Estudio y construcción de herramientas de validación HTML

Eduardo Fernández Casal
2º ciclo de Ingeniería en Informática

Jordi Ferrer Duran

06/2012

A mi padre

FICHA DEL TRABAJO FINAL

Título del trabajo:	Estudio y construcción de herramientas de validación HTML
Nombre del autor:	Eduardo Fernández Casal
Nombre del consultor:	Jordi Ferrer Duran
Fecha de entrega (mm/aaaa):	06/12
Área del Trabajo Final:	Compiladores
Titulación:	2º ciclo de Ingeniería Informática

Resumen del Trabajo (máximo 250 palabras):

El trabajo consta de dos tareas claramente diferenciadas, pero englobadas dentro del área de la validación de código HTML. En la primera de estas dos tareas se trata el estudio del proceso de validación de código HTML. Dentro de esta tarea nos centraremos en la evolución y las diferentes versiones del lenguaje HTML y analizaremos los errores más comunes en las páginas actuales. También estudiaremos herramientas de validación ya desarrolladas. Veremos qué funcionalidades ofrecen y compararemos sus comportamientos con documentos HTML incorrectos. Finalmente, analizaremos el proceso de *parsing* que tiene lugar en los diferentes navegadores actuales, desglosaremos las tareas de las que se compone y contrastaremos los resultados generados por cada navegador.

La segunda tarea consiste en el desarrollo de una herramienta de validación de código XHTML5 propia. La herramienta recibirá como entrada código XHTML5 con posibles errores y devolverá como productos de salida código XHTML5 válido, además de un archivo XML con información referente a los errores encontrados y reparados. A no ser de que se trate de errores globales que afecten de manera global al documento, se indicará la línea. También se incluirá la decisión tomada para repararlo.

Abstract (in English, 250 words or less):

The Final Year Project consists of two essentially different parts, which share a common theme: HTML code validation. The first of these two parts focuses on the study of the validation process. It supplies a brief introduction to the evolution of HTML and XHTML, the new tags introduced in HTML5 and the most common errors found in today's websites. Already developed HTML validation tools are analyzed and examined in detail in order to compare their features and evaluate their performances. Lastly, a comparison of the parsing

process in the most common browsers found nowadays is provided.

In the second part of the project the focus of the project is shifted towards the development of a XHTML5 validation tool. The input is a XHTML5 file whose content may or may not comply with the W3C specification, and therefore, may or may not be a valid XHTML5 document. The output provided by this tool will be a fixed XHTML5 document and an error log returned in the form of a XML file. Information as to the course of action pursued to fix the error and its location will also be included.

Palabras clave(entre 4 y 8):

xhtml, xhtml5, html5, html, validator, parser, scanner

Índice

Lista de figuras

1. Introducción

1.1 Contexto y justificación del Trabajo

Del Plan Docente podemos extraer la siguiente frase, que define el propósito global del Proyecto de Fin de Carrera:

"El Proyecto de Fin de Carrera (PFC) es una asignatura que está pensada para realizar un trabajo de síntesis de los conocimientos adquiridos en otras asignaturas de la carrera y que requiere ponerlos en práctica de manera conjunta en un trabajo concreto."

1.2 Objetivos del Trabajo

Este proyecto tiene dos objetivos principales. El primer objetivo es analizar la problemática de los errores en el código HTML. Concretamente, se pretende hacer el estudio siguiente:

- Identificar los problemas más habituales en el código HTML que podemos encontrar en las páginas actuales.
- Estudiar las herramientas, paquetes y librerías más frecuentes para validar y corregir el código HTML.
- Analizar las técnicas que se usan para tratar el HTML inválido: cómo se recuperan de los errores, qué tipos de errores se admiten, ...
- Hacer un estudio de las técnicas de *parsing* utilizadas en los navegadores web actuales (podéis hacer algún pequeño experimento con HTML incorrecto para ver el comportamiento de diferentes navegadores).

Una vez hecho este estudio, el segundo objetivo es desarrollar una herramienta de corrección de código XHTML5 utilizando las herramientas de compiladores estudiadas a lo largo de la titulación. La herramienta de corrección debería recibir como entrada código XHTML5 con posibles errores y generar como salida:

- Un código XHTML5 "reparado", donde los errores se hayan eliminado y donde posiblemente se habrá perdido alguna información del documento original.
- Un documento con la información de los errores detectados y las decisiones realizadas para repararlo en formato XML (la estructura del XML se podrá definir libremente)"

1.3 Enfoque y método seguido

La primera sección del proyecto es una parte de documentación que resolveremos con la ayuda de herramientas de búsqueda en línea.

Para llevar a cabo la segunda parte del proyecto se ha decidido hacer uso de J2EE. La principal razón tras esta elección es cumplir con uno de los objetivos especificados en la propuesta del Proyecto de Fin de Carrera y en el Plan Docente, que es la aplicación de los conocimientos adquiridos a lo largo de la titulación.

Atendiendo a los actores del sistemas, podremos encontrar una estructura con dos elementos diferenciados:

- Aplicación web: actúa como interfaz entre el usuario y el validador en sí. Ofrece al usuario un formulario mediante el cual puede comunicarse con la herramienta. Recibe el código HTML introducido y lo suministra como entrada al validador. Tras la ejecución de este, recoge los productos de salida y los muestra en pantalla.

- Validador: piedra angular del Proyecto de Fin de Carrera y verdadero propósito de la segunda tarea de este. Su desarrollo incluye toda la funcionalidad clave del trabajo, mientras que la interfaz web solo añade funcionalidad extra.

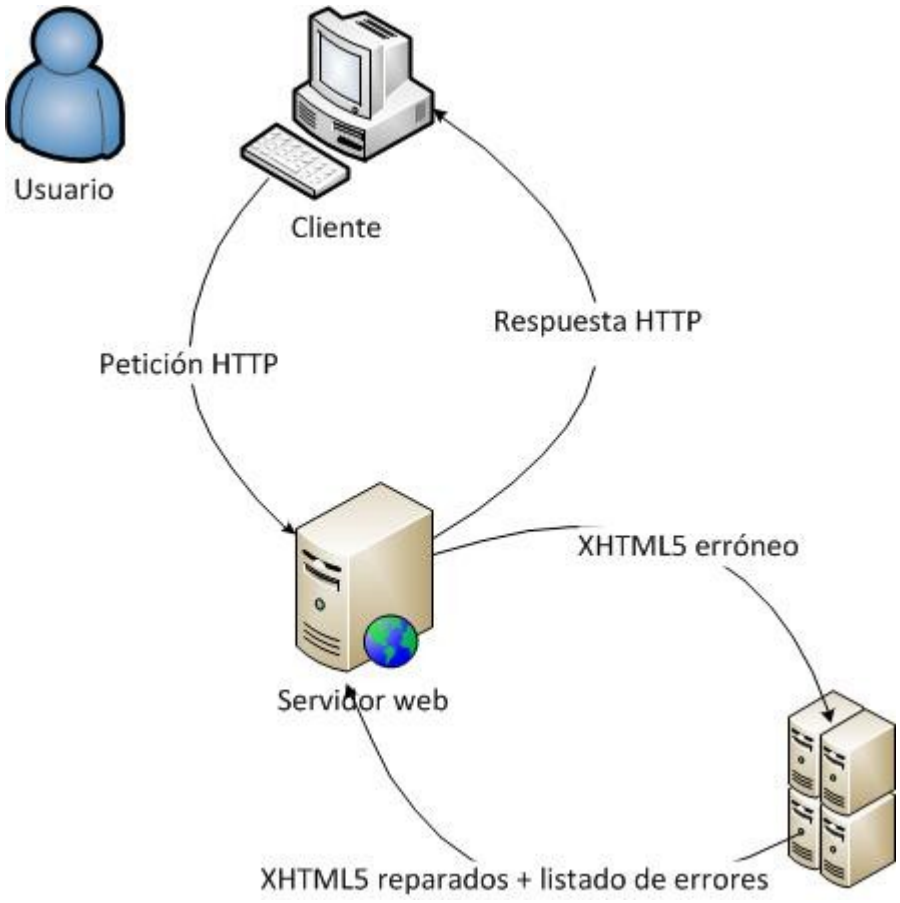


Ilustración 1: Diagrama de actores

Como podemos prever, el validador presenta la estructura más compleja de todo el trabajo. Conviene por tanto ahondar en ella y describirla con más detalle.

- Análisis léxico. El texto de entrada recibido desde la interfaz web es leído y dividido en las unidades mínimas con las que trabajaremos en el resto de módulos: los *tokens*. La idea es siempre ser lo más flexibles posible, ya que estamos suponiendo que el código HTML no será válido en la mayoría de las ocasiones. Para acometer esta fase del desarrollo emplearemos un archivo JLEX en el que definiremos las reglas léxicas correspondientes.

- Analizador sintáctico. El *scanner* implementado en el archivo JLex se comunica directamente con el *parser* implementado en un archivo CUP.

- Analizador semántico. El *parser* no se encargará unicamente de las labores de análisis sintáctico, sino que verá aumentada su funcionalidad con labores de análisis semántico.

- Generador de errores. A medida que los tres tipos de análisis son llevados a cabo sobre el texto de entrada, los diferentes errores son almacenados en una estructura interna. El propósito del generador de errores es estructurar estos errores en un fichero XML con un estructura determinada.

- Corrector de código. Además de llevar cuenta de todos los errores encontrados, los tendremos en consideración para modificar el código XHTML5 de entrada con el fin de subsanarlos.

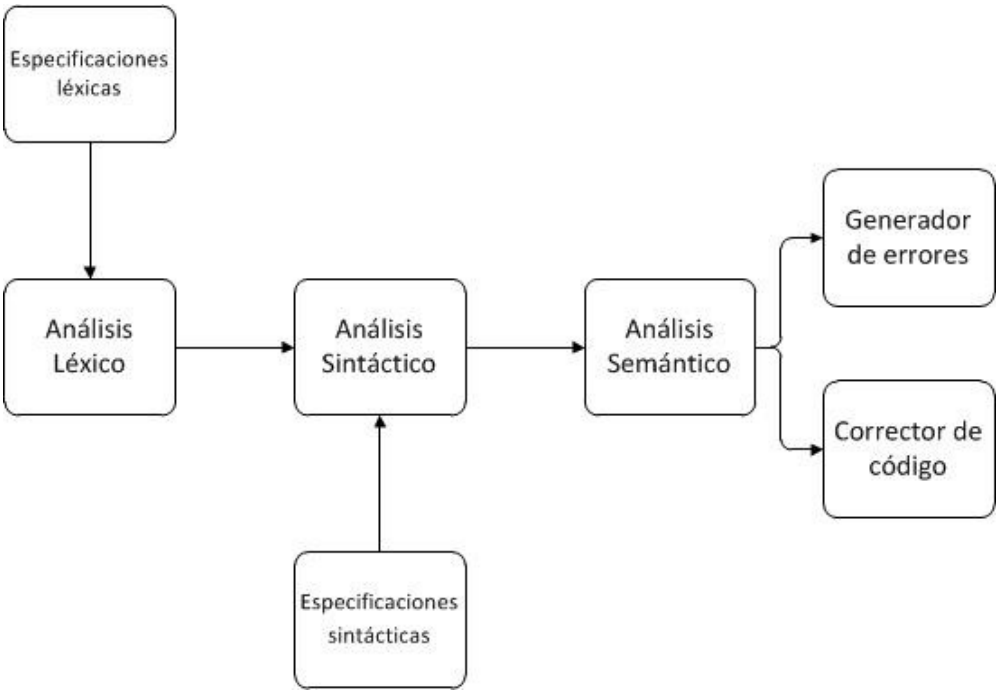


Ilustración 2: Esquema de la construcción del validador

1.4 Planificación del proyecto

Para llevar a cabo este proyecto solamente será necesario un punto de trabajo estándar. No obstante, la conexión a Internet será indispensable para llevar a cabo la documentación y la recopilación de información correspondientes a la primera fase del proyecto.

1.4.1 Instalación y configuración del entorno

Instalación y configuración del servidor de aplicaciones, el IDE o entorno de desarrollo.

1.4.2 Estudio de herramientas

Esta tarea se corresponde aproximadamente con el entregable de la segunda prueba de evaluación continua.

1.4.2.1. Detección de errores más habituales

Identificar los problemas más habituales en el código HTML que se pueden encontrar en las páginas actuales.

1.4.2.2. Recopilación de herramientas más frecuentes

Estudiar las herramientas, paquetes y librerías más frecuentes para validar y corregir el código HTML.

1.4.2.3. Recuperación de errores

Analizar las técnicas que se emplean para tratar el HTML inválido: como se recuperan de los errores, qué tipos de errores se admiten...

1.4.2.4. *Parsing* de navegadores

Hacer un estudio de la de las técnicas de *parsing* utilizadas en los navegadores actuales.

1.4.3 Desarrollo de la nueva herramienta

Esta tarea se corresponde aproximadamente con el entregable de la tercera práctica de evaluación continua,

1.4.3.1. Validador

Esta tarea engloba todas las subtareas necesarias para desarrollar la herramienta capaz de recibir como entrada código XHTML5 con posibles errores y devolver un listado de errores en formato XML y una versión reparada del documento de entrada.

1.4.3.1.1. Estudio XHTML5

Estudio y análisis de la especificación de HTML5 y de las características y estructura de los archivos XML.

1.4.3.1.2. Analizador léxico

Desarrollo del módulo del validador encargado de dividir el código de entrada en las unidades mínimas de trabajo del resto de la herramienta o *tokens*.

1.4.3.1.3. Analizador sintáctico

Desarrollo del módulo encargado de comprobar la organización de los *tokens* en el documento de manera que la estructura del documento sea correcta.

1.4.3.1.4. Analizador semántico

Desarrollo del módulo encargado de comprobar no solo que la estructura del documento sea correcta, si no que sea válido de acuerdo a las restricciones definidas en la especificación de HTML5.

1.4.3.1.5. Generación HTML válido

Se toma como partida el código de entrada y se subsanan los errores encontradas con el fin de obtener código XHTML5 válido.

1.4.3.1.6. Generación XML

Listado de los errores de validación con formato XML indicando ubicación y acción tomada para subsanarla.

1.4.3.1.7. Pruebas

Las pruebas se realizan a lo largo de cada una de las fases anteriores, pero se ha reservado tiempo para su validación de manera global. En esta tarea se incluye el tiempo necesario para solucionar los posibles errores encontrados.

1.4.3.2. Aplicación web

Esta tarea engloba todas las subtareas orientadas a la elaboración de la interfaz web de la herramienta.

1.4.3.2.1. Diseño y maquetación

Esbozo y maquetación CSS de las pantallas de la interfaz web.

1.4.3.2.2. Programación

Desarrollo de la interfaz web, la llamada al validador e integración de la maquetación.

1.4.3.2.3. Pruebas

Pruebas sobre el validador conectado a la interfaz web, esto es, la herramienta completa.

1.4.3.3. Documentación

1.4.3.3.1. Memoria

Elaboración de este documento. Se realiza de manera gradual, al mismo paso que el resto de tareas de desarrollo.

1.4.3.3.2. Presentación

Es la última tarea del proyecto. Se realiza después de todas las demás.

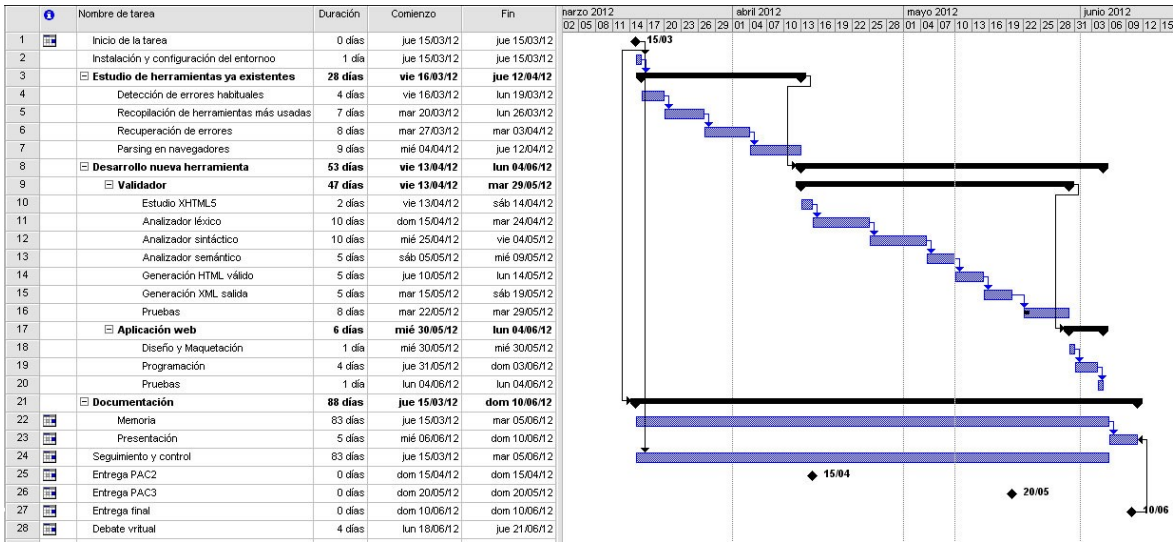
1.4.3.4. Seguimiento

Tarea continua durante toda el ciclo de vida del proyecto. Evalúa el estado del proyecto en función del avance en los procesos de desarrollo y de elaboración de la memoria.

1.4.3.5. Debate virtual

Tras la entrega del proyecto se inicia un debate virtual como los miembros del tribunal. Este debate tiene una duración de cinco días.

El diagrama de Gantt derivado de la planificación es el siguiente:



	Nombre de tarea	Duración	Comienzo	Fin
1	Inicio de la tarea	0 días	jue 15/03/12	jue 15/03/12
2	Instalación y configuración del entornoo	1 día	jue 15/03/12	jue 15/03/12
3	**Estudio de herramientas ya existentes**	**28 días**	**vie 16/03/12**	**jue 12/04/12**
4	Detección de errores habituales	4 días	vie 16/03/12	lun 19/03/12
5	Recopilación de herramientas más usadas	7 días	mar 20/03/12	lun 26/03/12
6	Recuperación de errores	8 días	mar 27/03/12	mar 03/04/12
7	Parsing en navegadores	9 días	mié 04/04/12	jue 12/04/12
8	**Desarrollo nueva herramienta**	**53 días**	**vie 13/04/12**	**lun 04/06/12**
9	**Validador**	**47 días**	**vie 13/04/12**	**mar 29/05/12**
10	Estudio XHTML5	2 días	vie 13/04/12	sáb 14/04/12
11	Analizador léxico	10 días	dom 15/04/12	mar 24/04/12
12	Analizador sintáctico	10 días	mié 25/04/12	vie 04/05/12
13	Analizador semántico	5 días	sáb 05/05/12	mié 09/05/12
14	Generación HTML válido	5 días	jue 10/05/12	lun 14/05/12
15	Generación XML salida	5 días	mar 15/05/12	sáb 19/05/12
16	Pruebas	8 días	mar 22/05/12	mar 29/05/12
17	**Aplicación web**	**6 días**	**mié 30/05/12**	**lun 04/06/12**
18	Diseño y Maquetación	1 día	mié 30/05/12	mié 30/05/12
19	Programación	4 días	jue 31/05/12	dom 03/06/12
20	Pruebas	1 día	lun 04/06/12	lun 04/06/12
21	**Documentación**	**88 días**	**jue 15/03/12**	**dom 10/06/12**
22	Memoria	83 días	jue 15/03/12	mar 05/06/12
23	Presentación	5 días	mié 06/06/12	dom 10/06/12
24	Seguimiento y control	83 días	jue 15/03/12	mar 05/06/12
25	Entrega PAC2	0 días	dom 15/04/12	dom 15/04/12
26	Entrega PAC3	0 días	dom 20/05/12	dom 20/05/12
27	Entrega final	0 días	dom 10/06/12	dom 10/06/12
28	Debate vrtual	4 días	lun 18/06/12	jue 21/06/12

Ilustración 3: Diagrama de Gantt

Meta parcial	Fecha
PAC 1	15/03/2012
PAC 2	15/04/2012
PAC 3	20/05/2012
Entrega final	10/06/2012
Debate virtual	18/06/22 - 22/06/2012

1.5 Breve sumario de productos obtenidos

- Validador: paquete comprimido que contiene el código fuente y los archivos compilables que implementan el validador de código XHTML5. También se incluye el código fuente de la aplicación web que actuará como interfaz entre el usuario y dicho validador.
- Memoria: documento que sigue la estructura definida previamente y detalla concienzudamente el trabajo realizado para el proyecto. Describe la metodología empleada a la hora de acometer el proyecto y la resolución del problema planteado. Su extensión máxima serán 90 páginas.

