulario XML, en el que todos los elementos van precedidos del namespace 'fo', y que por tanto al escribir el elemento raiz del documento XML debemos declararlo de la siguiente manera:

<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">

— Que esta formado por un conjunto de elementos: fo:simple-page-master, fo:flow, fo:block, etc. mediante los cuales y sus propiedades (definidas en los atributos): font-size, font-family, etc. describimos cómo se visualizan de forma genérica los componentes de un documento. **La especificación define por tanto todos estos elementos y sus propiedades y cómo deben expresarse mediante un vocabulario XML.**

- **Procesadores XSL-FO**

Un procesador XSL es la aplicación que procesa un documento XML compuesto de XSL-FOS y lo presenta de manera que una persona lo pueda leer facilmente.



Entre los procesadores de XSL-FO más significativos se pueden destacar los siguientes:

— XEP, desarrollado por RenderX.

— PassiveTex. Es una librería de macros en Tex que pueden ser usadas para procesar documentos XML formados por XSL-FO.

— XSL Formatter, de Antenna House Inc.

— Unicorn Formatting Objects (UFO), es un procesador de XSL-FO implementado en C++.

— FOP, que es un procesador de XSL-FO desarrollado en JAVA por Apache XML Project.

8.7. XHTML

XHTML, acrónimo inglés de *e**X**tensible **H**yper **T**ext **M**arkup **L**anguage* (lenguaje extensible de marcado de hipertexto), es el lenguaje de marcado pensado para sustituir a HTML como estándar para las páginas web. XHTML es la versión XML de HTML, por lo que tiene, básicamente, las mismas funcionalidades, pero cumple las especificaciones, más estrictas, de XML. Su objetivo es avanzar en el proyecto del World Wide Web Consortium de lograr una web semántica, donde la información, y la forma de presentarla estén claramente separadas. En este sentido, XHTML serviría únicamente para transmitir la información que contiene un documento, dejando para hojas de estilo (como las hojas de estilo en cascada) y JavaScript su aspecto y diseño en distintos medios (ordenadores, PDA's, teléfonos móviles, impresoras...).

8.7.1. Ventajas

Las principales ventajas del XHTML sobre otros formatos son:

— Compatibilidad parcial con navegadores antiguos: la información se visualiza, aunque sin formato. Apuntar que el XHTML 1.0 fue diseñado expresamente para ser mostrado en navegadores que soportan HTML de base.

— Un mismo documento puede adoptar diseños radicalmente distintos en diferentes aparatos, pudiendo incluso escogerse entre varios diseños para un mismo medio.

— Facilidad de edición directa del código y de mantenimiento.

— Formato abierto, compatible con los nuevos estándares que actualmente está desarrollando el W3C como recomendación para futuros agentes de usuario o navegadores.

— Los documentos escritos conforme a XHTML 1.0 pueden potencialmente presentar mejor rendimiento en las actuales herramientas web que aquellos escritos conforme a HTML.

8.7.2. Inconvenientes

— Algunos navegadores antiguos no son totalmente compatibles con los estándares, lo que hace que las páginas no siempre se muestren correctamente. Esto cada vez es menos problemático, al ir cayendo en desuso.

— Muchas herramientas de diseño web aún no producen código XHTML correcto.

8.7.3. Diferencias entre HTML y XHTML

Se eliminan elementos no semánticos:

— Desaparecen las etiquetas de HTML <font>, .

— Desaparecen varios atributos de formato.

— El aspecto del documento se describe únicamente a través de hojas de estilo.

Al ser XML, se exige:

— Incluir siempre la etiqueta doctype apropiada.

— Todas las etiquetas deben cerrarse, aunque sea poniendo una barra /, como, por ejemplo:
 pasa a ser
.

— Solo pueden incluirse datos en formatos admitidos por XML. Esto da problemas para incluir JavaScript en los documentos directamente.

— Todos los atributos deben tener un valor y meterlo entre comillas.

Además, los nombres y atributos de todas las etiquetas deben estar en minúsculas.

