Capítulo 2

Tecnologías Básicas Empleadas en el Desarrollo del Proyecto

En el presente capítulo se hará un breve recorrido por aquellas tecnologías que han servido de instrumento para los trabajos desarrollados en el proyecto. Así, se examinan brevemente las distintas tecnologías argumentando y describiendo la utilización de cada una de ellas en el ámbito del presente proyecto.

Este recorrido por las distintas herramientas tratará los siguientes puntos:

- UML
- HTML
- XML
- HTML-XML
- JavaScript

2.1 UML

El modelado es una parte central de todas las actividades que conducen a la producción de un buen software. Construímos modelos para comprender mejor el sistema

que estamos construyendo, para visualizar y controlar la arquitectura del sistema y por último para comunicar la estructura deseada y el comportamiento de nuestro sistema.

El modelado proporciona una comprensión del sistema. Nunca es suficiente un único modelo. Más bien, para comprender cualquier cosa, a menudo se necesitan múltiples modelos conectados entre sí. El vocabulario y las reglas de un lenguaje como UML indican como crear y leer modelos bien formados, pero no dicen qué modelos se deben crear ni cuando se deberían crear. Esta es la tarea del proceso de desarrollo de software.

Para muchos programadores la distancia entre pensar en una implementación y transformarla en código es prácticamente nula. Sin embargo, esto plantea algunos problemas que se tratan de solucionar con UML:

- En primer lugar, la comunicación de esos modelos conceptuales a otros está sujeta a errores a menos que cualquier persona implicada hable el mismo lenguaje.
- En segundo lugar, hay algunas cuestiones sobre un sistema software que no se pueden entender a menos que se construyan modelos que trasciendan el lenguaje de programación textual.
- Por último, si el desarrollador que escribió el código no dejó documentación escrita sobre los modelos que tenía en mente, esa información se perderá para siempre, o , a lo sumo, será sólo parcialmente reproducible a partir de la implementación.

2.1.1 Bloques constructivos de UML

El vocabulario de UML incluye tres clases de bloques de construcción :

1. Elementos
2. Diagramas
3. Relaciones

Los elementos son abstracciones que son ciudadanos de primera clase en un modelo; las relaciones ligan estos elementos entre sí; los diagramas agrupan colecciones interesantes de elementos.

1. Elementos de UML

Hay cuatro tipos de elementos en UML:

1. Elementos estructurales

   Los elementos estructurales son los nombres de los modelos UML. En su mayoría son las partes estáticas de un modelo, y representan cosas que son conceptuales o materiales.

2. Elementos de comportamiento.

   Los elementos de comportamiento son las partes dinámicas de los modelos UML. Estos son los verbos de un modelo, y representan comportamiento en el tiempo y el espacio.

3. Elementos de agrupación.

   Los elementos de agrupación son las partes organizativas de los modelos UML.

4. Elementos de anotación.

   Una nota es un símbolo para mostrar restricciones y comentarios junto a un elemento o una colección de elementos.

Estos elementos son los bloques básicos de construcción orientados a objetos de UML.

2. Diagramas.

Un diagrama es la representación gráfica de un conjunto de elementos, visualizado la mayoría de las veces como un grafo conexo de nodos (elementos) y arcos (relaciones). Los diagramas se dibujan para visualizar un sistema desde diferentes perspectivas, de forma que un diagrama es una proyección de un sistema.

UML incluye nueve diagramas:

1. Diagrama de clases
2. Diagrama de objetos
3. Diagramas de casos de uso
4. Diagrama de secuencia
5. Diagrama de colaboración
6. Diagrama de estados
7. Diagrama de actividades

8. Diagrama de componentes
9. Diagrama de despliegue

3. Relaciones

UML proporciona cuatro tipos de relaciones que suponen los bloques básicos de construcción para relaciones de UML.

1. Dependencia.

La dependencia es una relación semántica entre dos elementos, en la cual un cambio a un elemento (el elemento independiente) puede afectar a la semántica de otro elemento (el elemento dependiente).

2. Asociación.

Es una relación estructural que describe un conjunto de enlaces, los cuales son conexiones entre objetos.