1.6 Breve descripción del resto de capítulos

Capítulo 2: Estudio de herramientas de validación HTML
Su fecha de finalización coincide más o menos con la fecha de entrega de la PAC2. Se corresponde con el primer objetivo del trabajo.

Identificaremos los errores de validación más comunes, estudiaremos herramientas de validación ya desarrolladas y analizaremos el proceso de *parsing* de los navegadores.

Capítulo 3: Analizador léxico
Primera fase de desarrollo del validador. Comprende el primer paso de la ejecución desde la recepción del código a HTML a validar hasta su descomposición en las unidades mínimas de trabajo o *tokens*.

Capítulo 4: Analizador sintáctico
Segunda fase de desarrollo del validador. Recibe como entrada la secuencia de *tokens* generada por el analizador léxico y determina si su estructura corresponde con la de documento HTML válido y si este es correcto.

Capítulo 5: Analizador semántico
Tercera fase de desarrollo del validador. Llega a trabajar de manera paralela con el analizador semántico. Parte de un documento con una estructura correcta de acuerdo a las reglas sintácticas definidas y pasa a analizar a fondo su contenido.

Capítulo 6: Generador de errores
Cubre la parte del desarrollo del validador que genera el listado de errores de validación en formato XML.

Capítulo 7: Corrector de código
Su fecha de finalización coincide más o menos con la fecha de entrega de la PAC3. Se encarga de la generación del código XHTML5 correcto.

Capítulo 8: Interfaz web
Descripción de la tarea de desarrollo de la interfaz web. Se centra en la integración del validador con una interfaz web de manera que no estemos limitados a su empleo en línea de comandos y pueda ser utilizado como una aplicación web. Por esta razón será la última tarea de desarrollo.

Capítulo 9: Plan de pruebas
Se describe brevemente y de forma general la estrategia tomada a la hora de verificar el funcionamiento del validador.

2. Estudio de herramientas de validación HTML

2.1 Introducción al HTML

HTML son las siglas de *HyperText Markup Language* o en castellano Lenguaje de Marcado de Hipertexto. Este lenguaje es el más usado en la elaboración de páginas web. Comenzó siendo un estándar *de facto* en el mundo del desarrollo web, pero su presencia se ha visto reforzada y a partir de 1995 se ha convertido en el estándar *de jure*.

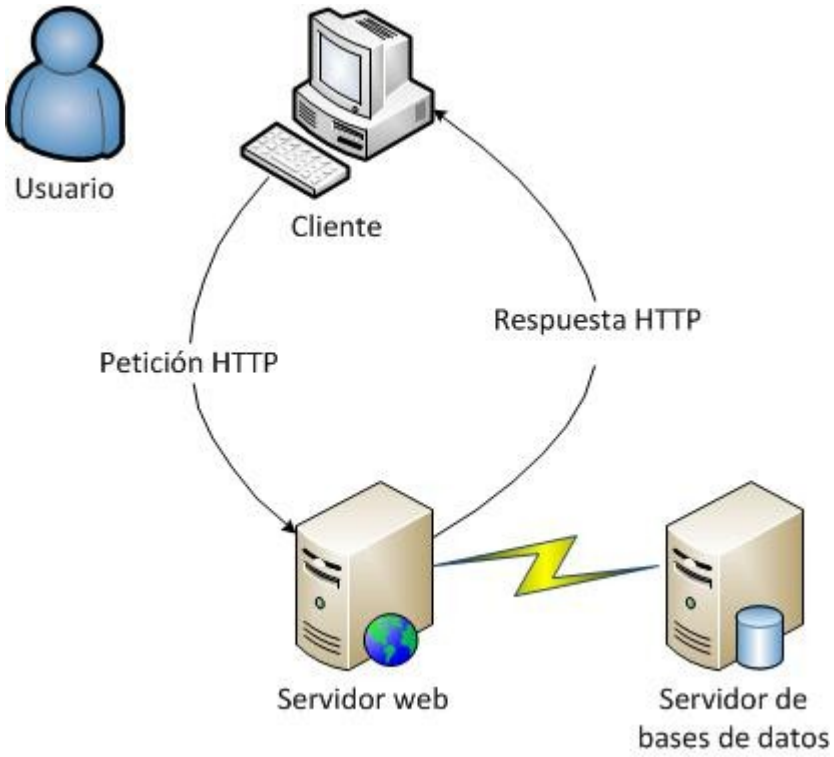


Ilustración 4: Comunicación cliente-servidor simplificada

Simplificando mucho, la World Wide Web se basa en una estructura de clientes y servidores que solicitan información y ofrecen información, respectivamente. Su diálogo se establece mediante peticiones HTTP y respuestas HTTP. El cliente, supongamos que se trata de un navegador, realiza su petición a un servidor Web determinado. Este la recibe, la trata de manera más o menos compleja en función de su naturaleza (código HTML estático o dinámico) y su tecnología y emite una respuesta codificada en HTML. El navegador la renderiza y la muestra por pantalla para que el usuario la visualice.

Como todo lenguaje de marcado su objetivo es codificar junto al contenido información referente a su estructura o su presentación. Esta información adicional es introducida mediante el uso de lo que se

conocen como marcas. El concepto de lenguaje de marcado no surgió ni mucho menos con la red, sino que es mucho anterior a ella. De hecho, el término marca surge de las anotaciones que se hacían tanto en manuscritos como en pruebas de impresión e indicaban la tipografía, el tamaño o el estilo a emplear en cada porción de texto.

HTML basa su estructura y funcionamiento en un estándar ISO previo: SGML *Standard Generalized Markup Language* o Estándar de Lenguaje de Marcado Generalizado. SGML no impone conjunto alguno de etiquetas válido, sino que se limita a definir las reglas del etiquetado del documento. Así pues, establece que el marcado ha de ser declarativo. Esto es, debe describir la estructura y atributos del documento, no el proceso al que se someterá este. De esta manera, el marcado declarativo no entrarán en conflicto con futuros usos del documento. Por otro lado, el proceso de marcado ha de ser correcto y lo suficientemente riguroso de manera que los documentos marcados puedan ser manejados del mismo modo o con las mismas técnicas que otros objetos como programas o bases datos, que está definidos de manera muy estricta.

Otro lenguaje muy implantado en la World Wide Web también está basado en SGML: XML. Tal y como SGML, XML es un metalenguaje, esto es, un lenguaje que es empleado para definir otros. XML mantiene además una estrecha relación con HTML porque a partir de él se deriva una versión de HTML conocida como XHTML. HTML y XHTML son muy similares entre sí, su principal diferencia es la más estricta estructura de este último, heredada precisamente de XML.

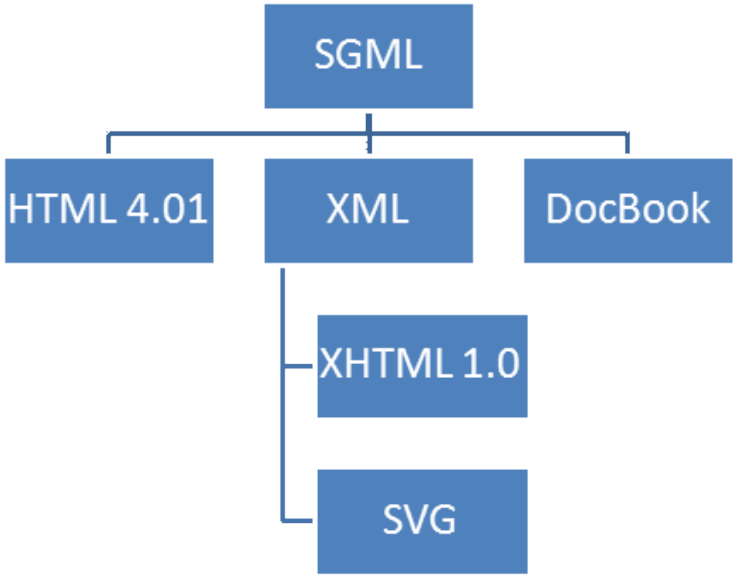


Ilustración 5: Relación jerárquica entre diferentes lenguajes

Debemos resaltar, empero, que a pesar de su similitudes y a diferencia de sus predecesores, la sintaxis de HTML5 ya no está basada en SGML. A pesar de esto, ha sido diseñado para que sea compatible con versiones anteriores de HTML.

La estructura de un documento HTML en sí es bastante sencilla y el tipo de elementos que pueden aparecer en él se reducen a cinco categorías:

- etiquetas
- atributos
- tipos de datos
- entidades HTML
- declaración de tipo de documento

Las etiquetas son la estructura básica del lenguaje HTML y se componen de contenido y de atributos. No todos ellos pueden albergar contenido, mientras que en muchos otros el contenido es opcional. La estructura de una etiqueta genérica sería la siguiente:

```
<etiqueta clave=”valor” atributo>contenido</etiqueta>
```

Si el elemento no tiene contenido:

```
<etiqueta clave=”valor” atributo/>
```

Los atributos como vemos se estructuran como pares clave-valor o bien, como elementos únicos que afectan al elemento con su mera presencia. Las comillas son opcionales.

Por otro lado, HTML define una serie de tipos de datos, especializaciones del tipo de datos de caracteres, que permite fijar el tipo de contenido de un elemento. Así pues, podremos incluir dentro de ciertas etiquetas contenido de tipo lenguaje de script o de estilos CSS.

Las entidades HTML permiten eliminar las ambigüedades a la hora de introducir caracteres reservados en el contenido de las etiquetas. El carácter '<', por ejemplo, es interpretado por el navegador como el inicio de una etiqueta, por lo que si se incluye dentro del contenido de otra, dará lugar a problemas de renderizado y el código HTML generado no será válido. No obstante, si sustituimos '<' por su entidad HTML equivalente: '`<`', el navegador lo tratará como un mero carácter, no como el inicio de una etiqueta y no se generará error alguno. Es decir, las entidades HTML nos permiten “escapar” el texto, forzar al navegador a que no lo procese, sino que lo trate como meras cadenas de texto. En la versión 4.0 de HTML ya se definían 252 entidades HTML de tipo carácter.

Finalmente, la declaración del tipo documento es imprescindible dentro de un documento HTML válido. Además, debe aparecer antes de

cualquier otro elemento del documento. Su propósito es indicar al navegador con qué tipo de documento está trabajando. En la versión 4.01 de HTML existían varios tipo de declaraciones, pero en HTML5 se ha reducido su número a uno:

```
<!DOCTYPE html>
```

Su sencillez queda aún más patente cuando la comparamos con una de las declaraciones de tipo de documento de HTML 4.01:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
```

Ahora bien, cuando hablamos de XHTML5, además de todas las características que hemos tratado hasta ahora hay que añadir las estrictas reglas de XML. Después de todo, XHTML5 es la serialización XML del lenguaje HTML5. Es por ello, que la sintaxis se hace más estricta. Por ejemplo, los atributos de las etiquetas han de ser siempre pares clave-valor y el valor ha de estar englobado por comillas simples o dobles. Además, las etiquetas siempre deberán ser cerradas, tanto aquellas que tengan contenido como las que no. Tampoco hay que olvidar que XHTML solo permite caracteres en minúsculas.

A modo de resumen, incluiremos un documento XHTML5 muy sencillo que pone en práctica los conceptos introducidos en este subapartado, si bien no incluye todos los tipos de elementos.

```
<!DOCTYPE html>
<html>
     <head>
          <title>Hola mundo</title>
     </head>
     <body>
          <p class=”first”>Hola mundo</p>
     </body>
</html>
```

2.2 Evolución histórica

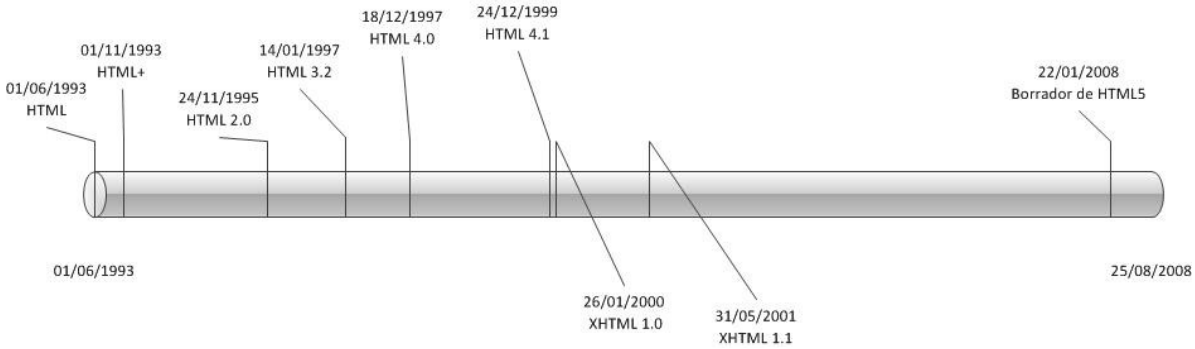


Ilustración 6: Hitos más importantes en la evolución del lenguaje HTML

La primera versión de HTML, HTML 1.0, es publicada oficialmente por la Internet Engineering Task Force (IETF) a mediados de 1993. Con anterioridad, Tim Berners-Lee había solicitado financiación al CERN junto a Robert Cailliau con el fin de desarrollar una herramienta de hipertexto. El proyecto no fue validado y la propuesta no pasó de allí, aunque Berners-Lee siguió trabajando en ese campo. Años después, como decimos, Berners-Lee junto con Dan Connolly publican a través de IETF la primera especificación HTML, que ya incluía un DTD de SGML para definir su gramática. A finales de ese mismo año, Dave Raggett publica otro borrador para HTML+ en el que ya trata cuestiones referentes a tablas y formularios.

A finales de 1994 se forma el World Wide Web Consortium (W3C) con Tim Berners-Lee a la cabeza.

Los borradores de HTML y HTML+ expiran en 1994, por lo que la IETF forma un grupo de trabajo y publica en 1995 HTML 2.0, cuya intención es servir de base para todos los estándares posteriores.

La IETF cierra en septiembre de 1996 su grupo de trabajo sobre HTML. El W3C pasa a llevar la iniciativa y publica a comienzos del años siguiente HTML 3.2 como recomendación. El primer borrador había aparecido de HTML 3 en marzo de 1995, formalizado por Dave Raggett.

En diciembre de 1996 el grupo de trabajo del W3C comienza a trabajar en Cougar, la nueva versión de HTML. Es publicada un año después como una recomendación W3C con el nombre HTML 4.0 y sus tres variantes: Strict, Transitional y Frameset.

En diciembre de 1999 es publicada la siguiente versión: HTML 4.1, también como recomendación W3C y con tres variantes.

Asimismo, el 26 de enero de 2000, el W3C publica como recomendación XHTML 1.0 con las tres variantes. El 31 de mayo del año siguiente, hace lo propio con XHTML 1.1, basado en XHTML 1.0 Strict.

En enero de 2008 es publicado HTML5 como borrador. A día de hoy continúa su desarrollo.

2.3 Errores más habituales

Como bien mencionábamos previamente, HTML5 ya no está definido a partir de SGML y tampoco a partir de XML, como XHMTL 1.0, por lo que su estructura es menos estricta que la de este ya que un documento HTML5 válido no tiene por qué ser un documento XML válido. Aún así, existen muchos errores a la hora de su validación que suelen repetirse con mucha frecuencia. A continuación los citaremos sin un orden concreto:

- Ausencia de la declaración del tipo de documento
  Al enunciar los diferentes elementos de un documento HTML definíamos precisamente el primer elemento que debe aparecer en un documento HTML: la declaración del tipo de documento. Es un elemento imprescindible porque indica de qué versión de HTML se trata, de manera que el navegador puede renderizar su contenido apropiadamente. En versiones anteriores, por ejemplo HTML 4.01, esta declaración hacía referencia a un Document Type Definition o DTD. En la nueva versión ya no se hace referencia a DTD alguno y la línea ha ganado sencillez.

- Estructura básica
  La etiqueta title es indispensable en un documento HTML válido. Además, debe de estar incluida dentro de la cabecera (declarada con la etiqueta head) al comienzo del documento, tras la declaración del tipo de documento. Como title, otras etiquetas como meta o link solo pueden ser incluidas en la cabecera, si no queremos generar errores de validación.

- Entidades HTML
  Es imperativo convertir los caracteres especiales en entidades HTML con el fin de que el navegador escape esos caracteres y no intente tratarlos como si fuesen parte del código. Una entidad HTML nunca 'romperá' de esta manera el código. Los caracteres más problemáticos suelen ser “<”, “>” y “&”. Para evitar dicho problema habrá que sustituirlos por sus entidades HTML correspondientes: “<”, “>” y “&”. Como vemos todas siguen el mismo patrón: comienzan con un ampersand y finalizan con un punto y coma.

- Uso de etiquetas y atributos inválidos
  Evidentemente, el conjunto de etiquetas disponibles es limitado y es obligatorio ceñirse a ellas. Además, cada una de ellas podrá ir acompañada de unos atributos concretos. El hecho de no seguir estas normas. Implicará errores de validación. Son muy

frecuentes con la introducción de nuevas versiones, ya que muchas etiquetas desaparecen y otras tantas son definidas. Concretamente en HTML5 han desaparecido las etiquetas acronym, applet, basefont, big, center, dir, font, frame, frameset, isindex, noframes, strike, tt. Por otro lado, nuevas etiquetas han sido añadidas al estándar: article, aside, audio, bdo, canvas, command, data, datalist, details, embed, figcaption, figure, footer, header, hgroup, keygen, mark, meter, nav, output, progress, rp, rt, ruby, section, source, summary, time, track, video, wbr. En el Anexo I trataremos con más detenimiento todas ellas.

- Anidamiento incorrecto
  Las etiquetas de un documento HTML5 han de cerrarse en el orden opuesto al que se han abierto. En otro caso, la estructura del documento será errónea y se generarán errores de validación.

- Cierre de etiquetas incorrecto
  Toda etiqueta con contenido ha de seguir la siguiente estructura:

```
<etiqueta>contenido</etiqueta>
```

  Esto es, toda etiqueta que se abra, ha de cerrarse. A diferencia de XHTML, HTML es más permisivo y permite que las etiquetas simples sin contenido no necesiten cierre. Así pues, todas las siguientes etiquetas serían válidas:

```
<etiqueta>
<etiqueta/>
<etiqueta />
```

  Esta mayor permisividad del código HTML puede producir despistes en los desarrolladores a la hora de trabajar con código XHTML. En este caso, solo el último ejemplo es válido.