8.8. XQUERY

"libros.xml"

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<bib>
  <libro año="2008">
    <titulo>Programación XML</titulo>
    <autor><apellido>Lopez</apellido><nombre>L.</nombre></autor
  >
    <editorial>ADAMS</editorial><precio> 65.95</precio>
  </libro>
  <libro año="2007">
    <titulo> Programación para Linux</titulo>
    <autor><apellido>Lopez</apellido><nombre>L.</nombre></autor
  >
    <editorial>ADAMS</editorial><precio>65.95</precio>
  </libro>
  <libro año="2000">
    <titulo>Data on the Web</titulo>
    <autor><apellido>Lopez</apellido><nombre>Serge</nombre></au
  tor>
    <autor><apellido>Rodríguez</apellido><nombre>Peter</nom-
  bre></autor>
    <autor><apellido>Alvarez</apellido><nombre>Dan</nombre>
    </autor><editorial>Morgan Kauf
  </editorial><precio>39.95</precio>
  </libro>
  <libro año="1999">
    <titulo> Economics Digital TV</titulo>
    <editor><apellido>Suarez</apellido><nombre>Darcy</nombre>
    <afiliacion>CITI</afiliacion>
    </editor>
    <editorial>Kluwer Acad </editorial><precio>129.95</precio>
  </libro>
</bib>
```

A continuación se muestra el contenido del DTD correspondiente al archivo *"libros.xml".*

```
<!ELEMENT bib (libro* )>
<!ELEMENT libro (titulo,(autor+ | editor+ ),editorial,
precio )>
<!ATTLIST libro year CDATA #REQUIRED >
```

```
<!ELEMENT autor (apellido, nombre )>
<!ELEMENT editor (apellido, nombre, afiliacion )>
<!ELEMENT titulo (#PCDATA )>
<!ELEMENT apellido (#PCDATA )>
<!ELEMENT nombre (#PCDATA )>
<!ELEMENT afiliacion (#PCDATA )>
<!ELEMENT editorial (#PCDATA )>
<!ELEMENT precio (#PCDATA )>
```

La siguiente consulta devuelve los títulos de los libros que tengan más de dos autores ordenados por su título.

```
for $b in doc("libros.xml")//libro
let $c := $b//autor
where count($c) > 2
order by $b/titulo
return $b/ titulo
```

El resultado de esta consulta se muestra a continuación.
<title>Data on the Web</title>

La siguiente consulta devuelve los títulos de los libros del año 2000.

```
for $b in doc("libros.xml")//libro
where $b/@año = "2000"
return $b/titulo
```

```
doc("libros.xml")/bib/libro/titulo[/bib/libro/autor/apellido='Lopez']
```

Esta expresión XPath, que también es una consulta XQuery válida, devuelve los títulos de los libros que tengan algún autor de apellido 'Lopez'.

La consulta con una cláusula "for" se muestra a continuación.

```
for $d in doc("libros.xml")/bib/libro/titulo
return
<titulos>{ $d }</titulos>
```

El resultado de esta consulta se muestra a continuación:

```
<titulos><titulo>Programación XML</titulo></titulos>
<titulos><titulo> Programación para Linux</titulo></titulos>
<titulos><titulo>Data on the Web</titulo></titulos>
<titulos><titulo> Economia mundial</titulo></titulos>
```

A continuación repetimos la misma consulta sustituyendo la cláusula for por una cláusula let.

```
let $d := doc("libros.xml")/bib/libro/titulo
return
<titulos>{ $d }</titulos>
```

El resultado de esta consulta se muestra a continuación.

```
<titulos>
<titulo>Programación XML</titulo>
<titulo> Programación para Linux</titulo>
<titulo>Data on the Web</titulo>
<titulo> Economia mundial</titulo>
</titulos>
```

Esta consulta devuelve el título de cada uno de los libros de archivo "libros.xml" junto con el número de autores de cada libro.

```
for $b in doc("libros.xml")//libro
let $c := $b/autor
return
<libro>{ $b/titulo, <autores>{ count($c) }</autores>}</libro>
```

En esta se devuelve los titulos de todos los libros contenidos en el archivo "libros.xml" y todos los comentarios de cada libro contenidos en "comentarios.xml":

```
for $t in doc("libros.xml")//titulo,
$e in doc("comentarios.xml")//entrada
where $t = $e/titulo
return <comentario>{ $t, $e/comentario }</comentario>
```

La siguiente consulta devuelve los títulos de todos los libros almacenados en el archivo “libros.xml” y sus dos primeros autores. En el caso de que existan más de dos autores para un libro, se añade un tercer autor “et al.”.

```
for $b in doc(“libros.xml”)//libro
return
<libro>
{ $b/titulo }
{
for $a at $i in $b/autor
where $i <= 2
return <autor>{string($a/last), “, “,
string($a/first)}</autor>
}
{
if (count($b/autor) > 2)
then <autor>et al.</autor>
else ()
}
</libro>
```

Esta devuelve los titulos de los libros en los que al menos uno de sus autores es L. Lopez:

```
for $b in doc(“libros.xml”)//libro
where some $a in $b/autor
satisfies ($a/last=”Lopez” and $a/first=”L.”)
return $b/titulo
```

La siguiente consulta devuelve todos los títulos de los libros en los que todos los autores de cada libro es L. Lopez.