3. Generalización.

Es una relación de especialización/generalización en la cual los objetos del elemento especializado (el hijo) pueden sustituir a los elementos del objeto general (padre).

4. Realización.

Es una relación semántica entre clasificadores, en donde un clasificador especifica un contrato que otro clasificador garantiza que cumplirá.

A lo largo del desarrollo de la presente documentación, y conforme a las necesidades que se han ido detectando, se han ilustrado los diferentes puntos tratados a través de la utilización de alguno de los diagramas que UML proporciona.

2.2 HTML

El HTML, cuyas siglas pertenecen a HyperText Markup Language (Lenguaje de Anotación en Hipertexto), es un sencillo lenguaje basado en etiquetas, utilizado en todas las páginas web, independientemente de otros lenguajes añadidos que empleen.

Para crear una página en HTML, no es necesario emplear ningún tipo de programa, sino que tan sólo con cualquier editor de texto es válido para escribir cualquier página, dándole la extensión '.htm' o 'html'. En la realización del presente proyecto se ha utilizado la herramienta Microsoft Development Environment 6.0, perteneciente al Visual Studio 6.0, para el desarrollo de todos las lenguajes utilizados, es decir, HTML, XML y JavaScript.

El funcionamiento del lenguaje, se basa en las etiquetas o tags, siempre encerradas entre corchetes (los símbolos mayor y menor), y habitualmente con una de inicio y otra de fin: <ETIQUETA>...</ETIQUETA>

Un documento se divide en dos zonas: el encabezamiento y el cuerpo, y todo el documento está englobado por la etiqueta HTML, que indica el inicio y el fin, del siguiente modo:

<HTML>

<HEAD>...</HEAD>

<BODY>...</BODY>

</HTML>

Dentro del encabezamiento se incluye el título de la página (<TITLE>...</TITLE>) y otra información sobre el contenido, autor, palabras clave, etc.

En la parte correspondiente al cuerpo, se incluye toda la descripción de la página y es donde se desarrolla el lenguaje.

El texto va a venir organizado en párrafos con la etiqueta <P> y </P> para cerrar, y entre ellas se podrá incluir la etiqueta
 que produce un salto de línea sin espacio. Entre las etiquetas del párrafo se escribirá el texto que se desee visualizar.

Para incluir negrita, cursiva o un subrayado se emplean las etiquetas <B> y </B>, <I> y </I> y <U> y </U> respectivamente, al principio y final de cada bloque a seleccionar.

Para la inclusión de gráficos, ya sea del tipo GIF o JPEG indistintamente, se emplea la etiqueta <IMG SRC="grafico.gif"> representando en pantalla el gráfico llamado ‘grafico.gif’’.

Por último, uno de los fundamentos, sin los que no tendría valor Internet, es la posibilidad de enlazar con otras páginas, para lo que se utiliza la etiqueta <A

HREF>, </A>, de la siguiente manera: <A HREF="pagina.html"> Ir a página </A> para acceder a una página dentro de la misma estructura que la actual o <A HREF="http://dominio.com"> Ir a www.dominio.com </A> para acceder a una página externa.

HTML permite el uso de multitud de etiquetas y atributos con diversas posibilidades, que aparecen detalladas en manuales específicos o cursos. Hay que tener en cuenta, que algunos comandos sólo son propios de un navegador, sin producir ningún efecto sobre otros, como ocurre con ciertas funciones que soporta MS Explorer y no Netscape Navigator y viceversa, para lo cual habrá que tener en cuenta si se van a emplear.

Entre estas etiquetas que permiten diversas posibilidades, destacan las de tipo INPUT, tanto del tipo button, text o list, que proporcionan un alto grado de interactividad con el usuario a través de scripts en JavaScript.

Para publicar esta información en Internet tan sólo es necesario enviar los archivos con un programa FTP a la ubicación del servidor deseada. Es importante saber, que para que se ejecute la página principal, debe tener el nombre, habitualmente, 'index.htm' o 'index.html' de modo que al solicitar la dirección 'www.dominio.com/miweb' aparecerá el archivo ubicado en 'www.dominio.com/miweb/index.html'.

2.3 XML

En las siguientes líneas se hará una breve introducción a XML y se analizará la forma en que ha sido utilizado en el presente proyecto.