```
<etiqueta />
```

- Ausencia de atributos requeridos
  Ciertas etiquetas requieren las presencia de ciertos atributos concretos. Por ejemplo, la etiqueta img, que permite incluir imágenes en los documentos HTML, requiere dos atributos: src y alt. El primero sirve para indicar la ubicación de la imagen en sí. No suele dar problemas porque si no se introduce correctamente, la imagen no se visualiza y el error se detecta de manera inmediata. El que suele ser olvidado con más frecuencia es alt, que fija el texto alternativo que ha de mostrarse en caso de que la imagen no pueda ser mostrada por la razón que sea.

- Atributos no válidos

El hecho de que una etiqueta pueda recibir una serie de atributos, no implica que el resto de etiquetas también pueda recibirlo. Un ejemplo de este comportamiento son los atributos width y height de las etiquetas img. Muchos editores WYSIWYG las introducen de manera incorrecta en otras etiquetas para maquetar el contenido, por lo que en ocasiones es necesario es depurarlo.

- Estructura incorrecta de los atributos
  Como bien hemos mencionado antes, XHTML sigue la sintaxis mucho más estricta del XML, por lo que los atributos han de tener siempre la estructura par clave-valor, manteniendo el valor entre comillas.

- Estructura incorrecta de las etiquetas HTML
  Las etiquetas se pueden clasificar en 2 grandes grupos: elementos de bloque y elementos en línea. En un documento XHML válido los elementos de bloque pueden albergar tanto otros elementos de bloque como elementos en línea. Los elementos en línea, en cambio, solo pueden contener otros elementos en línea. Un elemento de bloque dentro de un elemento en línea generaría un error de validación.

  Por otro lado, las tablas presentan una estructura muy concreta que es necesario respetar si queremos un código XHTML válido:

```
<table>

<thead>
<tr>
    <th scope="col">Enemies List</th>
</tr>
</thead>

<tfoot>
<tr>
    <td>&copy; Bomb Voyage</td>
</tr>
</tfoot>

<tbody>
<tr>
    <td>Mr. Incredible</td>
    <td>Elastigirl</td>
    <td>Gazer Beam</td>
</tr>
</tbody>
```

```
</table>
```

Ni `thead` ni `tfoot` son elementos indispensables pero en caso de aparecer han de hacerlo en ese orden y con esa estructura. Tbody, por su parte sí es indispensable.

- Atributos `id`, `class` y `name`
  El valor de estos atributos ha de ser un literal que puede incluir caracteres numéricos, pero nunca comenzar con uno.

2.4 Herramientas, paquetes y librerías

Dado que el estándar XHTML5 se encuentra todavía en desarrollo, no existen herramientas disponibles para su verificación. No obstante, sí existen una serie de recursos disponibles para la verificación de versiones anteriores, tanto de HTML5 y posteriores, como de XTHML 1.0, por ejemplo. A continuación trataremos las más relevantes y frecuentes en el ámbito de la programación web. Comenzaremos con el validador oficial desarrollado por el W3C, para pasar a analizar el resto sin un orden establecido.

El principal problema de los diferentes validadores no es detectar los errores de validación, sino ser capaces de reponerse y continuar la validación del documento sin que dichos errores supongan un freno insalvable para su ejecución. Para comprobarlo hemos elaborado 10 casos de prueba de acuerdo al listado de errores más frecuentes que elaboramos en epígrafes anteriores. Resumámoslos:

1. Ausencia de declaración del documento
2. Estructura básica del HTML incorrecta
3. Uso incorrecto de entidades HTML
4. Etiquetas y/o atributos inválidos
5. Anidamiento incorrecto
6. Cierre incorrecto de etiquetas
7. Ausencia de atributos obligatorios
8. Estructura incorrecta de los atributos
9. Estructura incorrecta de etiquetas HTML
10. Valor incorrecto de los atributos

De acuerdo a este listado, hemos elaborado cada caso de prueba de la siguiente manera: el primer error del documento será del tipo que indica el punto del listado en el que nos encontremos. Asimismo, incluiremos tras este primer error más errores HTML. El objetivo en definitiva no es solo comprobar si el validar detecta el error, sino además asegurarnos de que el error inicial no echa al traste su ejecución y este puede seguir validando. Dado que la mayoría de validadores soportan más de una

versión y existen además algunos que validan de manera experimental, haremos 3 versiones por cada caso de prueba; uno para HTML5, otro para HTML 4.01 y, finalmente, otra para XHTML 1.0. Por claridad, para nombrarlos seguiremos la nomenclatura versión_de_HTML-nº_del_caso_de_prueba. Por ejemplo, el caso de prueba cuyo primer error sea etiquetas y/o atributos inválidos en HTML5 será el caso de uso HTML5-4. En la sección de anexos de este documento podremos encontrar el código de cada uno de ellos.

A continuación incluiremos una tabla resumen que recoja de manera sucinta la información que desarrollaremos en los subapartados siguientes.

Herramienta	**XHTML 5**	**HTML 5**	**HTML 4.01**	**HTML 3.2**	**HTML 2.0**	**XHTML 1.1**	**XHTML 1.0**	**Otros**
W3C Markup Validation Service		X	X	X	X	X	X	X
Validator.nu	X	X	X					X
WDG HTML Validator			X	X	X	X	X	X
HTML Tidy			X	X	X	X	X	X
Total Validator	X	X	X	X	X	X	X	X
Relaxed			X					
Validome			X	X	X	X	X	X

2.4.1 W3C Markup Validation Service[1]

El W3C Markup Validation Service es el validador por excelencia. Es desarrollado por el W3C y colaboradores desinteresados, apoyados a su vez por donantes entre los que destaca HP. Su potencia no se limita únicamente a la validación de código HTML/XHTML, sino que también permite comprobar la validez de, por ejemplo, documentos SMIL y MathML. Como bien establecimos en los puntos anteriores, todo este tipo de lenguajes, hasta la versión 1.1 de XHTML y la versión 5 de HTML heredan sus características y funcionamiento de SGML. Este tipo de lenguajes ven definidas su estructura mediante Document Type Definition o DTDs. El funcionamiento del validador se centra en cotejar la organización del documento y la gramática que se supone que este sigue. Toda estructura que se salga de lo especificado por la gramática será considerado un error de validación.

Dentro de su gran versatilidad, la herramienta ofrece tres modos de funcionamiento: a través de URI, a través de archivo y por entrada directa.

1 http://validator.w3.org/

En el modo de trabajo a través de URI el validador recibe como entrada la URI del documento que queremos validar, además de una serie de opciones de configuración sobre su funcionamiento. Así podremos indicarle la codificación de caracteres, el tipo de documento. También podemos definir el formato de salida de los errores de validación encontrados. Podremos listarlos de manera secuencial o agruparlos por tipo. También podremos decidir mostrar el código fuente que estemos validando y que los errores sean subrayados en este. Finalmente, este validador incorpora la funcionalidad de HTML Tidy, de manera que junto al listado de errores de validación también puede devolver una posible versión del código corregido. El único inconveniente es que HTML Tidy aun no trabaja con HTML5 por lo que en caso de trabajar con esta versión, el código de salida será HTML 4.01.

Tras haber ejecutado los casos de prueba esta herramienta resulta la más estable y más eficiente. Si acaso un tanto incómoda de usar al tener que acudir siempre al sitio web del W3C. No obstante, existen *add-ons* o *plug-ins* para navegadores que permiten enlazar automáticamente con esta con una sencilla combinación de letras. Su único fallo es cierta confusión a la hora de gestionar el anidamiento incorrecto de etiquetas HTML.

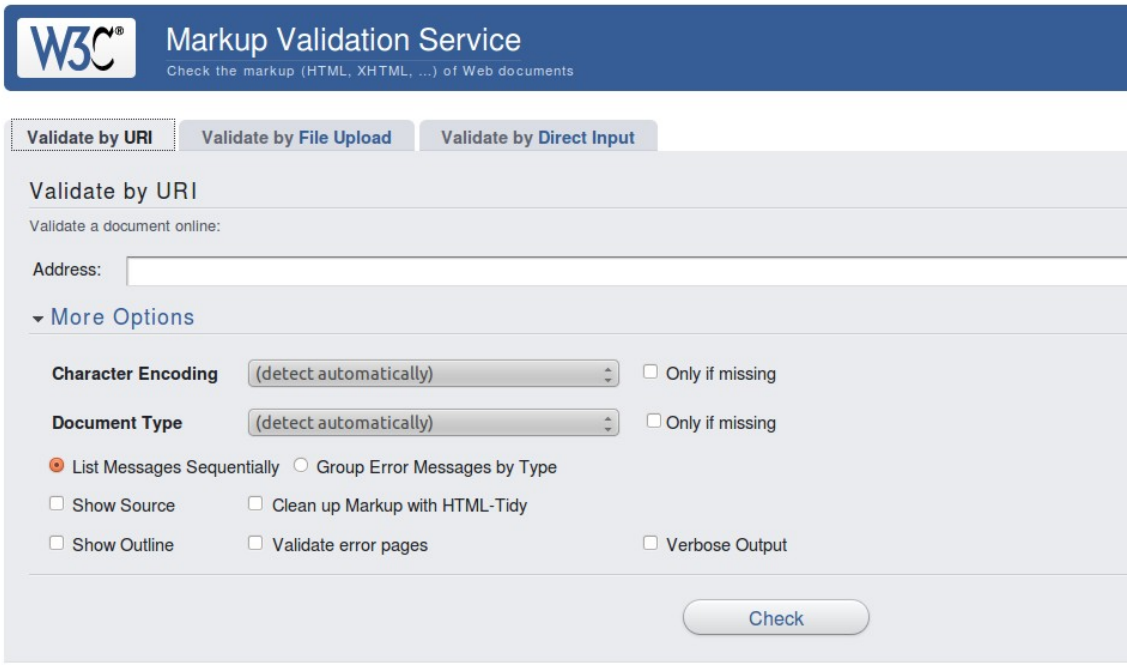


Ilustración 7: Markup Validation Service a través de URI

En el modo de trabajo con archivo, el validador recibe como entrada un archivo sobre el que trabajará de manera análoga. También es posible especificarle las mismas opciones de funcionamiento que en el caso anterior.

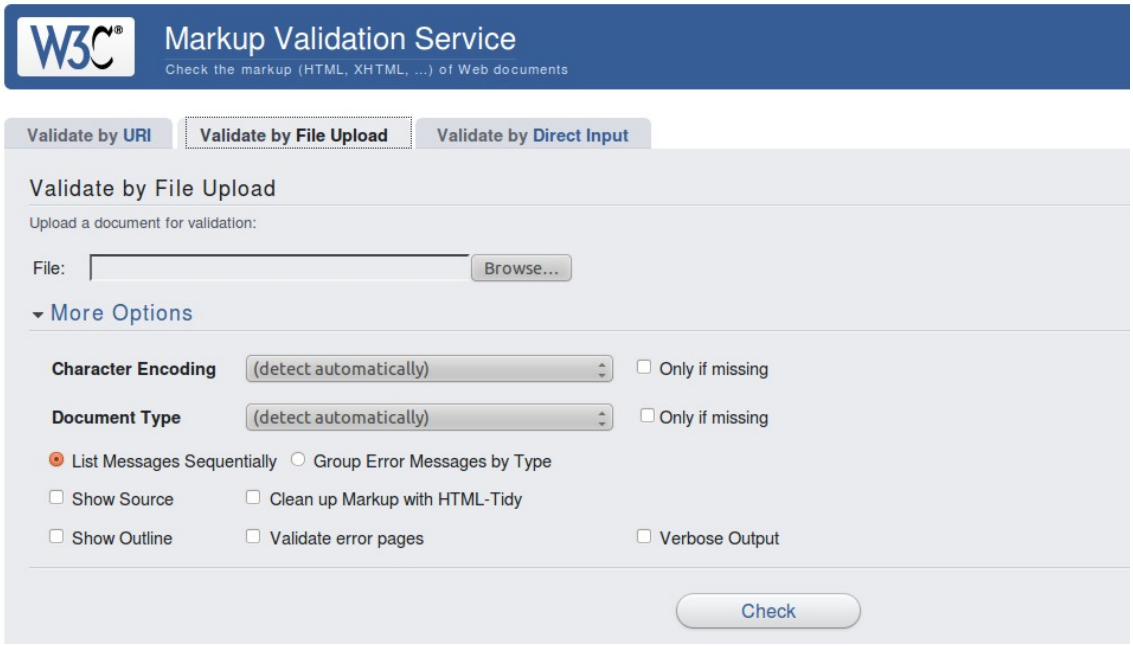


Ilustración 8: Markup Validation Service a través de archivo

Finalmente, en el modo de trabajo por entrada directa, dispondremos de un cuadro de texto en el que insertar directamente el código que queramos validar. En este caso, el validador nos ofrece más potencia ya que no solo permite trabajar con documentos completos, sino que permite validar fragmentos. En este caso, no obstante, su funcionamiento se ve limitado única y exclusivamente a fragmentos de documentos HTML 4.01 y XHTML 1.0. Como en los dos casos anteriores, se le pueden especificar una serie de opciones a aplicar sobre el formato de salida de los errores.

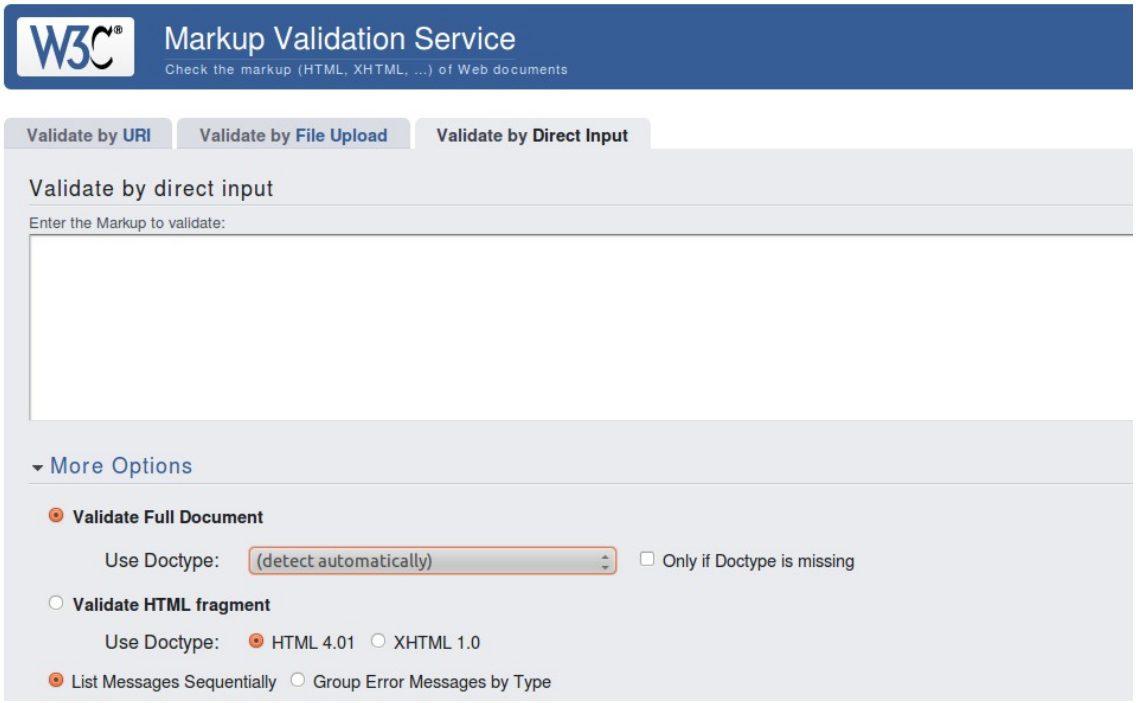


Ilustración 9: Markup Validation Service a través de entrada directa

2.4.2 Validator.nu[2]

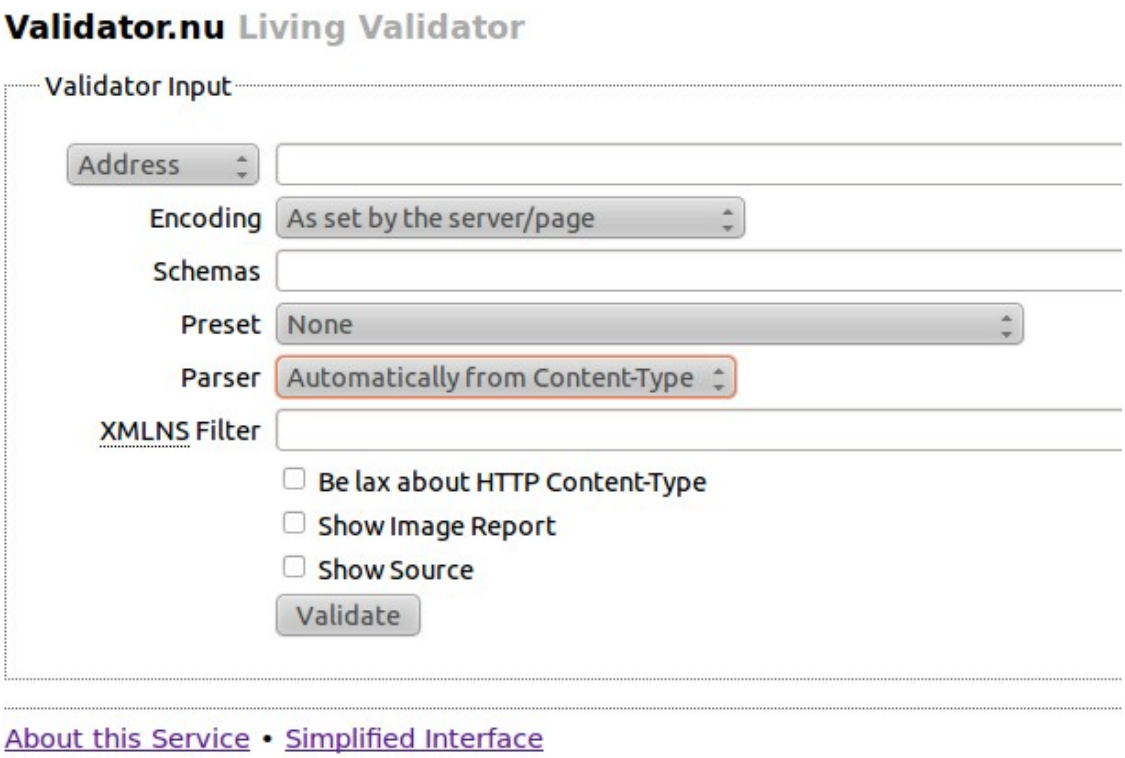


Ilustración 10: Pantalla genérica de Validator.nu

2 http://validator.nu/

Validator.nu tiene dos modos de trabajo: genérico y (X)HTML5. En ambos modos puede trabajar con URIs, archivos o entrada directa. Resumiendo, estos son los esquemas con los que Validator.nu puede trabajar:

- HTML5 (experimental)
- HTML5+ARIA (experimental)
- HTML 4.01 Strict + IRI / XHTML 1.0 Strict + IRI
- HTML 4.01 Transitional + IRI / XHTML 1.0 Transitional + IRI
- HTML 4.01 Frameset + IRI / XHTML 1.0 Frameset + IRI s. Do not use. :-)
- XHTML5 (experimental)
- XHTML5+ARIA, SVG 1.1 plus MathML 2.0 (experimental) cruft.
- XHTML 1.0 Strict, SVG 1.1, MathML 2.0 + IRI
- XHTML 1.0 Strict, Ruby, SVG 1.1, MathML 2.0 + IRI
- XHTML Basic + IRI
- SVG 1.1 + IRI

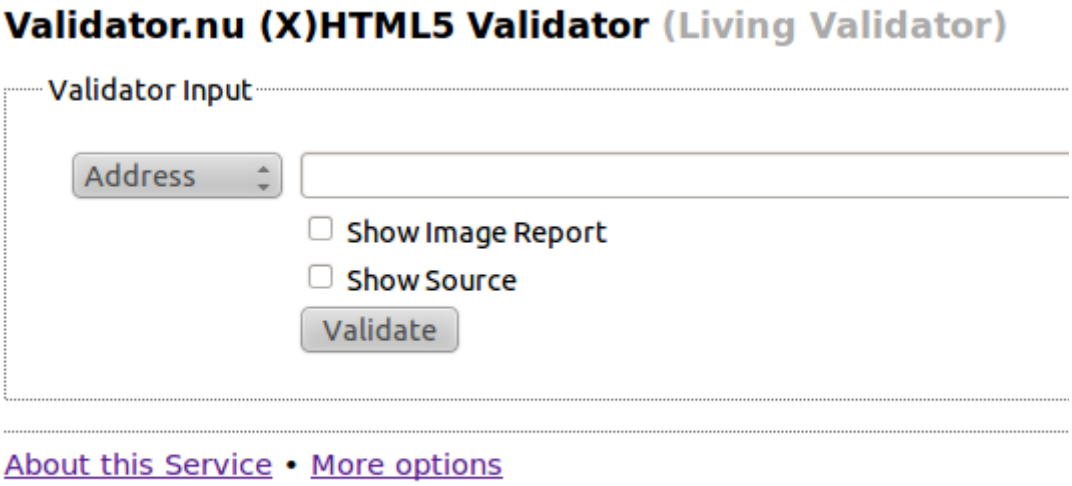
En el caso del validador genérico podemos fijar una serie de parámetros de configuración del proceso de validación:

- Schemas. Cuando este campo es dejado en blanco, el validador tratará de escoger uno él mismo.
- Parser. Nuevamente, el usuario puede especificar un parser en concreto. En caso contrario el propio validador escogerá uno automáticamente.
- Be lax about HTTP Content-Type. Cuando esta opción está activada se permiten los tipos text/html, text/xsl, text/plain como content types de XML. En el caso de HTML, se permite el text/plain.
- Show Image Report. Si esta opción está marcada, se mostrará las alternativas textuales a todos los elementos de tipo imagen. Esta opción es especialmente interesante cuando nos centremos en el trabajo con la accesibilidad.
- Show source. En este caso, esta opción generará como salida el código decodificado a partir de la entrada. Si el validador encuentra un error fata, la salida no será completa.

En el modo de trabajo (X)HTML5[3], las opciones se reducen únicamente a dos:

- Show Image Report.
- Show source.

3 http://html5.validator.nu/



La herramienta ofrece una calidad es su labor validadora a la altura del W3C Markup Validator Service. Sin embargo, carece de los añadidos de este que le otorgan una gran ventaja y una mayor funcionalidad.

2.4.3 WDG HTML Validator[4]

Este validador es muy similar en funcionamiento al W3C Markup Validation Service. En un comienzo existían claras diferencias, pero a medida que el validador de W3C ha ido evolucionando, estas se han reducido a meras cuestiones menores. Su funcionamiento se basa completamente en el uso de DTDs y no soporta el manejo de lenguajes que no hereden de SGML. Es por ello que no trabaja con HTML5. Como en el caso del validador del W3C, existen los tres modos de trabajo: URI, archivo y entrada directa. Las dos opciones de configuración comunes a los tres modos de configuración son:

- Include warnings. Además de los errores de validación, se mostrarán las advertencias.
- Show input. Además del informe de errores, se mostrará también el código con el que se trabaja.

4 http://www.htmlhelp.com/tools/validator/

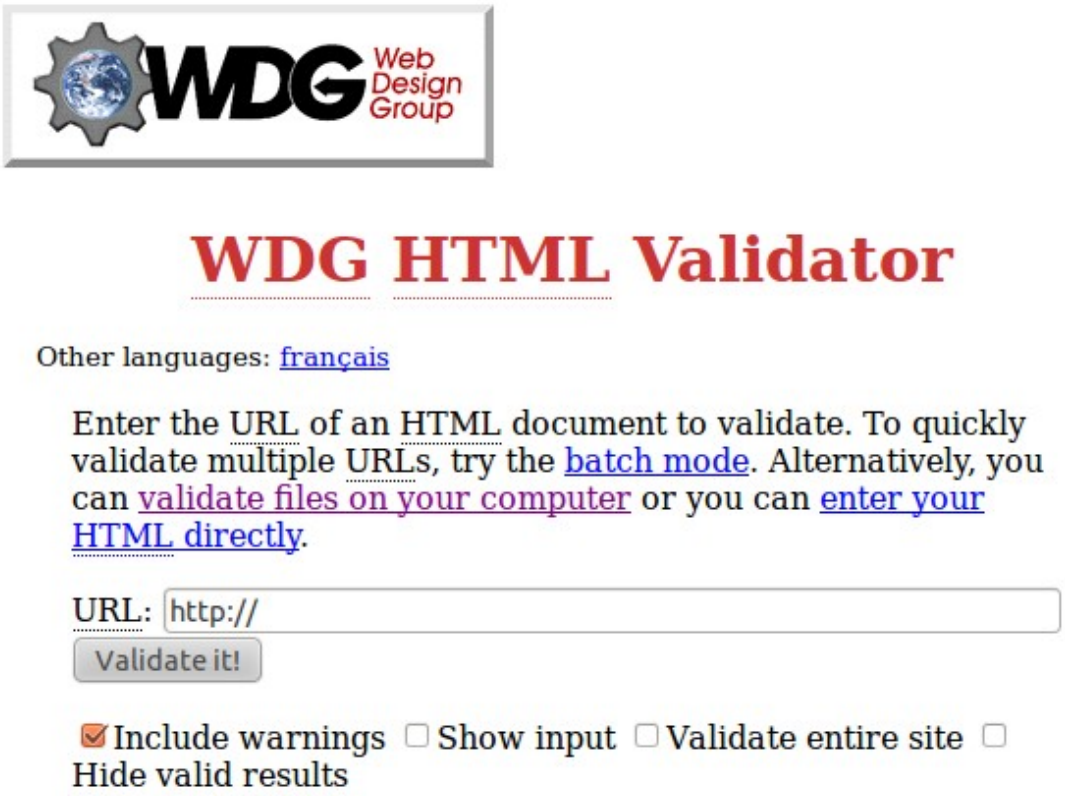


Ilustración 11: WDG Validation en modo URI

En el modo de trabajo con URI, se incorporarán otras dos opciones:

- Validate entire site. En caso de marcar esta opción, no solo se validará únicamente la URI de partida, sino que se analizarán todas las que cuelguen de ella. Por cada una de estas páginas se mostrará un informe.
- Hide valid results. En el caso de que se active la opción anterior, el listado de páginas hijas puede ser demasiado extenso. Habilitando esta opción, nos aseguraremos de que el informe solo incluye aquellas páginas con algún tipo de error.

Esta herramienta no soporta aún la validación de código HTML5 y su informe de errores está poco cuidado. La única ventaja sobre otras herramientas es la posibilidad de validar un sitio completo a través de los enlaces.

2.4.4 HTML Tidy[5]

Tidy fue en su origen un proyecto personal de Dave Raggett, aunque ahora ha pasado a ser continuado por un grupo de desarrolladores que han continuado con su implementación en el marco de SourceForge. Se

5 http://tidy.sourceforge.net/

trata de un programa y una librería escrito en C cuyo principal objetivo es recibir como entrada HTML con posibles errores y generar como salida un código HTML válido. Entre los diferentes errores que es capaz de solucionar se encuentran:

- Etiquetas mal cerradas o cierres ausentes.
- Comillas o atributos no presentes.
- Uso de caracteres reservados en vez de entidades HTML.

Además, como resultado de este proceso de mejora del código HTML, y aunque esto no sea un error, el código generado puede presentar una mejor identación.

Un ejemplo de llamada muy sencilla a Tidy podría ser:

```
tidy -f errs.txt -m index.html
```

El archivo index.html contiene el código HTML a tratar, que además será actualizado con las correcciones convenientes. Por esta razón es convenientes realizar una copia previa. Por otro lado, con el flag -f indicamos la ruta del archivo donde se almacenará el informe de errores.

En sitios como http://infohound.net/tidy/ y http://services.w3.org/tidy/tidy podemos encontrar aplicaciones online que hacen uso de esta librería.

El uso de la herramienta de manera directa resulta tedioso por ser necesario el empleo de la línea de comandos. Su verdadera utilidad reside en la integración con otras para ofrecer una experiencia más completa. Ejemplo de esto es su incorporación al W3C Markup Validation Service. Su única carencia es la falta de soporte por ahora de la versión 5 de HMTL. Aparte de esto, la única falta de la que perece adolecer es un tratamiento un tanto simplista del anidamiento incorrecto de las etiquetas HTML, optando siempre por declarar una etiqueta global que englobe todas en vez de considerar alterar el orden de las etiquetas de cierre.

2.4.5 Total Validator[6]

Esta herramienta es mucho más potente que un simple validador HTML. Sus funcionalidades incluyen:

- Validación del código
- Validación de accesibilidad (WCAG 1.0 y 2.0)
- Validación CSS
- Comprobación de enlaces rotos

6 http://www.totalvalidator.com/

- Corrección ortográfica
- Integración con FireFox como add-on.

Las versiones de (X)HTML que puede validar son entre otras:

- HTML5
- XHTML5
- HTML5 políglota
- XHTML 1.1
- XHTML 1.0 (Basic, Strict y Transitional)
- HTML 4.01 (Basic, Strict y Transitional)
- HTML 4 (Basic, Strict y Transitional)
- HTML 3.2
- HTML 2.0

Entre sus opciones de salida podemos decidir que se muestren únicamente los errores, o bien también las advertencias.

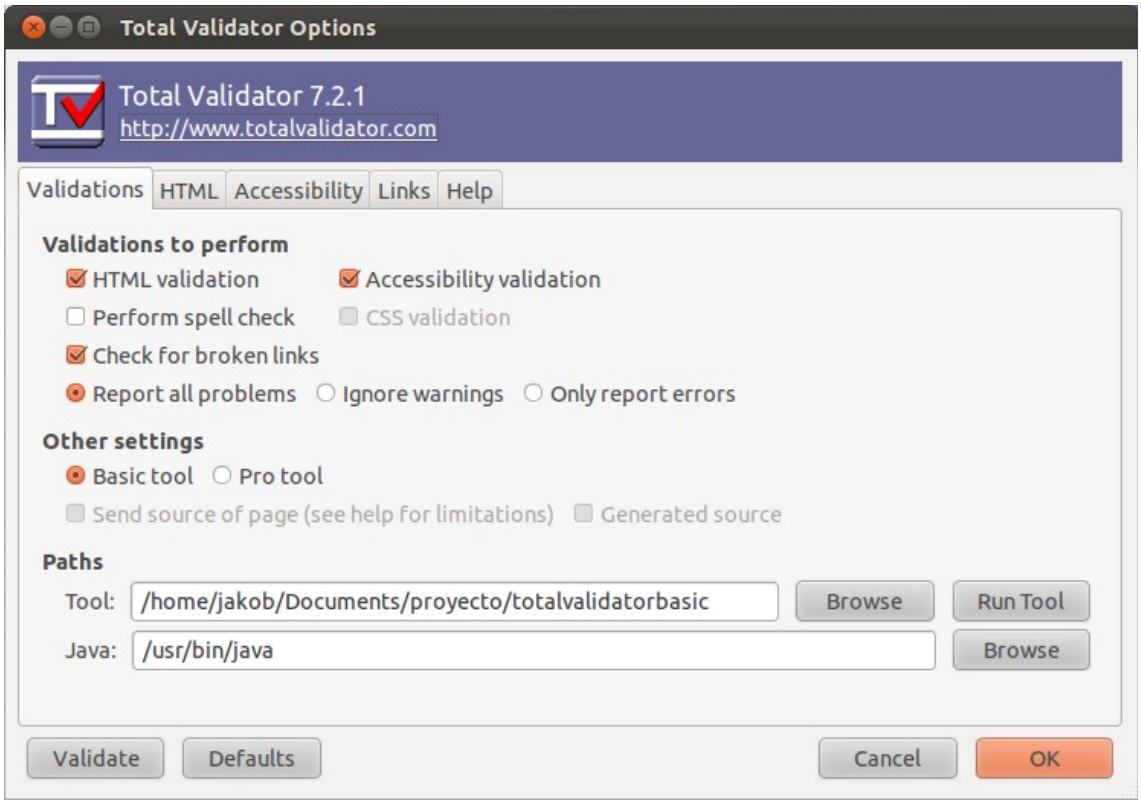


Ilustración 12: Pantalla principal de Total Validator

Esta herramienta es especialmente interesante para todo desarrollador web, precisamente por la posibilidad de lanzarlo directamente desde el navegador y por todas las funciones que acompañan a la validación del código HTML en sí: comprobación de enlaces rotos, verificación ortográfica, validación de la accesibilidad. El único inconveniente es

cierto lentitud en la velocidad de procesamiento del documento, especialmente si añadimos más validaciones que la estricta del código HTML.

2.4.6 Relaxed, the HTML validator[7]

Relaxed está escrito en Java y hace uso de lenguajes de schema como Relax NG y Schematron. El uso de este tipo de lenguajes más modernos y el abandono de los DTDs hace posible que se lleva a cabo la validación automática de restricciones mucho más complejas que las recogidas en dichos DTDs y que se expresan de palabra en el texto de los diferente estándares. Gracias a la combinación de estos dos lenguajes pueden expresarse restricciones demasiado complejas para los DTDs. En la actualidad, Relaxed puede validar documentos que siguen los estándares HTML 4.01 y XHTML 1.0. En todo caso, es importante resaltar que la validación de HTML 4.01 la lleva a cabo realizando una conversión previa de HTML a XHTML. Relaxed permite dos modos de trabajo: validación por URI o por archivo. El informe de errores indica el número de línea del error, una breve explicación y su nivel de gravedad.

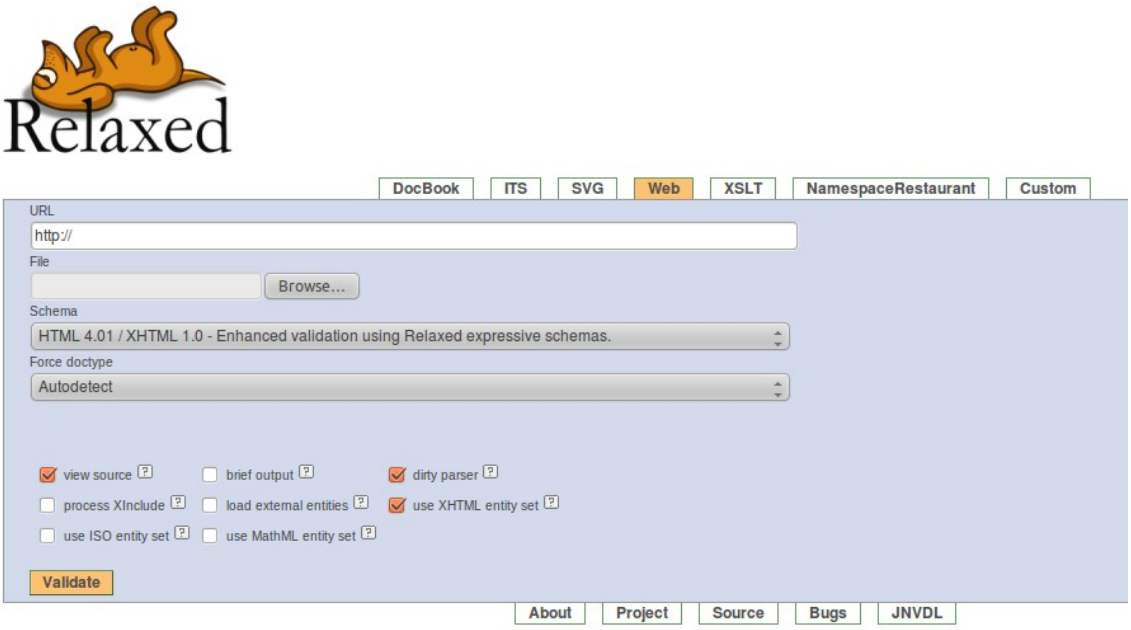


Ilustración 13: Pantalla principal de Relaxed

Su comportamiento como validador deja bastante que desear. La detección de un error de validación ofusca todos los errores de validación posteriores. De esta forma, casos de prueba que con los otros validadores generaban 11 errores, con esta herramienta generan solo dos. En la mayoría de los casos solamente uno. Decantarse por esta opción para validar documentos puede convertir el trabajo en una tarea

7 http://relaxed.vse.cz/relaxed/validate?group=Web

tediosa y poco productiva y, en definitiva, en una pérdida de tiempo En resumen se trata de un validador muy poco recomendable.

2.4.7 Validome

http://www.validome.org/

Validome ofrece validación de HTML, XHTML y WML. En concreto, todas las versiones soportadas son:

- XHTML 1.1
- XHTML 1.0 (Basic, Strict y Transitional)
- XHTMl Basic
- HTML 4.01 (Basic, Strict y Transitional)
- HTML 4 (Basic, Strict y Transitional)
- HTML 3.2
- HTML 2.0
- WML (1.0, 1.1, 1.2, 1.3 y 2.0)

Permite trabajar en modo URI y archivo. En su informe de errores se incluye también un listado de posibles advertencias.

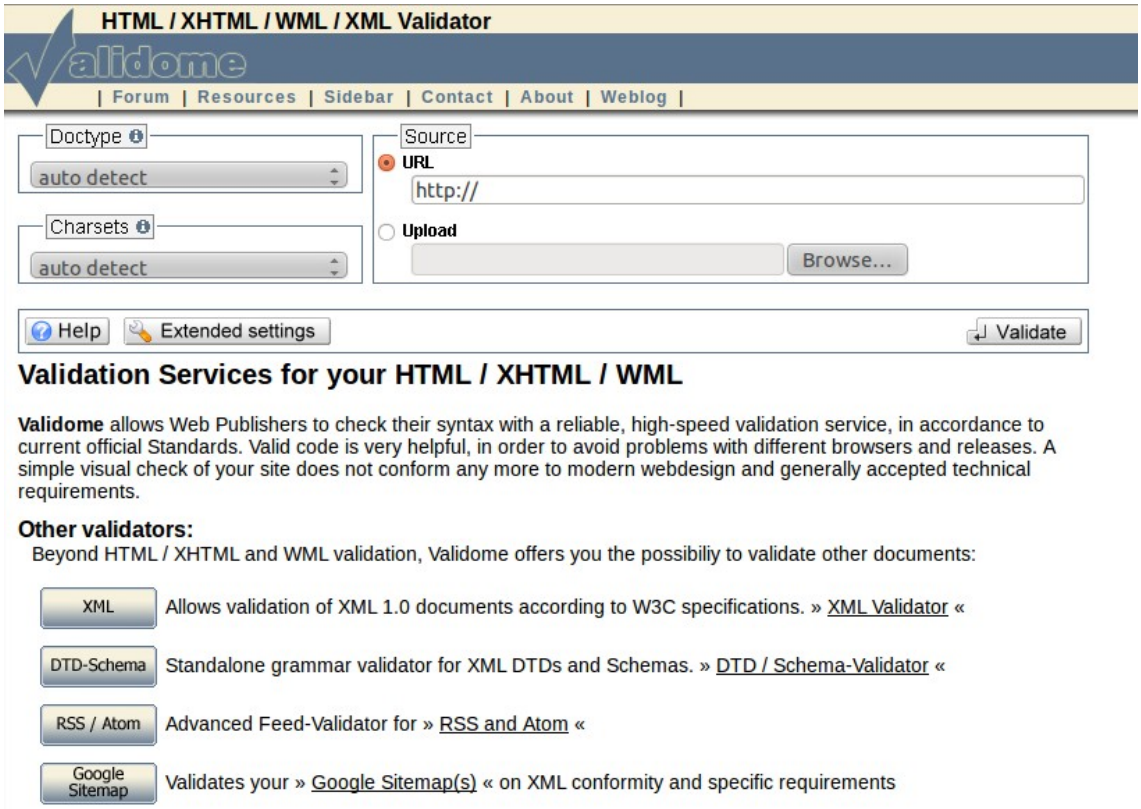


Ilustración 14: Pantalla principal de Validome

Se trata de un validador correcto pero que palidece frente a los que hemos tratado hasta ahora. A diferencia de los anteriores, no valida documentos en los que no esté presente la declaración del tipo. Por lo demás su funcionamiento es aceptable, sin añadidos ni florituras.

2.5 El *parsing* en los navegadores actuales

Como podemos prever el proceso que lleva a cabo un navegador para "entender" el código HTML es un proceso complejo en el que intervienen numerosos elementos diferentes. A grandes rasgos el navegador debe leer el código HTML que desea mostrar, parsearlo y generar a partir de él árbol DOM. El proceso no termina aquí ya que el código HTML seguramente contenga numerosos contenidos enlazados. Dentro de esta categoría podemos encontrar elementos como imágenes, hojas de estilo, scripts u otro tipo de archivos como películas de flash, o archivos de audio o vídeo. También ha de encargarse de organizar la estructura de la página. Los diferentes elementos han de ser ubicados, han de tener un tamaño determinado, un color. El texto ha de tener una tipografía concreta, un interlineado, una alineación, etc. Finalmente, toda esa información ha de combinarse para ser ofrecida al usuario, se procede así al renderizado. Hemos de resaltar, que si bien la enumeración de los diferentes pasos se ha hecho de manera secuencial, el proceso no tiene por qué serlo. Los scripts pueden modificar el contenido del documento a medida que este es cargado. Para hacerle el proceso de carga de la página no todos los elementos tienen que ser cargados inmediatamente. De esta forma, las imágenes puede ser descargadas a medida que el usuario a medida que vayan siendo noticias. Además, todo este funcionamiento ha de llevarse a cabo teniendo en cuenta las restricciones de seguridad aplicables en cada situación.

Ilustración 15: Fases del proceso de renderizado

¿Cómo analizar el código HTML? Esa es la gran pregunta que cabe responder llegados a este punto. Como es de esperar, el modo de hacerlo no es arbitrario. Para garantizar una compatibilidad óptima entre los diferentes navegadores, estos deberían seguir las especificaciones fijadas por el W3C. Hemos usado el condicional de manera consciente, ya que en el pasado no todos los navegadores seguían todas estas especificaciones, lo que generaba grandes quebraderos de cabeza para los desarrolladores. Estas incompatibilidades resultaban en la necesidad de desarrollar versiones diferentes del mismo documento web para los diferentes navegadores, debido a que, al contrario de lo que cabría esperar, un mismo código HTML con unos estilos CSS determinados no mostraban la misma apariencia en todos ellos. Es por ello que con HTML5 se ha prestado y se está prestando especial atención a las especificaciones publicadas por W3C, para evitar la pérdida de productividad generando ajustes y en ocasiones hojas de estilo específicas para las diferentes versiones de los navegadores.

No debemos perder de vista que cierta permisividad no es del todo una mala opción a la hora de trabajar con HTML. Después de todo, el propósito es mostrar información, aún a pesar de que su estructura no sea totalmente válida. Un modo de operar tan estricto como el de un compilador de lenguajes de Java, por ejemplo, sería poco funcional. Sin embargo, esta permisividad también produce situaciones en las que el mismo código HTML genera resultados visuales muy diferentes. Pongamos como muestra un ejemplo extraído de http://css-tricks.com/. Si partimos de un elemento p con unos estilos muy definidos obtendremos resultados muy variables en diferentes navegadores:

```
p {
	font-family: Verdana;
	background-color: #7A2121;
	color: #B93333;
	text-decoration: underline;
	word-spacing: Normal;
	text-align: left;
	letter-spacing: 1px;
	text-indent: 15px;
	line-height: 16px;
	font-size: 10px;
	font-weight: bold;
	font-style: italic;
	text-transform: uppercase;
}
```

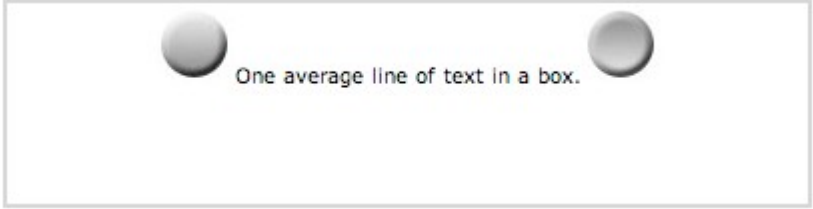


Ilustración 16: Mac Firefox 2

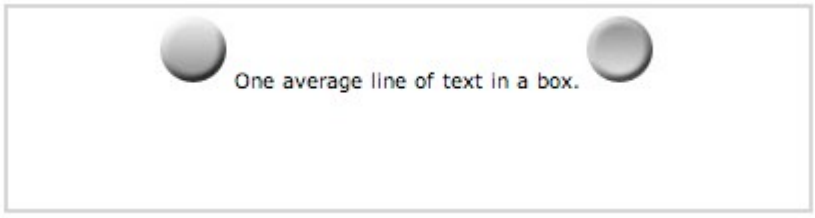


Ilustración 17: Mac Safari 2

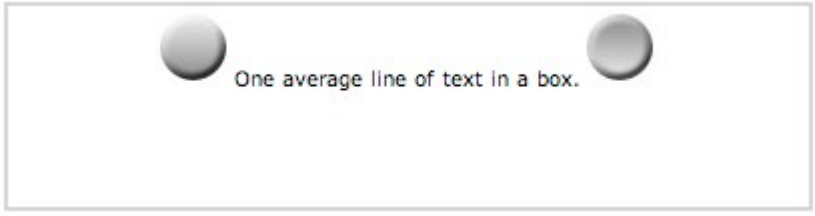


Ilustración 18: Mac Opera 9

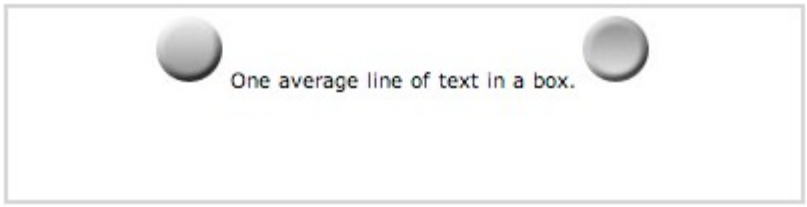


Ilustración 19: Mac IE 5

Ilustración 20: PC IE 6 (Windows 2000)

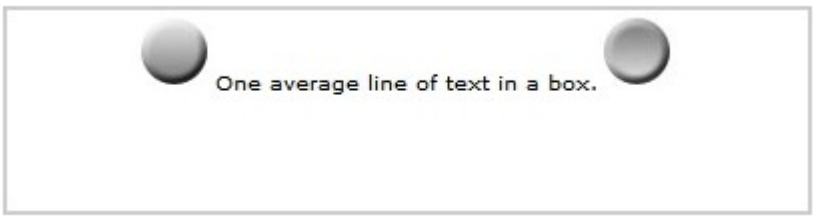


Ilustración 21: PC IE 7

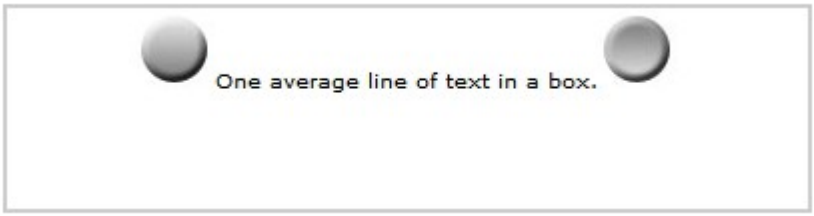


Ilustración 22: PC IE 8

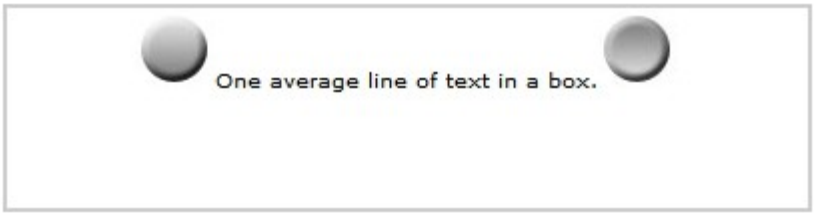


Ilustración 23: PC Firefox

Como podemos comprobar, a pesar de haber fijado unos estilos muy restrictivos, los resultados no son todo lo exactos que cabría pensar.

Estamos refiriéndonos constantemente a los navegadores, pero en realidad no son estos los responsables de esta tarea. De ello se encargan los denominados motores de renderizado, los cuales se encuentran incrustados dentro de los propios navegadores, pero también dentro de toda aplicación que esté destinada a mostrar o editar contenido web. Otro ejemplo de este tipo de aplicaciones son los clientes de e-mail. Si bien existen numerosos navegadores, el número de motores de renderizado es menor, puesto que existen navegadores que comparten un mismo motor de renderizado. Por ejemplo, si consultamos información referente a los navegadores más usados podremos comprobar que son 4 los motores de renderizado que copan el mercado.[8]

8 http://news.cnet.com/8301-30685_3-57407891-264/microsofts-ie-reclaims-lost-ground-in-browser-battle/

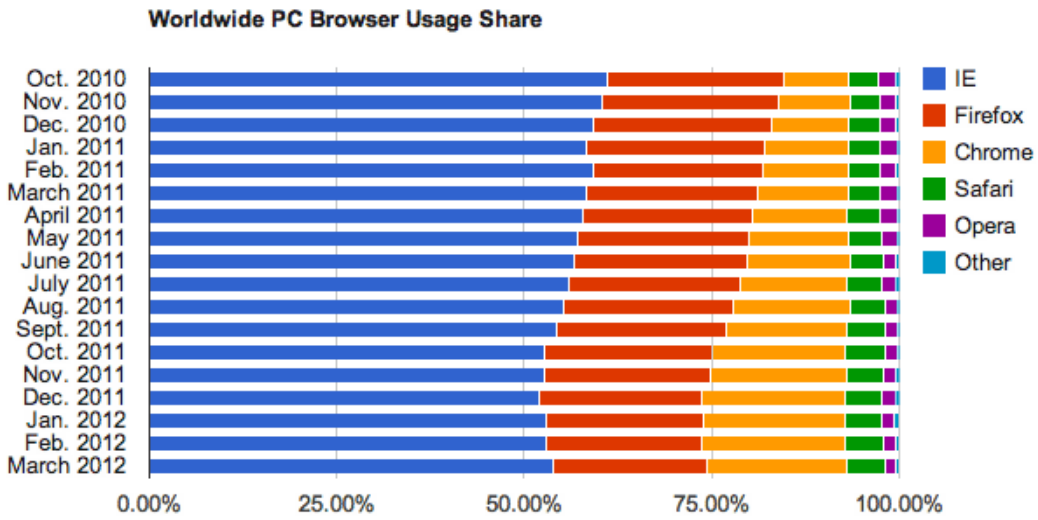


Ilustración 24: Evolución del porcentaje de uso de los navegadores web

Internet Explorer hace uso del motor de renderizado Trident. Firefox por su parte emplea Gecko. Webkit, desarrollado a partir del motor de renderizado abierto KHTML de KDE, sirve tanto a Chrome como a Safari. Por último, Opera emplea su propio motor de renderizado: Presto.

2.5.1 Trident

Como bien decíamos, un motor de renderizado no solo resulta útil en un navegador web. La funcionalidad de Trident, por ejemplo, no solo está incluida en Internet Explorer. Los gestores de correo Outlook Express y Microsoft Outlook trabajan con Trident. No pensemos en cualquier caso que su uso se limita a desarrollos de Microsoft. Clientes de mensajería instantánea como Skype y GTalk, así como los navegadores incluidos en los reproductores Winamp (Bento Browser) y RealPlayer (RealNetworks) también hacen uso de él.

Las primeras versiones de Internet Explorer no hacen uso de Trident, ya que este fue incorporado únicamente a partir de la versión 4.0. Precisamente Microsoft fue destino de numerosas críticas debido a la falta de compatibilidad de estas primeras versiones de su navegador. Internet Explorer 6 resultaba especialmente difícil en este aspecto y requería gran cantidad de tiempo añadido para optimizar la visualización del contenido. Gran parte se dedicaba a la definición de estilos propios para esta versión, debido a sus tristemente conocidos problemas con el renderizado de CSS, A partir de la versión 7 decidió intensificar sus esfuerzos para intentar subsanar estas cuestiones e intentar seguir más fielmente los estándares web. Ante la aparición de nuevos navegadores más competitivos y afrontando una pérdida de mercado lenta pero constante, han redoblado sus esfuerzos con las versión 9. Como apunte

técnico añadiremos que el lenguaje de programación escogido para su desarrollo es C++. Su soporte de HTML5 y CSS3 es aún limitado.

2.5.2 Gecko

Gecko, como Trident, está escrito en C++. A diferencia de este, es multiplataforma, por lo que puede ser ejecutado bajo numerosos sistemas operativos, todo un punto a su favor. Entre ellos se encuentran Linux, Mac OS X, Solaris, BSDs, OS/2, AIX, OpenVMS y por supuesto Microsoft Windows. Además de en Mozilla y todos sus navegadores derivados (por ejemplo, Firefox y Camino), podemos encontrarlo en en el gestor de correo Thunderbird y la aplicación SeaMonkey. La versión de Picasa para Linux también está basada en Gecko.

Su desarrollo comenzó dentro de Nestcape en 1997. Por aquel entonces se enfrentaban a un único competidor: Internet Explorer y su motor de renderizado era claramente superior. En 1998 el nuevo motor de renderizado fue liberado con licencia de código abierto lo que permitió que aparte del propio navegador, otras aplicaciones pudieran hacer uso de él.

Mientras que Trident era conocido sobre todo en el pasado por su poca adhesión por los estándares web, Gecko[9] adolece precisamente de lo contrario. Sus detractores critican su soporte de funcionalidades no estándar, que si bien aumentan la compatibilidad con documentos escritos especialmente para Internet Explorer. A su juicio, esta permisividad hace un flaco favor a la implantación de los estándares web.

Su soporte de HTML5 es aún limitado.

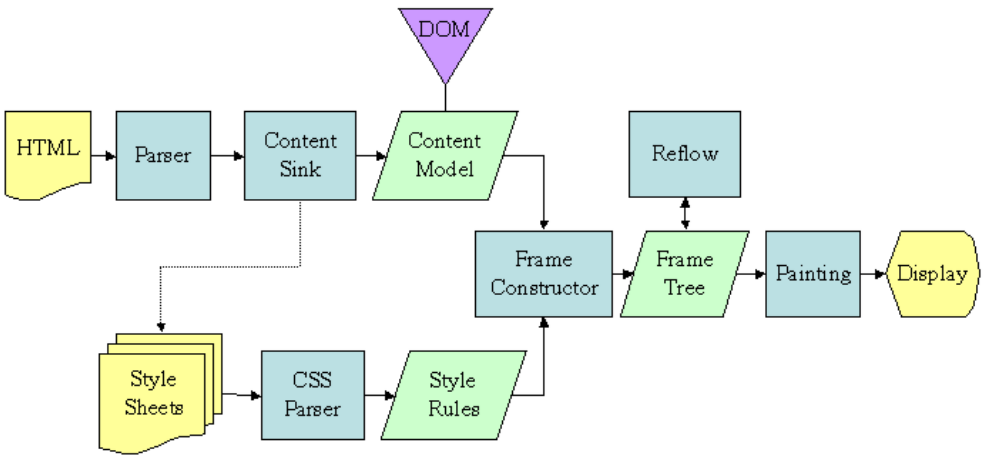


Ilustración 25: Esquema de funcionamiento de Gecko

9 http://www-archive.mozilla.org/newlayout/doc/gecko-overview.htm

2.5.3 WebKit

A pesar de que Firefox es el segundo navegador más usado, Webkit es el segundo motor renderizado más empleado, tras el omnipresente Trident. Este segundo puesto en el podio se debe a que tanto Safari como Chrome hacen uso de su funcionalidad. Su origen se encuentra en las librerías KHTML y KJS de KDE escritas en C++ cuyo desarrollo comenzó en 1998. El proyecto WebKit[10] se inicia en 2001 dentro de Apple. Los factores que llevaron a partir de estas librerías fue la claridad del diseño, el menor número de líneas de código y un mejor seguimiento de los estándares. El desarrollo de ambos proyectos se realizó en paralelo con desarrolladores de ambos equipos trabajando de forma conjunta, hasta que diferentes enfoques en la codificación así como ciertas incomodidades surgidas con las políticas de Apple obligaron a separar definitivamente ambos desarrollos. En la actualidad, ambos continúan por separado.

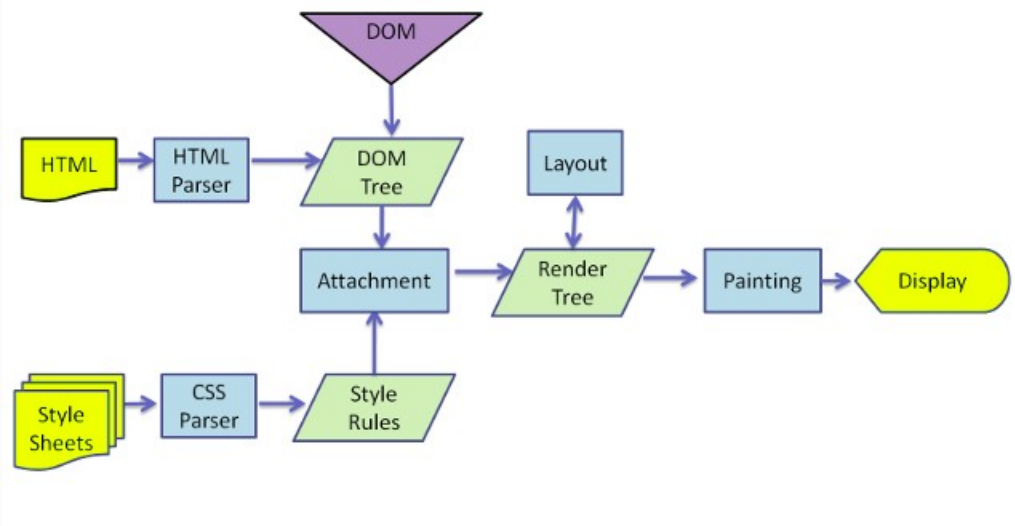


Ilustración 26: Esquema de funcionamiento de WebKit

2.5.4 Presto

Presto es el motor de renderizado de las últimas versiones del navegador Opera. Fue publicado en 2003 de la mano de la versión 7 del navegador para Windows. Los navegadores de las consolas Nintendo DS, Nintendo DSi y Wii también hacen uso de Presto. Dreamweaver de Macromedia también usaba Presto en un principio, pero a partir de la versión CS4 ha optado por WebKit. Su difusión se ve limitada debido a que se trata de software propietario, pero a diferencia de Trident no va de la mano de un navegador tan extendido como lo está Internet Explorer.

10 http://taligarsiel.com/Projects/howbrowserswork1.htm

3. Analizador léxico

Para acometer la implementación del analizador léxico del validador de código XHTML5 haremos uso de JLex. Se trata de un generador automático de analizadores léxicos. Está realizado en Java y genera asimismo código Java. Nos hemos decantado por esta opción ya que conocemos de sobra su potencia al haber tratado con él en las asignaturas Compiladores I y Compiladores II. Su funcionamiento consiste en la traducción de un fichero fuente escrito en lenguaje JLex a un fichero objeto en lenguaje Java.

El funcionamiento del analizador léxico es implementado en un único archivo, denominado fichero de especificación léxica de JLex. Este fichero se organiza en tres secciones separadas por "%%". Así tenemos:

- Código de usuario. Esta sección se transfiere directamente al analizador léxico generado. Normalmente se incluyen sentencias de importación de paquetes y definiciones de clases.
- Directivas de JLex. Las directivas permiten particularizar el funcionamiento del analizador generado, definir macros y listar los estados.
- Reglas basadas en expresiones regulares. Son las reglas léxicas que definen el funcionamiento del analizador léxico. Su estructura es la siguiente:

  [Lista de estados (opcional)][Expresión Regular][Acción léxica]

3.1 Código de usuario

En el caso concreto que nos compete la sección de código de usuario incluye una serie de sentencias `import` con los que añadimos funcionalidad ya desarrollada. No obstante, lo más relevante en las declaración de la clase `Obj`, que consta de 3 atributos:

- `name`
- `line`
- `content`

El propósito de esta clase es representar los *tokens* en los que se dividirá el texto de entrada del validador. Así pues, `name` almacenará el nombre del token y `line` la línea. El uso de `content` se reservará para los `tokens` de tipo texto.

3.2 Directivas JLex

En esta sección fijaremos el nombre de la clase generada con

```
% class Final
```

También definiremos los dos estados por lo que podrá el analizador léxico:

```
%state COMMENT, TAG
```

El modo en el que el analizador entrará o saldrá de estos estados estará fijado mediante las reglas léxicas. Finalmente, también crearemos una serie de macros que permitirán una definición de estas mucho más limpia. Por ejemplo:

```
blank=[\n\r\t ]
character = [a-zA-Z0-9_\-'\"!]
extended_character = [^'\"]
```

3.3 Reglas léxicas

Como establecimos previamente, las reglas léxicas definen el comportamiento básico del analizador léxico. Son por definición, el elemento más importante del fichero de especificación léxica.