```
for $b in doc(“libros.xml”)//libro
where every $a in $b/autor
satisfies ($a/last=”Lopez” and $a/first=”L.”)
return $b/titulo
```

La siguiente consulta devuelve los títulos de los libros que mencionen “Unix” y “programacion” en el mismo párrafo. Si el libro tiene más de un párrafo solo es necesario que aparezca en, al menos, uno de ellos.

```
for $b in doc("bib.xml")//libro
where some $p in $b//parrafo satisfies
(contains($p,"Unix") AND contains($p,"programacion"))
return $b/title
```

La siguiente consulta devuelve el título de todos los libros que mencionen “programacion” en cada uno de los párrafos de los libros almacenados en “bib.xml”.

```
for $b in doc("bib.xml")//libro
where every $p in $b// parrafo satisfies
contains($p,"programacion")
return $b/title
```

Esta consulta es distinta de la anterior ya que no es suficiente que “programacion” aparezca en al menos uno de los párrafos, sino que debe aparecer en todos los párrafos que existan.

Una consulta que usa el operador unión para obtener una lista ordenada de apellidos de todos los autores y editores:

```
for $l in distinct-values(doc("libros.xml")
//(autor | editor)/apellido)
order by $l
return <apellidos>{ $l }</apellidos>
```

Consulta que usa el operador sustracción para obtener un nodo libro con todos sus nodos hijos salvo el nodo <precio>.

```
for $b in doc("libros.xml")//libro
where $b/titulo = "Programación XML"
return
<libro>
{ $b/@* }
{ $b/* except $b/precio }
</libro>
```

La siguiente consulta devuelve todos los nodos libro que tengan al menos un nodo autor.

```
for $b in doc("libros.xml")//libro
where not(empty($b/autor))
return $b
```

Como la consulta anterior tiene una cláusula "where" que comprueba una negación sobre "empty()", podemos rescribirla usando la función "exists()". El resultado de esta consulta es el mismo que el resultado de la consulta anterior.