```
<YYINITIAL>{blank}+ { /* Do nothing */ }
<YYINITIAL>"<"      { /* Inside an HTML tag */
  yybegin(TAG);

return new Symbol(sym.TK_LT, new Obj((yyline+1),
yytext()));
}
<TAG>['\"]{extended_character}+['\"]   {
                                        return new
Symbol(sym.TK_STRING, new Obj((yyline+1), yytext()));
                                       }
<TAG>{character}+ {
                                        return new
Symbol(sym.TK_LITERAL,       new       Obj((yyline+1),
yytext()));
}
<TAG>"="{                                          retu
rn   new   Symbol(sym.TK_EQ,   new   Obj((yyline+1),
yytext()));
}
<TAG>">" { yybegin(YYINITIAL);
                                        return new
Symbol(sym.TK_GT, new Obj((yyline+1), yytext()));
```

```
}
<TAG>"/" {
                                                    return new
Symbol(sym.TK_SLASH, new Obj((yyline+1), yytext()));
}
<TAG>[ \r\t]+    { /* Do nothing */ }
<TAG>\n {yybegin(YYINITIAL);}
<YYINITIAL><!-- {
/* Inside HTML comment */
yybegin(COMMENT);
}
<COMMENT>--> { /* Outside HTML comment */
yybegin(YYINITIAL);
}
<COMMENT>. { /* Ignore comment contents */ }
<YYINITIAL>[^<]+ { /* Text */
                                                    return new
Symbol(sym.TK_TEXT,     new     Obj((yyline+1),     "",
yytext()));
}
```

Así pues, el analizador recorre el texto de entrada, compara el texto de entrada con los patrones, teniendo en cuenta el estado que se encuentra y devuelve un testigo. Los comentarios son ignorados. El tipo de testigos es limitado, solo existen 7:

- TK_TEXT: representa un texto
- TK_LITERAL: representa una cadena de texto
- TK_EQ: el carácter =.
- TK_GT: el carácter >.
- TK_LT: el carácter <.
- TK_STRING: : una cadena de texto englobada por comillas.
- TK_SLASH: la barra lateral /.

Podemos enfocar el analizador léxico como una máquina de estados finitos:

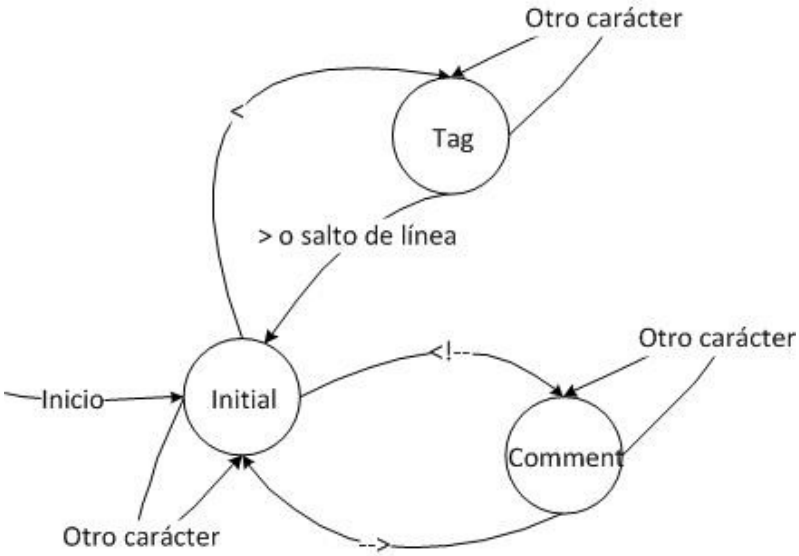


Ilustración 27: El analizador léxico como máquina de estados finitos

4. Analizador sintáctico

Si para acometer el desarrollo del analizador léxico hicimos uso del generador de analizadores léxicos JLex, a la hora de hacer lo propio con el analizador sintáctico aprovecharemos la funcionalidad de CUP (*Java Based Constructor of Useful Parsers*). CUP es un generador de analizadores sintácticos LALR, que puede complementar la funcionalidad desarrollada por un analizador sintáctico mediante JLex. Como este tiene una estructura definida:

- Sentencias de importación y empaquetamiento.
- Código de usuario. Está permitido añadir código al analizador sintáctico. Es muy usual incorporar funcionalidades de análisis semántico en esta sección para complementar el funcionamiento del analizador.
- Lista de símbolos. Esta lista se incluirá tanto los elementos terminales como los no-terminales empleados en la definición de la gramática.
- Declaraciones de precedencia. De manera opcional puede definirse la precedencia y la asociatividad de los terminales.
- Gramática. La última sección es la de mayor relevancia y contiene la gramática del lenguaje que se pretende analizar. La estructura de estas reglas es la siguiente:

    - no-terminal ::= secuencia_de_terminales_y/o_noterminales [{: código_java :}]

    - [secuencia_de_terminales_y/o_noterminales
      [{: código_java :}]]

    - [asignación_contextual_de_precedencia]
    - [| secuencia_de_terminales_y/o_noterminales [{: código_java :}]]

    - [asignación_contextual_de_precedencia]

No tiene mucho sentido incluir en este documento todas las reglas de la gramática, dado que se extendería demasiado. No obstante, explicaremos el funcionamiento del analizador sintáctico incorporando línea de código concretas, con el fin de no hacerlo muy pesado.

Mediante el uso de la directiva %cup en el archivo JLex habilitaremos la comunicación entre este y el archivo CUP. Así pues, el analizador sintáctico recibirá como entrada el texto pretratado ya por el analizador léxico en forma de *tokens* o testigos. El sintáctico los comparará con las reglas sintácticas definidas en la sección de gramática y ejecutará el

código Java asociado. Por ejemplo, existe una regla sintáctica que define la estructura de una etiqueta de apertura:

```
open_tag ::=  TK_LT TK_LITERAL:e1 attributes TK_GT    {:
                                        parser.handleOpenTag((Obj) e1);
                                            :}
        | TK_LT TK_STRING:e1 attributes TK_GT    {:
                                        parser.handleOpenTag((Obj) e1);
                                            :}
        | TK_LT TK_LITERAL:e1 TK_GT   {:
                                        parser.handleOpenTag((Obj) e1);
                                            :}
        | TK_LT TK_STRING:e1 TK_GT  {:
                                        parser.handleOpenTag((Obj) e1);
                                            :}
```

Como vemos, en caso de que la entrada coincida con alguna de estas 4 estructuras, el analizador sintáctico llamará a la función auxiliar handleOpentTag que recibirá como parámetro una instancia de la clase Obj. Es el analizador léxico el encargado de esa inicialización como vimos en el apartado anterior.

La mecánica del funcionamiento será siempre la misma. El documento XHTML5 puede ser concebido como una sucesión de etiquetas simples o complejas que pueden contener dentro de ellas a su vez más elementos (otras etiquetas o texto). No olvidemos que las etiquetas también pueden ir acompañados de atributos. Para afrontar la comprobación de todos elementos haremos uso de funciones auxiliares. Como hemos visto, `handleOpenTag` es una de ellas. La mayoría de ellas realizan una serie de comprobaciones para garantizar la corrección sintáctica del documento. En total son las cinco siguientes:

- `handleAttribute`. Gestiona los atributos de las etiquetas. Es llamada cada vez que se encuentra un atributo de una etiqueta, ya sea de apertura, de cierre o autocerrada. Añade el atributo a la tabla hash `attributeTable`. En caso de encontrar atributos repetidos, almacenará el error. Si el atributo no tiene una estructura del tipo clave=”valor”, también se generará el error.

- `handleOpenTag`. Gestiona las etiquetas de apertura. En caso de existir atributos, estos ya habrán sido introducidos en la tabla hash `attributeTable`, por lo que podrá desentenderse de su gestión. Además añade una entrada a la pila `scopes` con el fin de dejar constancia de que a continuación pasaremos a analizar el contenido de la etiqueta.

- `handleCloseTag`. Gestiona las etiquetas de cierre. Al igual que `handleOpenTag` no deberá trabajar con los atributos,

simplemente comprobar la cima de la pila `scopes`. En caso de que las dos etiquetas no concuerden, estaremos ante un error de anidamiento.

- `handleTag`. Gestiona las etiquetas autocerradas. En este caso no es necesario comprobar la cima de la pila. Como las dos anteriores, ya recibirá la tabla con los atributos convenientemente introducción.

- `handleText`. Gestiona los elementos de texto plano, esto es contenido de texto de las etiquetas.

5. Analizador semántico

En el capítulo 4 pudimos ver como dentro de la estructura de un fichero CUP se encuentra una sección reservada para el código de usuario. En esta sección ya habíamos definido funciones auxiliares que nos simplificaban la tarea del análisis sintáctico del documento. También la aprovecharemos para implementar la funcionalidad del analizador semántico. Hasta ahora el analizador únicamente se centraba en comprobar que el documento XHTML5 consistía en una serie de etiquetas anidadas. El análisis semántico irá más allá e intentará comprobar que no solamente la estructura del documento es correcta, si no que su significado sea correcto.

En esta sección analizaremos funciones auxiliares cuyo principal objetivo es el análisis semántico, si bien en algunos casos llevarán a cabo tareas que podrían ser englobadas en el ámbito del análisis sintáctico. En un afán por mantener lo más ordenada posible esta memoria se incluirán únicamente en este capítulo.

Primeramente, alguna de las funciones auxiliares cuyo comportamiento ya describimos en el capítulo 4 ven ampliadas sus competencias. En concreto son aquellas encargadas de manejar los tres tipos de etiquetas: `handleTag`, `handleOpenTag` y `handleCloseTag`. En estas tres funciones se comprueba el nombre de la etiqueta y se emite un error en caso de que la etiqueta no permita el autocierre o el contenido. Por ejemplo, la etiqueta div no permite el autocierre. Por esta razón, `<div/>` no es una etiqueta válida. La función `handleTag` recibiría esta etiqueta y generaría el error correspondiente. Por otro lado, `img` es una etiqueta que no permite contenido. En caso de encontrar la etiqueta `<img>`, la función `handleOpenTag` realizaría la comprobación y generaría el error. Asimismo, si apareciera la etiqueta de cierre `</img>`, la función encargada de gestionarla y generar el error sería `handleCloseTag`.

Además de estas mejoras a funciones auxiliares, se han implementado nuevas funciones que realizan comprobaciones adicionales:

- `checkTag`. Esta función es llamada desde las tres funciones auxiliares encargadas las etiquetas: `handleTag`, `handleOpenTag` y `handleCloseTag`. Se encarga de comprobar que la etiqueta es una etiqueta aceptada por HTML5. Etiquetas inventadas o pertenecientes a versiones más antiguas generarán el error correspondiente y serán ignoradas. También se aseguran de que aquellas que sí son permitidas estén en minúsculas. En caso de encontrarse algún carácter en mayúscula, el validador emitirá el error pero no ignorará la etiqueta.

- `checkContext`. Es llamada a continuación de `checkTag`. Una vez que se ha comprobado que la etiqueta en sí correcta, pasamos a comprobar que la etiqueta esté permitida en su ubicación. Para ello haremos uso nuevamente de la pila con el fin de averiguar dentro de qué etiqueta nos encontramos. Cada etiqueta tendrá una entrada en una tabla hash, en la que su valor asociado será una lista que contendrá todas las etiquetas permitidas. Si la etiqueta actual no se encuentra dentro de la lista asociada a la etiqueta que se encuentra en la cima de la pila, se generará un error.

- `checkTree`. Una vez que se ha tratado el documento y se han llevado las comprobaciones de todas las funciones anteriores, `checkTree` se encarga de llevar a cabo una comprobación a nivel global del documento. En concreto, verifica la presencia de etiquetas imprescindible dentro de todo documento XHTML5 válido: la declaración, la cabecera y el título. Además, verifica que dichas etiquetas, además de la etiqueta `body`, no se encuentren repetidas.

6. Generador de errores

Hemos ido haciendo referencia a medida que describíamos el funcionamiento de los analizadores léxico, sintáctico y semántico al hecho de “generar errores”. En las etapas iniciales del desarrollo ese comportamiento se limitaba a imprimir un mensaje por salida estándar. En el enunciado del trabajo se especifica, no obstante, que el formato ha de ser XML con una estructura libre. Nos decantaremos por una sencilla de la forma:

```
<errors>
     <error line=”0” message=”Descripción del error y
               medidas tomadas para repararlo” />
     ...
</errors>
```

Para acometer la tarea usaremos primeramente una estructura de almacenamiento auxiliar. Nos valdrá para esto una lista. Para representar los errores haremos uso de la clase `Obj` que ya habíamos definido en el fichero JLex. Ahora, en vez de usar para representar una etiqueta, lo usaremos para representar un error. En el campo `line` almacenaremos la ubicación del error y en el campo `name`, la descripción del error, además de la decisión tomada para subsanarlo.

Una vez finalizada la ejecución del *parser*, dispondremos de una lista con todos los errores encontrados a lo largo del análisis del documento. El paso natural es extraer esta información con el fin de elaborar el requerido documento XML. Para ello tendremos que incluir una serie de librerías que nos permitirán llevar a cabo la tarea de manera muy sencilla.

Primero necesitaremos un documento DOM en el insertar nuestro código XML.

```
errorsDoc = docBuilder.newDocument();
```

A continuación será necesario crear un nodo raíz a partir del cual cuelguen el resto de elementos.

```
currentErrorNode = errorsDoc.createElement("errors");
errorsDoc.appendChild(currentErrorNode);
```

Iteraremos sobre la lista de errores para crear los nodos y colgarlos de la raíz.

```
 for (Iterator it = errors.iterator(); it.hasNext();)
{
        Obj entry = (Obj) it.next();
```

```
                        addToErrorTree(entry.getLine(),
entry.getName());
      }
```

La función `addToErrorTree` recibe como parámetros un entero y una cadena de texto. Crea el nodo y le asigna los atributos. No obstante, si el entero es igual 0, interpreta el error como global y no añade el atributo.

```
  public static void addToErrorTree(int line, String
message){

    Element node = errorsDoc.createElement("error");
    if(line != 0) {
                              node.setAttribute("line",
String.valueOf(line));
    }
    node.setAttribute("message", message);
    currentErrorNode.appendChild(node);

  }
```

Finalmente será necesario convertir el árbol XML en un `OutputStream` o `Writer` que nos permitirá imprimirlo por pantalla.

```
sw = new StringWriter();
      result = new StreamResult(sw);
                  DOMSource   errorsSource   =   new
DOMSource(errorsDoc);
       trans.setOutputProperty(OutputKeys.OMIT_XML_DEC
LARATION, "no");
      trans.transform(errorsSource, result);
      String errorsXmlString = sw.toString();
```

7. Corrector de código

El último paso en el desarrollo del validador es la generación de código XHMTL5 reparado. No solamente hemos de informar de los errores que hemos encontrado, sino que además hemos de eliminarlos, incluso aunque eso implique la pérdida de información. Dicha funcionalidad será incorporada e integrada dentro de las funciones auxiliares que ya han sido desarrolladas con el fin de detectar los errores.

Si tenemos en cuenta que un documento XHTML5 cumple las restricciones de un documento XML, veremos que podemos adoptar el mismo enfoque que el tomado para generar el listado de errores en formato XML. Inicialmente crearemos el documento:

```
dbfac = DocumentBuilderFactory.newInstance();
docBuilder = dbfac.newDocumentBuilder();
doc = docBuilder.newDocument();
```

La diferencia con la generación de errores es que en este caso no disponemos de una estructura que recorrer para sencillamente ir añadiendo los nodos. Tendremos que añadirlos cada vez que detectemos un elemento. Afortunadamente, ya existen funciones encargadas de tratar estos elementos:

- `handleTag`
- `handleOpenTag`
- `handleCloseTag`
- `handleText`

No obstante, existen un pequeño detalle que debemos tener en cuenta. En el caso del XML de errores, su estructura es muy básica ya que todos los nodos colgaban de la raíz. Al tratar con el XHTML5 reparado debemos tener en cuenta que la estructura será mucho más compleja y que el nodo del que colgarán los elementos variará a lo largo de la ejecución. Para ello nos ayudaremos de una pila:

```
static Stack<Element> nodes;
```

En las funciones handleTag, handleOpenTag y handleTag se crea el nodo. Si la pila está vacía, se tomará ese nodo como raíz del documento. En caso contrario, el nuevo nodo será descendiente del almacenado en la cima de la pila. La pila solo se verá modificada al tratar una etiqueta de apertura o una etiqueta de cierre.

```
  public static void addToTree(Obj tag) {
    //Si la etiqueta no tiene nombre, es un nodo de texto
    if(tag.getName().equals("") ){
        String str = tag.getContent().toString();
        Text text = doc.createTextNode(str);
          //Si no hay nodo en la pila, añado el nodo como
raíz
        if(nodes.empty()) {
            doc.appendChild(text);
        }
           //Si hay nodo en la pila, añado el nuevo como
descendiente
        else {
            nodes.peek().appendChild(text);
        }
    }
    else {
        currentNode = doc.createElement(tag.getName());

        //Recorremos todos los atributos para añadirlos
                                  for    (Iterator    it    =
attributeTable.entrySet().iterator(); it.hasNext();) {
            Map.Entry entry = (Map.Entry) it.next();
            Obj key = (Obj) entry.getKey();
            String attrName = key.getName();
            String attrValue = null;
            try{
                                                  attrValue   =
attributeTable.get(key).getName();
            } catch(NullPointerException e) {}
                //El atributo todavía tiene las comillas,
debemos eliminarlos
                        attrValue  =  attrValue.substring(1,
attrValue.length() - 1);
            currentNode.setAttribute(attrName, attrValue);
        }

          //Si no hay nodo en la pila, añado el nodo como
raíz
        if(nodes.empty()) {
            try {
            doc.appendChild(currentNode);
            }catch(Exception e){}
        }
           //Si hay nodo en la pila, añado el nuevo como
descendiente
        else {
            nodes.peek().appendChild(currentNode);
        }
    }
  }
```

La función handleOpenTag además de crear el nodo, lo añadirá a la pila. Por su parte, handleCloseTag elimina de la pila el nodo de la cima en caso de no exista ningún tipo de error de anidamiento.
De esta manera iremos elaborando el árbol DOM a medida que analizamos el documento de entrada y detectamos los errores. Finalizado el análisis, repetiremos el proceso realizado anteriormente:

```
StringWriter sw = new StringWriter();
StreamResult result = new StreamResult(sw);
DOMSource source = new DOMSource(doc);
trans.transform(source, result);
String xmlString = sw.toString();
```

8. Interfaz web

8.1 Introducción

Tras haber finalizado el desarrollo de la herramienta de validación comprobamos que el funcionamiento es correcto pero no resulta del todo práctico. La herramienta requiere ser ejecutada de manera local. El usuario tendría que hacer uso de la consola para lanzarla y pasarle como parámetro el nombre del archivo donde se encuentra el código XHTML5 que desea validar. La tarea para el programador resulta más liviana, pero el resultado es una herramienta poco atractiva para el usuario final. Además, este deberá asegurarse de disponer de la máquina virtual de Java para poder hacer uso del validador.

Para mejorar la herramienta hemos decidido desarrollar una aplicación web. De esta manera, el usuario solo tendrá que acceder a la URL e introducir el código XHTML5 a validar. El único requisito será un navegador y una conexión a Internet. Como contrapartida, la tarea de desarrollo se complica ya que tenemos que tener en cuenta nuevos requerimientos como la presencia de un servidor de aplicaciones y nuevas tareas como el despliegue.

8.2 Diseño

El enfoque adoptado para nuestra aplicación en muy sencillo, Después de todo, la complejidad del desarrollo se centra principalmente en el proceso de validación de código que ya tenemos desarrollado. La idea es presentar una pantalla inicial con un formulario en el que el usuario pueda introducir el código que desea validar. Tras el envío del formulario, la aplicación retornará por pantalla el listado de los errores en formato XML y el código reparado.

Para acometer el diseño de la aplicación emplearemos el patrón de diseño MVC (Modelo-Vista-Controlador). Se trata de un patrón muy empleado en el desarrollo de interfaces para usuario. Su nombre se deriva de los tres elementos básicos que conforman su estructura:

- Modelo: implementa la lógica de funcionamiento o de negocio de la aplicación. Este parte del patrón ya la tendríamos prácticamente lista con todo el desarrollo previo del validador .
- Vista: muestra información sobre el estado de la aplicación.
- Controlador: recoge la información procedente de la vista y la traduce en operaciones sobre el modelo. Se encarga también del proceso inverso al recoger la información devuelta por el modelo y redirigirla a la vista para que la muestre por pantalla.

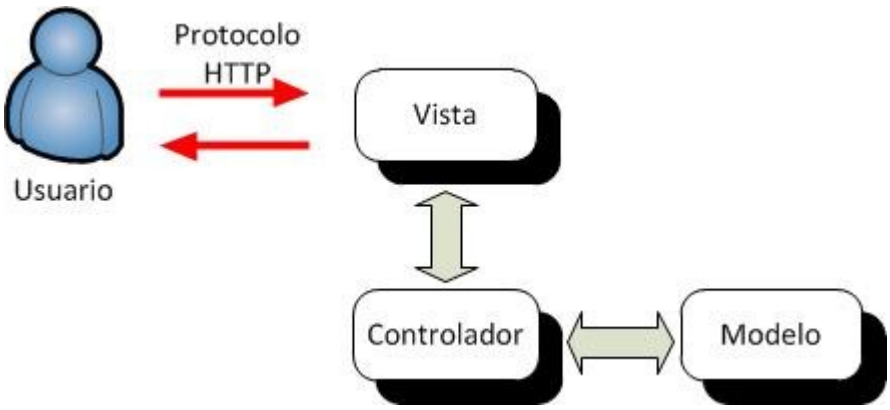


Illustration 28: Esquema del patrón Modelo-Vista-Controlador

Comprobamos por tanto una clara diferenciación entre los datos (representados por el modelo) y la presentación (representada por la vista). El controlador es el nexo de unión. La funcionalidad de este último la resolveremos con un *servlet*. Los *servlets* no son más que clases Java que extienden la funcionalidad de los servidores. Su cometido será recibir las peticiones HTTP y responder adecuadamente con llamadas al modelo o a la vista. En nuestro caso concreto implementaremos dicha funcionalidad en un alarde de originalidad en la clase validator.Controller. Esta clase, como todo *servlet*, hereda de la clase HttpServlet y presenta dos métodos principales: doGet y doPost. Estos métodos son los encargados de responder a las peticiones HTTP de tipo GET y de tipo POST, respectivamente. En el método doPost, recibiremos el código introducido por el usuario y lo pasaremos al validador, que nos devolverá el listado de errores y el código reparado. Finalmente, devolveremos esta información a la vista, que se encargará de mostrarla por pantalla. El método doGet no hará llamada alguna al modelo, simplemente pintará la pantalla inicial.

La funcionalidad de la vista la resolveremos con la tecnología JSP (*Java Server Page*). Este tipo de archivos presenta la estructura básica de un documento HTML, pero nos permiten además la inclusión de código Java mediante los denominados *scriptlets*, encerrados entre etiquetas < % %> . En nuestro caso solo necesitaremos una única vista, que reutilizaremos para recibir el código del usuario y para mostrarle los resultados. El aspecto de la vista será mejorado con la inclusión de una hoja de estilos.

El fichero CUP ha de ser modificado ligeramente para que la clase parser retorne los resultados convenientemente al controlador, ya que la impresión por salida estándar ha dejado de ser útil.

Finalmente, en el archivo de configuración web.xml definiremos cuál queremos que sea la vista inicial de la aplicación. Además, podremos informar del funcionamiento de nuestra clase validator.Controller como *servlet* y le asignaremos la URL a la que apunta el formulario de la vista.

8.3 Despliegue

Una vez resuelta la programación de la aplicación, queda pendiente el despliegue de esta. Por despliegue entendemos su instalación en el servidor de aplicaciones JBoss. En la práctica este proceso consiste en la copia de un archivo .war (*Web ARchive*). Este tipo de archivo no es más que un archivo comprimido con todos los componentes de la aplicación. No solo sus clases y JSPs, también hojas de estilo, imágenes, ficheros de configuración, etc. Ahora bien, ¿cómo se obtiene este archivo? Una de las alternativas de que disponemos es Apache Ant. El funcionamiento de Ant es bien simple. Sencillamente consulta el archivo de configuración build.xml y ejecuta secuencialmente cada una de las órdenes allí especificadas para obtener el archivo validator.war y copiarlo en la carpeta deploy del servidor de aplicaciones JBoss. Una vez en su ubicación el servidor lo detecta, despliega la aplicación e inmediatamente puede ser accedida vía web.

Existe un pequeño detalle que hemos obviado hasta el momento. Mencionábamos con anterioridad que la funcionalidad del modelo estaba resuelta con las clases Java que habíamos obtenido en pasos previos a partir de los ficheros JLex y CUP. Ahora bien, para obtener las correspondientes clases Java a partir de dichos archivos es necesario ejecutar JLex y CUP pasándoles como entrada los respectivos ficheros .jlex y .cup para obtener como salida las clases Java de los analizadores léxicos y sintácticos. Hasta ahora lo habíamos hecho de manera manual. A la hora de acometer la aplicación web, simplemente habíamos incluido las clases Final.java (lexer), parser.java y sym.java en el paquete validator. En caso de querer modificar su funcionamiento, deberíamos repetir el proceso nuevamente de manera manual. No obstante, la potencia de Ant nos permite automatizar también este proceso. Solo tendremos que incluir las librerías JLex y CUP en el proyecto y añadir unas líneas al archivo de configuración build.xml, para que antes de generar el archivo .jar y el .war, detecte si ha habido cambios en cualquier de los archivos .jlex o .cup y genere las nuevas clases.

```
  <target name="cup" depends="jlex">
    <cup srcfile="${source}/validator/final.cup"
         destdir="${source}"
         interface="true"
       parser="parser" />
  </target>

  <target name="jlex" depends="init">
             <jflex   file="${source}/validator/final.jlex"
destdir="${source}" />
  </target>
```

9. Plan de pruebas

El propósito de todo plan de pruebas es definir la estrategia a tomar con el fin de comprobar la funcionalidad de la herramienta desarrollada. A estas alturas partimos con cierta ventaja, ya que en el capítulo 2 de esta memoria ya hemos realizado un pequeño plan de pruebas para comparar el funcionamiento de los diferentes validadores. Podemos tomar este limitado plan de pruebas de base para llevar a cabo una labor más exhaustiva. En cualquier caso, no nos extenderemos demasiado, debido a las limitaciones de este documento. Conviene recordar que el contenido de los casos de uso originales se encuentran incluidos en la sección de anexos. Por comodidad incluiremos los productos de salida del validador (listado de errores y código corregido) en una carpeta incluida dentro del entregable.

El volumen de errores detectados tiende a ser grande, debido al afán por retornar un código XHTML5 reparado lo más fiel posible al original. Así pues, cuando encontramos una etiqueta inválida en un contexto determinado, el validador la desecha, por no su contenido. El proceso de validación continúa para su contenido, por lo que en caso de ser correcto, será incluido en el código reparado de salida.

Como decimos, emplearemos de base los casos de uso ya definidos, añadiremos nuevos. En todos ellos se incluye otros errores además del especificado, para comprobar que el validador no detiene su ejecución.

- html5-1: Documento XHTML5 sin declaración
- html5-2: Documento XHTML5 sin cabecera
- html5-3: Documento XHTML5 con etiqueta sin cerrar
- html5-4: Documento XHTML5 con etiquetas falsas
- html5-5: Documento XHTML5 con errores de anidamiento
- html5-6: Documento XHTML5 con varios cuerpos
- html5-7: Documento XHTML5 con etiquetas sin atributos obligatorios
- html5-8: Documento XHTML5 con etiquetas con atributos de estructura incorrecta
- html5-9: Documento XHTML5 con tabla de formato incorrecto (W3C emite advertencias)
- html5-10: Documento XHTML5 con entidades HTML incorrectas
- html5-11: Documento vacío
- html5-12: Documento XHTML5 correcto
- html5-13: Documento XHTML5 con varias cabeceras
- html5-14: Documento XHTML5 con varios cuerpos
- html5-15: Documento XHTML5 con la cabecera tras el cuerpo

10. Conclusiones

- Hemos podido ahondar en la evolución del lenguaje HTML, el proceso de validación y el funcionamiento de los navegadores más comunes hoy. Además, debido a lo extenso del trabajo a realizar, la gestión del tiempo ha sido una de las facetas más importantes durante su ejecución. Ha sido necesario un control constante del estado del proyecto y del tiempo disponible para su finalización satisfactoria..

- Los dos objetivos principales del proyecto han sido completados. Por un lado tenemos un estudio sobre los errores HTML. Por el otro, una herramienta de validación de código XHTML5 complementada con una interfaz web gráfica.

- Tras haber finalizado el proyecto podemos concluir que ha sido posible llevar a cabo la tarea sin desviaciones de la planificación de relevancia. En cualquier caso, se han llegado a ciertas conclusiones, de las que sería interesante dejar constancia:
    - La tarea “Estudio de herramientas” ha estado sobredimensionada. A causa de las indicaciones incluidas en la propuesta del PFC en las que se hacía especial hincapié en la importancia de la parte más teórica del proyecto, esta se ha llevado casi la mitad del tiempo disponible para el proyecto. Esto ha derivado en una necesidad de optimizar el uso del tiempo a la hora de desarrollar el validador en sí. Queremos pensar que la calidad de este no se ha visto comprometida, pero en cualquier caso, al tratarse de una desarrollo tan complejo, existen numerosas ideas y líneas de trabajo que se han quedado en el tintero por falta de tiempo. La ausencia de estas ampliaciones -que trataremos en el siguiente apartado - no desmerecen la funcionalidad obtenida, pero añadiría mucho más potencia a la herramienta.
    - La tarea “Aplicación web” ha sido subestimada y en previsión se ha restado tiempo a la tarea de pruebas del validador con el fin no desviar el resto de tareas posteriores. Por esto, la tarea de pruebas de la aplicación incluyó pruebas sobre el validador y acabó convertida en una tarea de pruebas global.
    - Ha sido necesario añadir una subtarea extra a la tarea de documentación para elaborar una pequeña guía de instalación de la aplicación. Esta guía será entregada junto a la aplicación.
    - A pesar de que las tareas de desarrollo de la aplicación y elaboración de la memoria tenían prevista su finalización una semana antes de la entrega del 10 de junio, se han alargado hasta esa misma fecha. De esta manera, estas dos tareas junto con la elaboración de la presentación se han realizado en paralelo durante la última semana.

- Futuras líneas de trabajo:
    - Si bien la lógica de la interfaz web es bien sencilla podría valorarse el empleo de algún framework en la capa de presentación como podría ser Struts.
    - La interfaz web podría ser mejorada ofreciéndole al usuario la posibilidad de introducir el código XHTML5 a través un archivo o especificando una URL. Para afrontar esta segunda ampliación podríamos hacer uso de HTTPClient. Esta herramienta simula las peticiones HTTP que realizaría un navegador común y recogería los resultados devueltos por lo servidores web remotos.
    - El validador desarrollado solo detecta errores de validación. Herramientas más avanzadas como la desarrollada por W3C generan también advertencias. Sería interesante incorporar dicha funcionalidad. Especialmente en el caso de estructuras más complejas como las que presentan las etiquetas table, audio y video, por ejemplo.
    - Ofrecer la posibilidad al usuario no solo de visualizar el XML de los errores y el documento XHTML5 corregido, sino de descargárselo con un botón; sin tener que recurrir a copiar y pegar el texto.

11. Glosario

Análisis léxico: proceso consistente en la conversión de una cadena de caracteres en una secuencia de *tokens* de acuerdo a una serie de reglas léxicas, comúnmente definidas mediante expresiones regulares. Desempeñado por un analizador léxico o *scanner*.

Análisis semántico: proceso posterior al análisis léxico y sintáctico en el que ya no se analiza la estructura sino el contenido semántico, esto es, el significado.

Análisis sintáctico: proceso de análisis de una secuencia de *tokens* de acuerdo una serie de reglas definidas por una gramática. Desempeñado por un analizador sintáctico o *parser*.

Apache Ant: herramienta orientada a la automatización de procesos de compilación y construcción de software. Emplea el formato XML para describir dichos procesos y sus dependencias. Por defecto el nombre del archivo es build.xml.

CUP: generador de analizadores sintácticos en Java, desarrollado en Java. Iniciado por Scott Hudson. En la actualidad es la Universidad Técnica de Múnich la que continúa con el proyecto.

DOM (***Document Object Model***): modelización de documentos (X)HTML y XML. Ofrece una API que permite la manipulación de sus componentes.

Expresión regular:combinación de caracteres y metacaracteres que define la estructura de cadenas de texto.

Gramática: conjunto de reglas que definen la estructura de un lenguaje formal de acuerdo a la sintaxis de dicho lenguaje.

HTML (***HyperText Markup Language***): lenguaje de marcado más empleado a la hora de elaborar páginas web.

Interfaz web: interfaz gráfica de usuario que permite la comunicación entre emisor y receptor mediante el uso de páginas web, las cuales emplean como canal de comunicación Internet. Un requisito indispensable para su uso es la existencia de un navegador web.

J2EE: plataforma de programación basada en Java. Proporciona una API y un entorno que sirve como base para desarrollar software de aplicaciones

JBOSS: servidor de aplicaciones de código abierto desarrollado en Java. Multiplataforma, precisamente por estar desarrollado en Java.

Implementa asimismo todo el paquete de servicios de J2EE. Desarrollado por la empresa Jboss Inc., controlada por Red Hat.

JLex: generador de analizadores léxicos en Java, desarrollado en Java. Iniciado por Elliot Berk en la Universidad de Priceton. Es mantenido actualmente por C. Scott Ananian.

Patrón de diseño: abstracción de una solución a problemas comunes dentro del área del diseño orientado a objetos en cualquiera de sus fases. Su efectividad ha sido comprobada previamente y son reutilizables. Existen patrones de diseño aplicables en múltiples situaciones, desde la fase de análisis hasta la de diseño, pudiendo afectar desde la arquitectura a la implementación.

JSP (***JavaServer Pages***): Tecnología orientada a la creación de páginas web de contenido dinámico empleando lenguaje Java. En el modelo MVC desempeña el rol de vista.

Motor de renderizado: software que toma como entrada contenido marcado (por ejemplo, HTML o imágenes) e información de formato (por ejemplo, CSS) y devuelve por pantalla dicho contenido formateado.

Servlet: clase Java que extiende la funcionalidad de un servidor web. En el modelo MVC despeña el rol de controlador.

SGML (***Standard Generalized Markup Language***): estándar definido para la organización y etiquetado de documentos. Base de los lenguajes HTML, XML y XHTML.

W3C (**World Wide Web Consortium**): organización internacional fundada por Tim Berners-Lee, encargada de velar por el desarrollo de estándares en la red.

Tabla hash: también mapa hash o tabla de dispersión. Estructura de datos formada por pares clave-valor. Reciben su nombre por el uso de funciones hash para transformar la clave en número que identifica la posición ocupada por el valor asociado a dicha clave.

Token: también testigo o lexema. Cadena de caracteres mínima con significado léxico. Unidad de trabajo del analizador sintáctico.

XHTML (***eXtensible Hypertext Markup Language***): lenguaje de estructura es muy similar a la del lenguaje HTML, pero además cumple las restricciones propias de todo documento XML.

XML (***eXtensible Markup Language***): lenguaje de marcado extensible. El propósito de este metalenguaje es la codificación de la información en un formato que sea fácilmente manejable tanto por personas como por máquinas.

12. Bibliografía

HTML y XHTML

http://en.wikipedia.org/wiki/HTML
http://en.wikipedia.org/wiki/HTML5
http://en.wikipedia.org/wiki/Document_markup_language
http://en.wikipedia.org/wiki/Standard_Generalized_Markup_Language
http://www.w3.org/MarkUp/SGML/
http://www.w3schools.com/html5/tag_doctype.asp
http://html.conclase.net/articulos/xml
http://www.htmlcinco.com/category/xhtml-5/
http://wiki.whatwg.org/wiki/HTML_vs._XHTML
http://www.csulb.edu/divisions/students/dss/accessibility/web/webaim-12comm.html
http://line25.com/articles/10-common-validation-errors-and-how-to-fix-them

Fuentes consultadas de manera periódica entre 21-03-2012 y 10-04-2012.

Herramientas de validación

http://validator.w3.org/
http://validator.nu/
http://html5.validator.nu/
http://www.htmlhelp.com/tools/validator/
http://tidy.sourceforge.net/
http://www.totalvalidator.com/
http://relaxed.vse.cz/relaxed/validate?group=Web
http://www.validome.org/

Fuentes consultadas de manera periódica entre 21-03-2012 y 25-03-2012.

Renderizado

http://css-tricks.com/
http://news.cnet.com/8301-30685_3-57407891-264/microsofts-ie-reclaims-lost-ground-in-browser-battle/
http://www.html5rocks.com/en/tutorials/internals/howbrowserswork/
http://taligarsiel.com/Projects/howbrowserswork1.htm
http://www-archive.mozilla.org/newlayout/doc/gecko-overview.htm

Fuentes consultadas de manera entre 06-03-2012 y 08-03-2012.

Validador HTML

http://download.oracle.com/javase/6/docs/api/
http://stackoverflow.com/questions/tagged/java
http://javacook.darwinsys.com/download.html

Fuentes consultadas desde 12-04-2012 hasta el final del proyecto (10-06-2012).

Modern Compiler Implementation in Java, 2nd ed.(Andrew Appel_ Cambridge University Press)(2004)
Temario de la asignatura Compiladores I (UOC)
Temario de la asignatura Compiladores II (UOC)
Temario de la asignatura Ingeniería de la programación orientada a objetos (UOC)

Interfaz web

https://community.jboss.org/en/jbossas/config
http://stackoverflow.com/questions/tagged/jboss5.x
http://java.sun.com/j2se/1.3/docs/guide/jar/jar.html
http://www.programacion.com/java/tutorial/jar
http://www.surajitray.com/coders-corner/jboss-5-1-on-ubuntu-10-04/

Fuentes consultadas desde 20-05-2012 hasta el final del proyecto (10-06-2012).

Temario de la asignatura Ingeniería de la programación de componentes y sistemas distribuidos (UOC)

Gestión del proyecto

Temario de la asignatura Metodología y gestión de proyectos informáticos (UOC)

13. Anexos

A Etiquetas en HTML5

Etiqueta	Descripción	Atributos
<!-...-->	Comentario de código	Ninguno
<!DOCTYPE>	Tipo de documento	Ninguno
<a>	Hipervínculo	Generales href hreflang media rel target type
<abbr>	Abreviatura	Generales
<address>	Información de contacto	Generales
<area>	Área dentro de un mapa de imagen	Generales alt coords href hreflang media rel target type
<article>	Artículo	Generales
<aside>	Contenido extra	Generales
<audio>	Contenido de sonido	Generales autoplay controls loop preload src
<b>	Texto en negrita	Generales
<base>	URL base para el resto de URL relativas del documento	Generales href target
<bdi>	Texto formateado en otra dirección	Generales
<bdo>	Ignora la orientación actual del texto	Generales dir
<blockquote>	Sección de otra fuente	Generales cite

<body>	Cuerpo del documento	Generales
 	Salto de línea	Generales
<button>	Botón	Generales autofocus disabled form formaction formenctype formmethod formnovalid ate formatarget name type value
<canvas>	Lienzo para gráficos	Generales height width
<caption>	Título de tabla	Generales
<cite>	Cita	Generales
<code>	Código	Generales
<col>	Fija las propiedad de una columna dentro de un colgroup	Generales span
<colgroup>	Especifica un grupo de una o más columnas para su maquetación	Generales span
<command>	Botón invocable por un usuario	Generales checked disabled icon label radiogroup type
<datalist>	Lista de opciones predefinidas para inputs	Generales
<dd>	Descripción de un elemento en una lista de definición	Generales
<del>	Texto que ha sido borrado de un documento	Generales cite datetime
<details>	Detalles adicionales	Generales open
<dfn>	Definición	Generales
	Sección de un documento	Generales
<dl>	Lista de definición	Generales

<dt>	Elemento de una lista de definición	Generales
	Texto con énfasis	Generales
<embed>	Contenedor para una aplicación o un plug-in	Generales height src type width
<fieldset>	Elementos relacionados dentro de un formulario	Generales disabled form name
<figcaption>	Título de un elemento <figure>	Generales
<figure>	Contenido independiente	Generales
<footer>	Pie del documento	Generales
<form>	Formulario	Generales accept-charset action autocomplete enctype method name novalidate target
<h1>	Título de primer nivel	Generales
<h2>	Título de segundo nivel	Generales
<h3>	Título de tercer nivel	Generales
<h4>	Título de cuarto nivel	Generales
<h5>	Título de quinto nivel	Generales
<h6>	Título de sexto nivel	Generales
<head>	Información del documento	Generales
<header>	Cabecera del documento	Generales
<hgroup>	Agrupación de títulos	Generales
<hr>	Cambio temático	Generales
<html>	Raíz del documento	Generales manifest
<i>	Porción en otro tono	Generales
<iframe>	Marco en línea	Generales height name sandbox

		seamless src srcdoc width
<img>	Imagen	Generales src alt height ismap usemap width
<input>	Elemento de entrada	Generales accept alt autocomplete autofocus checked disabled form formaction formenctype formmethod formnovalidate formtarget height list max maxlength min multiple name pattern placeholder readonly required size src step type value width
<ins>	Texto insertado en un documento	Generales cite datetime
<keygen>	Generador de claves para formulario	Generales autofocus

		challenge disabled form keytype name
<kbd>	Entrada de teclado	Generales
<label>	Etiqueta para elemento input	Generales for form
<legend>	Título para un elemento fieldset, figure o details	Generales
	Elemento de una lista	Generales value
<link>	Relación entre el documento y un elemento externo	Generales href hreflang media rel sizes type
<map>	Mapa de imágenes	Generales name
<mark>	Texto resaltado	Generales
<menu>	Menú	Generales label type
<meta>	Metadata del documento	Generales charset content http-equiv name
<meter>	Indicador	Generales form high low max min optimum value
<nav>	Enlaces de navegación	Generales
<noscript>	Contenido alternativo para usuarios que no soporten scripts en el cliente	Generales
<object>	Objeto embebido	Generales data form

		heigth name type usemap width
	Lista ordenada	Generales reversed start type
<optgroup>	Opciones relacionadas en una lista desplegable	Generales label disabled
<option>	Opción en una lista desplegable	Generales disabled label selected value
<output>	Resultado de un cálculo	Generales for form name
<p>	Párrafo	Generales
<param>	Parámetro para un elemento object	Generales name value
<pre>	Texto preformateado	Generales
<progress>	Progreso de una tarea determinada	Generales max value
<q>	Corta cita	Generales cite
<rp>	Contenido alternativo para navegadores que no soportan anotaciones en caracteres ruby	Generales
<rt>	Explicación y/o pronunciación de caracteres de lenguas ideogramáticas	Generales
<ruby>	Anotación en caracteres ruby para lenguas ideogramáticas	Generales
<s>	Texto que ya no es correcto	Generales
<samp>	Texto de salida de un programa	Generales
<script>	Script	Generales async defer type charset

		src
<section>	Sección del documento	Generales
<select>	Lista del desplegable	Generales autofocus disabled form multiple name size
<small>	Texto pequeño	Generales
<source>	Fuentes para elementos video y audio	Generales media src type
	Sección del documento	Generales
	Texto importante	Generales
<style>	Información de estilo del documento	Generales type media scoped
<sub>	Subíndice	Generales
<summary>	Título visible para un elemento details	Generales
<sup>	Superíndice	Generales
<table>	Tabla	Generales border
<tbody>	Cuerpo de la tabla	Generales
<td>	Celda de la tabla	Generales colspan headers rowspan
<textarea>	Elemento de entrada	Generales autofocus cols disabled form maxlength name placeholder readonly required rows wrap
<tfoot>	Pie de una tabla	Generales

<th>	Cabecera de la tabla	Generales colspan headers rowspan scope
<thead>	Agrupación de cabeceras de la tabla	Generales
<time>	Fecha u hora	Generales datetime pubdate
<title>	Título del documento	Generales
<tr>	Fila de la tabla	Generales
<track>	Pistas de texto para elementos audio y vídeo	Generales default kind label src srclang
<u>	Texto con diferente estilo	Generales
	Lista no ordernada	Generales
<var>	Variable	Generales
<video>	Vídeo o película	Generales autoplay controls height loop muted poster preload src width
<wbr>	Salto de línea	Generales

B Atributos generales en HTML5

Nombre	**Descripción**
`accesskey`	Acceso directo al elemento
`class`	Nombre de clase (útil para aplicar estilos)
`contenteditable`	Elemento editable o no
`contextmenu`	Menú de contexto al que pertenece
`dir`	Dirección del texto del documento
`draggable`	Elemento arrastable o no
`dropzone`	Define si el texto es copiado, movido o enlazado cuando se arrastra a este elemento
`hidden`	Visible o no
`id`	Identificador único
`lang`	Idioma del contenido del elemento
`spellcheck`	Define si el contenido del elemento ha de ser revisado ortográficamente o no.
`style`	Estilo CSS en línea
`tabindex`	Orden de tabulador
`title`	Información adicional

C Casos de uso

Caso de uso HTML5-1

```
<html>
      <head>
            <title>Title of the document</title>
      </head>
      <body>
            The content of the document......
            <h1 id="1d">Heading 1</h1>
            <div id="1d" class="1m">
                  <span>
                        Content
                  </div>
            </span>
      </body>
</html>
```

Caso de uso HTML5-2

```
<!DOCTYPE html>
<html>
      <body>
            The content of the document......
            <h1 id="1d">Heading 1</h1>
            <div id="1d" class="1m">
                  <span>
                        Content
                  </div>
            </span>
      </body>
</html>
```

Caso de uso HTML5-3

```
<!DOCTYPE html>
<html>
      <head>
            <title>Title of the document</title>
      </head>
      <body>
            <img           src="public/images/menu/fondo_menu.png"
alt="Texto">
            <a         href="http://www.youtube.com/?tab=w1&gl=ES"
>Link</a>
            The content of the document......
            <h1 id="1d">Heading 1</h1>
            <div id="1d" class="1m">
                  <span>
```

```
                    Content
                </div>
            </span>
      </body>
</html>
```

Caso de uso HTML5-4

```
<!DOCTYPE html>
<html>
      <head>
            <title>Title of the document</title>
      </head>
      <body clase="wrong">
            <dib>wrong
            <img           src="public/images/menu/fondo_menu.png"
alt="Texto">
            <a         href="http://www.youtube.com/?tab=w1&gl=ES"
>Link</a>
            The content of the document......
            <h1 id="1d">Heading 1</h1>
            <div id="1d" class="1m">
                  <span>
                        Content
                  </div>
            </span>
            </dib>
      </body>
</html>
```

Caso de uso HTML5-5

```
<!DOCTYPE html>
<html>
      <head>
            <title>Title of the document</title>
      </head>
      <body>
            <div><span>wrong</div></span>
            <img           src="public/images/menu/fondo_menu.png"
alt="Texto">
            <a         href="http://www.youtube.com/?tab=w1&gl=ES"
>Link</a>
            The content of the document......
            <h1 id="1d">Heading 1</h1>
            <div id="1d" class="1m">
                  <span>
                        Content
                  </div>
            </span>
      </body>
</html>
```

Caso de uso HTML5-6

```
<!DOCTYPE html>
<html>
      <head>
            <title>Title of the document</title>
      </head>
      <body>
            <div><span>wrong<span><div>
            <img           src="public/images/menu/fondo_menu.png"
alt="Texto">
            <a         href="http://www.youtube.com/?tab=w1&gl=ES"
>Link</a>
            The content of the document......
            <h1 id="1d">Heading 1</h1>
            <div id="1d" class="1m">
                  <span>
                        Content
                  </div>
            </span>
      </body>
</html>
```

Caso de uso HTML5-7

```
<!DOCTYPE html>
<html>
      <head>
            <title>Title of the document</title>
      </head>
      <body>
            <div><span>wrong</span></div>
            <img src="public/images/menu/fondo_menu.png">
            <a>Link</a>
            The content of the document......
            <h1 id="1d">Heading 1</h1>
            <div id="1d" class="1m">
                  <span>
                        Content
                  </div>
            </span>
      </body>
</html>
```

Caso de uso HTML5-8

```
<!DOCTYPE html>
<html>
      <head>
```

```
            <title>Title of the document</title>
      </head>
      <body>
            <select>
              <option>1</option>
              <option selected>2</option>
              <option>3</option>
            </select>
            <div><span>wrong</span></div>
            <img           src="public/images/menu/fondo_menu.png"
alt="Texto">
            <a         href="http://www.youtube.com/?tab=w1&gl=ES"
>Link</a>
            The content of the document......
            <h1 id="1d">Heading 1</h1>
            <div id="1d" class="1m">
                  <span>
                        Content
                  </div>
            </span>
      </body>
</html>
```

Caso de uso HTML5-9

```
<!DOCTYPE html>
<html>
      <head>
            <title>Title of the document</title>
      </head>
      <body>
            <table>
                  <thead>
                        <tr>
                              <th scope="col">Enemies List</th>
                        </tr>
                  </thead>
                  <tbody>
                        <tr>
                              <td>Mr. Incredible</td>
                              <td>Elastigirl</td>
                              <td>Gazer Beam</td>
                        </tr>
                  </tbody>
                  <tfoot>
                        <tr>
                              <td>&copy; Bomb Voyage</td>
                        </tr>
                  </tfoot>
            </table>
            <div>
                  <span>wrong</span>
            </div>
```

```
          <img          src="public/images/menu/fondo_menu.png"
alt="Texto">
          <a         href="http://www.youtube.com/?tab=w1&gl=ES"
>Link</a>
          The content of the document...... <h1 id="1d">Heading
1</h1>
          <div id="1d" class="1m">
               <span> Content
          </div>
          </span>
     </body>
</html>
```

Caso de uso HTML5-10

```
<!DOCTYPE html>
<html>
     <head>
          <title>Title of the document</title>
     </head>
     <body>
          <div id="1id" class="1class">
               <span>wrong<span>
          <div>
          <img          src="public/images/menu/fondo_menu.png"
alt="Texto">
          <a         href="http://www.youtube.com/?tab=w1&gl=ES"
>Link</a>
          The content of the document......
          <h1 id="1d">Heading 1</h1>
          <div id="1d" class="1m">
               <span>
                    Content
               </div>
          </span>
     </body>
</html>
```

Caso de uso HTML4.01-1

```
<html>
     <head>
          <title>Title of the document</title>
     </head>
     <body>
          The content of the document......
          <h1 id="1d">Heading 1</h1>
          <div id="1d" class="1m">
               <span>
                    Content
               </div>
          </span>
     </body>
</html>
```

Caso de uso HTML4.01-2

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http://www.w3.org/TR/html4/strict.dtd">
<html>
      <body>
            The content of the document......
            <h1 id="1d">Heading 1</h1>
            <div id="1d" class="1m">
                  <span>
                        Content
                  </div>
            </span>
      </body>
</html>
```

Caso de uso HTML4.01-3

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http://www.w3.org/TR/html4/strict.dtd">
<html>
      <head>
            <title>Title of the document</title>
      </head>
      <body>
            <img           src="public/images/menu/fondo_menu.png"
alt="Texto">
            <a         href="http://www.youtube.com/?tab=w1&gl=ES"
>Link</a>
            The content of the document......
            <h1 id="1d">Heading 1</h1>
            <div id="1d" class="1m">
                  <span>
                        Content
                  </div>
            </span>
      </body>
</html>
```

Caso de uso HTML4.01-4

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http://www.w3.org/TR/html4/strict.dtd">
<html>
      <head>
            <title>Title of the document</title>
      </head>
      <body clase="wrong">
            <dib>wrong
```

```
          <img          src="public/images/menu/fondo_menu.png"
alt="Texto">
          <a         href="http://www.youtube.com/?tab=w1&gl=ES"
>Link</a>
          The content of the document......
          <h1 id="1d">Heading 1</h1>
          <div id="1d" class="1m">
               <span>
                    Content
               </div>
          </span>
          </dib>
     </body>
</html>
```

Caso de uso HTML4.01-5

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http://www.w3.org/TR/html4/strict.dtd">
<html>
     <head>
          <title>Title of the document</title>
     </head>
     <body>
          <div><span>wrong</div></span>
          <img          src="public/images/menu/fondo_menu.png"
alt="Texto">
          <a         href="http://www.youtube.com/?tab=w1&gl=ES"
>Link</a>
          The content of the document......
          <h1 id="1d">Heading 1</h1>
          <div id="1d" class="1m">
               <span>
                    Content
               </div>
          </span>
     </body>
</html>
```

Caso de uso HTML4.01-6

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http://www.w3.org/TR/html4/strict.dtd">
<html>
     <head>
          <title>Title of the document</title>
     </head>
     <body>
          <div><span>wrong<span><div>
          <img          src="public/images/menu/fondo_menu.png"
alt="Texto">
```

```
            <a         href="http://www.youtube.com/?tab=w1&gl=ES"
>Link</a>
            The content of the document......
            <h1 id="1d">Heading 1</h1>
            <div id="1d" class="1m">
                  <span>
                        Content
                  </div>
            </span>
      </body>
</html>
```

Caso de uso HTML4.01-7

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http://www.w3.org/TR/html4/strict.dtd">
<html>
      <head>
            <title>Title of the document</title>
      </head>
      <body>
            <div><span>wrong</span></div>
            <img src="public/images/menu/fondo_menu.png">
            <a>Link</a>
            The content of the document......
            <h1 id="1d">Heading 1</h1>
            <div id="1d" class="1m">
                  <span>
                        Content
                  </div>
            </span>
      </body>
</html>
```

Caso de uso HTML4.01-8

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http://www.w3.org/TR/html4/strict.dtd">
<html>
      <head>
            <title>Title of the document</title>
      </head>
      <body>
            <select>
              <option>1</option>
              <option selected>2</option>
              <option>3</option>
            </select>
            <div><span>wrong</span></div>
            <img           src="public/images/menu/fondo_menu.png"
alt="Texto">
```

```
          <a         href="http://www.youtube.com/?tab=w1&gl=ES"
>Link</a>
          The content of the document......
          <h1 id="1d">Heading 1</h1>
          <div id="1d" class="1m">
               <span>
                    Content
               </div>
          </span>
     </body>
</html>
```

Caso de uso HTML4.01-9

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http://www.w3.org/TR/html4/strict.dtd">
<html>
     <head>
          <title>Title of the document</title>
     </head>
     <body>
          <table>
               <thead>
                    <tr>
                         <th scope="col">Enemies List</th>
                    </tr>
               </thead>
               <tbody>
                    <tr>
                         <td>Mr. Incredible</td>
                         <td>Elastigirl</td>
                         <td>Gazer Beam</td>
                    </tr>
               </tbody>
               <tfoot>
                    <tr>
                         <td>&copy; Bomb Voyage</td>
                    </tr>
               </tfoot>
          </table>
          <div>
               <span>wrong</span>
          </div>
          <img           src="public/images/menu/fondo_menu.png"
alt="Texto">
          <a         href="http://www.youtube.com/?tab=w1&gl=ES"
>Link</a>
          The content of the document...... <h1 id="1d">Heading
1</h1>
          <div id="1d" class="1m">
               <span> Content
          </div>
          </span>
```

```
      </body>
</html>
```

Caso de uso HTML4.01-10

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http://www.w3.org/TR/html4/strict.dtd">
<html>
      <head>
            <title>Title of the document</title>
      </head>
      <body>
            <div id="1id" class="1class">
                  <span>wrong<span>
            <div>
            <img           src="public/images/menu/fondo_menu.png"
alt="Texto">
            <a         href="http://www.youtube.com/?tab=w1&gl=ES"
>Link</a>
            The content of the document......
            <h1 id="1d">Heading 1</h1>
            <div id="1d" class="1m">
                  <span>
                        Content
                  </div>
            </span>
      </body>
</html>
```

Caso de uso XHTML1.0-1

```
<html>
      <head>
            <title>Title of the document</title>
      </head>
      <body>
            The content of the document......
            <h1 id="1d">Heading 1</h1>
            <div id="1d" class="1m">
                  <span>
                        Content
                  </div>
            </span>
      </body>
</html>
```

Caso de uso XHTML1.0-2

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
```

```
"DTD/xhtml1-strict.dtd">
<html>
      <body>
            The content of the document......
            <h1 id="1d">Heading 1</h1>
            <div id="1d" class="1m">
                  <span>
                        Content
                  </div>
            </span>
      </body>
</html>
```

Caso de uso XHTML1.0-3

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"DTD/xhtml1-strict.dtd">
<html>
      <head>
            <title>Title of the document</title>
      </head>
      <body>
            <img           src="public/images/menu/fondo_menu.png"
alt="Texto">
            <a         href="http://www.youtube.com/?tab=w1&gl=ES"
>Link</a>
            The content of the document......
            <h1 id="1d">Heading 1</h1>
            <div id="1d" class="1m">
                  <span>
                        Content
                  </div>
            </span>
      </body>
</html>
```

Caso de uso XHTML1.0-4

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"DTD/xhtml1-strict.dtd">
<html>
      <head>
            <title>Title of the document</title>
      </head>
      <body clase="wrong">
            <dib>wrong
            <img           src="public/images/menu/fondo_menu.png"
alt="Texto">
```

```
          <a         href="http://www.youtube.com/?tab=w1&gl=ES"
>Link</a>
          The content of the document......
          <h1 id="1d">Heading 1</h1>
          <div id="1d" class="1m">
               <span>
                    Content
               </div>
          </span>
          </dib>
     </body>
</html>
```

Caso de uso XHTML1.0-5

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"DTD/xhtml1-strict.dtd">
<html>
     <head>
          <title>Title of the document</title>
     </head>
     <body>
          <div><span>wrong</div></span>
          <img           src="public/images/menu/fondo_menu.png"
alt="Texto">
          <a         href="http://www.youtube.com/?tab=w1&gl=ES"
>Link</a>
          The content of the document......
          <h1 id="1d">Heading 1</h1>
          <div id="1d" class="1m">
               <span>
                    Content
               </div>
          </span>
     </body>
</html>
```

Caso de uso XHTML1.0-6

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"DTD/xhtml1-strict.dtd">
<html>
     <head>
          <title>Title of the document</title>
     </head>
     <body>
          <div><span>wrong<span><div>
```

```
          <img          src="public/images/menu/fondo_menu.png"
alt="Texto">
          <a         href="http://www.youtube.com/?tab=w1&gl=ES"
>Link</a>
          The content of the document......
          <h1 id="1d">Heading 1</h1>
          <div id="1d" class="1m">
               <span>
                    Content
               </div>
          </span>
     </body>
</html>
```

Caso de uso XHTML1.0-7

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"DTD/xhtml1-strict.dtd">
<html>
     <head>
          <title>Title of the document</title>
     </head>
     <body>
          <div><span>wrong</span></div>
          <img src="public/images/menu/fondo_menu.png">
          <a>Link</a>
          The content of the document......
          <h1 id="1d">Heading 1</h1>
          <div id="1d" class="1m">
               <span>
                    Content
               </div>
          </span>
     </body>
</html>
```

Caso de uso XHTML1.0-8

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"DTD/xhtml1-strict.dtd">
<html>
     <head>
          <title>Title of the document</title>
     </head>
     <body>
          <select>
            <option>1</option>
            <option selected>2</option>
```

```
            <option>3</option>
          </select>
          <div><span>wrong</span></div>
          <img          src="public/images/menu/fondo_menu.png"
alt="Texto">
          <a         href="http://www.youtube.com/?tab=w1&gl=ES"
>Link</a>
          The content of the document......
          <h1 id="1d">Heading 1</h1>
          <div id="1d" class="1m">
               <span>
                    Content
               </div>
          </span>
     </body>
</html>
```

Caso de uso XHTML1.0-9

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"DTD/xhtml1-strict.dtd">
<html>
     <head>
          <title>Title of the document</title>
     </head>
     <body>
          <table>
               <thead>
                    <tr>
                         <th scope="col">Enemies List</th>
                    </tr>
               </thead>
               <tbody>
                    <tr>
                         <td>Mr. Incredible</td>
                         <td>Elastigirl</td>
                         <td>Gazer Beam</td>
                    </tr>
               </tbody>
               <tfoot>
                    <tr>
                         <td>&copy; Bomb Voyage</td>
                    </tr>
               </tfoot>
          </table>
          <div>
               <span>wrong</span>
          </div>
          <img          src="public/images/menu/fondo_menu.png"
alt="Texto">
```

```
          <a         href="http://www.youtube.com/?tab=w1&gl=ES"
>Link</a>
          The content of the document...... <h1 id="1d">Heading
1</h1>
          <div id="1d" class="1m">
               <span> Content
          </div>
          </span>
     </body>
</html>
```

Caso de uso XHTML1.0-10

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"DTD/xhtml1-strict.dtd">
<html>
     <head>
          <title>Title of the document</title>
     </head>
     <body>
          <div id="1id" class="1class">
               <span>wrong<span>
          <div>
          <img           src="public/images/menu/fondo_menu.png"
alt="Texto">
          <a         href="http://www.youtube.com/?tab=w1&gl=ES"
>Link</a>
          The content of the document......
          <h1 id="1d">Heading 1</h1>
          <div id="1d" class="1m">
               <span>
                    Content
               </div>
          </span>
     </body>
</html>
```