```
for $b in doc("libros.xml")//libro
where exists($b/autor)
return $b
```

---

9. Navegadores y lenguajes de programación web. Lenguajes de script

La representación de información mediante hipertexto es la base principal de la tecnología web, la conexión de páginas a través de enlaces proporciona un mecanismo flexible y adaptable a la creación de hiperespacios de información.

Una página web vista en un navegador, o cliente web, está compuesta por diferentes elementos como textos, imágenes, vídeos y diferente información procedente de bases de datos.

Inicialmente la web solo empleaba páginas estáticas basadas en HTML, donde la presentación era fija, pero posteriormente la evolución tecnológica aportó páginas dinámicas que se conformaban en función de la interacción con el operador.

Distinguimos para estas últimas aquellas páginas dinámicas que construyen la presentación en el cliente y las que construyen la respuesta a entregar en el servidor.

En las páginas dinámicas que se procesan en el cliente toda la carga de procesamiento la soporta el navegador mientras que en las páginas dinámicas que se ejecutan en el servidor se posibilita el acceso a muchos recursos externos al ordenador del cliente, principalmente bases de datos alojadas en servidores.

Para estas dos modalidades podemos clasificar los lenguajes de diseño web en:

— **Lenguajes web de cliente.**

— **Lenguajes web de servidor.**

Hacemos a continuación una revisión de los **lenguajes web de cliente**:

— **Javascript:** es un lenguaje interpretado sin compilación. Fue creado por Netscape Communications y es similar a JAVA, aunque no es un lenguaje orientado a objetos. Para evitar incompatibilidades entre navegadores el World Wide Web Consortium (W3C) diseñó un estándar denominado DOM (en inglés *Document Object Model*). Su principal desventaja es que el código es visible por el usuario.

— **Visual Basic Script:** es un lenguaje de programación de scripts del lado del cliente, pero solo compatible con Internet Explorer. Su utilización está desaconsejada en favor de Javascript. Está basado en Visual Basic, sin embargo, no todo lo que se puede hacer en Visual Basic se puede hacer en Visual Basic Script, ya que se trata de una versión reducida del primero.

— **DHTML:** no es en sentido estricto un lenguaje de programación, se trata de un conjunto de capacidades de las que disponen los navegadores modernos, las páginas que responden a las interacciones del usuario se pueden englobar dentro de DHTML, y en ellas se pueden mostrar y ocultar elementos, modificando su posición, dimensiones, color, etc.

— **CSS:** las hojas de estilo en cascada son un recurso para definir los estilos de un sitio web. Esto se consigue creando un archivo donde se definen las declaraciones de estilo y se enlazan todas las páginas del sitio con este archivo. De este modo, todas las páginas comparten una misma declaración de estilo.

— **Applets de JAVA:** se trata de pequeños programas hechos en JAVA, que se transfieren a los clientes web. Los applets están programados en JAVA y precompilados, son mucho menos dependientes del navegador que Javascript y más potentes. En relación con Javascript los applets son más lentos de procesar y tienen un espacio muy delimitado en la página donde se ejecutan. Los applets de JAVA no abren ventanas secundarias, ni controlan frames, formularios, capas, etc.

A continuación hacemos una revisión de los **lenguajes web de servidor**:

— **CGI, Common Gateway Interface:** actualmente se encuentra un poco desfasado por la dificultad con la que se desarrollan los programas y la pesada carga que supone para el servidor que los ejecuta. Los CGI se escriben habitualmente en Perl, sin embargo, otros lenguajes como C, C++ o Visual Basic pueden ser también empleados para construirlos. Cada programa CGI que se pone en marcha lo hace en un espacio de memoria propio. Así, si tres usuarios ponen en marcha un CGI a la vez se multiplicará por tres la cantidad de recursos que ocupe ese CGI. Esto supone una grave ineficiencia.

— **ASP, Active Server Pages:** es una tecnología del lado de servidor desarrollada por Microsoft para sitios web dinámicos y requiere tener instalado Internet Information Server (IIS). ASP no necesita ser compilado para ejecutarse. Existen varios lenguajes que se pueden utilizar para crear páginas ASP. El más utilizado es VBScript, nativo de Microsoft. Los

archivos tienen la extensión (asp). Actualmente está disponible la siguiente versión de ASP, ASP.NET, que comprende mejoras en cuanto a las posibilidades del lenguaje y rapidez de funcionamiento. Para el desarrollo de ASP.NET se puede utilizar C#, VB.NET o J#. Los archivos cuentan con la extensión (aspx). Para su funcionamiento se necesita tener instalado IIS con el Framework .Net.

— **PHP, Hypertext Preprocesor:** PHP es un acrónimo recursivo que significa "PHP Hypertext Pre-processor", (inicialmente se llamó Personal Home Page). Surgió en 1995, desarrollado por PHP Group. PHP es un lenguaje de script interpretado en el lado del servidor utilizado para la generación de páginas web dinámicas, embebidas en código HTML y ejecutadas en el servidor. PHP no necesita ser compilado para ejecutarse. Para su funcionamiento necesita tener instalado Apache o IIS con las librerías de PHP. La mayor parte de su sintaxis ha sido tomada de C, JAVA y Perl con algunas características específicas. Los archivos cuentan con la extensión (php). Se caracteriza por ser un lenguaje muy rápido. Soporta en cierta medida la orientación a objetos (clases y herencia). Es un lenguaje multiplataforma: Linux, Windows, entre otros. Tiene capacidad de conexión con la mayoría de bases de datos: MysSQL, PostgreSQL, Oracle, MS SQL Server, entre otras. Es libre, por lo que se presenta como una alternativa de fácil acceso para todos. Incluye gran cantidad de funciones. No requiere definición de tipos de variables ni manejo detallado del bajo nivel.

— **JSP, JAVA Server Pages:** es un lenguaje para la creación de sitios web dinámicos. Está orientado a desarrollar páginas web en JAVA. JSP es un lenguaje multiplataforma, creado para ejecutarse del lado del servidor. JSP fue desarrollado por Sun Microsystems. Posee un motor de páginas basado en los servlets de JAVA. Para su funcionamiento se necesita tener instalado un servidor Tomcat. Los archivos se encuentran con la extensión (jsp).

— **Lenguaje Perl:** Perl es uno de los lenguajes más antiguos siendo muy flexible, los scripts Perl se asemejan bastante a PHP. La principal causa de la "sucia" apariencia de Perl es la afición de sus desarrolladores a escribir numerosas funcionalidades en una sola línea de código. La potencia de Perl a la hora de procesar grandes cantidades de datos lo hace realmente popular a la hora de desarrollar aplicaciones del lado del servidor.

— **Lenguaje Python:** es un lenguaje de programación creado en el año 1990 por Guido van Rossum. Python es comparado habitualmente con Perl. Los usuarios lo consideran un lenguaje más "limpio" para programar. Permite la creación de todo tipo de programas incluyendo sitios web. Su código no necesita ser compilado.

— **Lenguaje Ruby:** es un lenguaje interpretado de alto nivel y orientado a objetos. Desarrollado en el 1993 por el programador japonés Yukihiro Matsumoto. Su sintaxis está inspirada en Phyton y Perl. Es distribuido bajo licencia de software libre (Opensource). Ruby es un lenguaje dinámico para programación orientada a objetos rápida y sencilla.