2.3.1 Orígenes de XML

XML, que significa Extensible Markup Language (lenguaje de marcado extensible), fue definido por el Grupo de Trabajo XML del Consorcio World Wide Web (W3C, World Wide Web Consortium). Este grupo describió el lenguaje de la siguiente manera:

“El lenguaje de marcado extensible (XML) es un subconjunto de SGML… Su objetivo consiste en posibilitar que el SGML genérico pueda suministrarse, recibirse y procesarse en la Web, de la misma manera que hoy en día es posible

con HTML. XML ha sido diseñado para que su implementación sea sencilla y permita interoperar tanto con SGML como con HTML".

Ésta es una cita de la versión 1.0 de la especificación oficial de XML, que fue terminada en febrero de 1998 por el Grupo de Trabajo XML.

XML está basado en el anterior estándar SGML (Standard Generalized Markup Language, ISO 8879), que data de 1986, pero que empezó a gestarse desde principios de los años 70, y a su vez basado en GML creado por IBM en 1969. Esto significa que aunque XML puede parecer moderno, sus conceptos están más que asentados y aceptados de forma amplia.

SGML significa "lenguaje estructurado generalizado de marcado" y es el progenitor de todos los lenguajes de marcado. SGML define una sintaxis básica, pero nos permite generar nuestros propios elementos y una estructura de documento adecuados.

A un conjunto de elementos de propósito general, utilizado para describir un determinado tipo de documento, se le denomina aplicación SGML. Una aplicación SGML también incluye reglas que especifican la manera en que los elementos podrán ser organizados, así como otras funcionalidades. Podemos escribir nuestra propia aplicación SGML para que describa un determinado tipo de documento con el que queramos trabajar, o una organización de estándares podría definir una aplicación SGML que describiera un tipo de documento utilizado extensamente. El ejemplo más conocido de este último tipo de aplicación es HTML, que es una aplicación SGML desarrollada en 1991 para describir páginas Web.

SGML puede parecer el lenguaje extensible perfecto para describir documentos web. Sin embargo, los miembros de W3C que analizaban estos temas consideraban que SGML era demasiado complicado e incómodo para suministrar de manera eficiente información en la Web. La flexibilidad y la fluidez de las funcionalidades proporcionadas por SGML harían difícil crear el software necesario para procesar y visualizar la información SGML en los navegadores web. Lo que hacía falta era un subconjunto moderno de SGML, diseñado específicamente para proporcionar información en la Web. En 1996, el Grupo de Trabajo XML de W3C desarrolló este subconjunto, al cual denominaron lenguaje de marcado extensible. Tal como rezaba la cita anteriormente reseñada, XML fue diseñado para una "implementación sencilla", una funcionalidad de la que claramente carecía SGML.

XML es, por tanto, una versión simplificada de SGML, optimizada para la Web. Al igual que SGML, XML nos permite determinar nuestro propio conjunto de elementos a la hora de describir un documento concreto. También al igual que SGML, un individuo o un comité de estándares pueden definir una aplicación XML, que es un subconjunto y una estructura de documento de propósito general, que pueden utilizarse para describir documentos de un determinado tipo (por ejemplo, documentos que contengan fórmula matemáticas o gráficos).

La sintaxis de XML ofece menos opciones que la de SGML, haciendo que la lectura de documentos sea más sencilla para los humanos, y que la creación de navegadores, scripts y páginas Web que accedan y muestren la información de los documentos sea más simple para los programadores.

2.3.2 Los Objetivos Oficiales de XML

A continuación se enumeran los diez objetivos de diseño de XML, enunciados en la especificación oficial de XML, expuesta en el sitio web W3C (http://www.w3.org/TR/REC-xml).

1. "XML se debe poder utilizar directamente en Internet."

   XML fue diseñado para almacenar y suministrar información a través de la Web.

2. "XML debe admitir una gran variedad de aplicaciones."

   Aunque su principal objetivo consiste en proporcionar información a través de la Web, mediante programas de servidor y navegadores, XML también está diseñado para ser usado con otros tipos de programas, como por ejemplo, para escribir scripts de voz que puedan enviarse a través del teléfono.

3. "XML debe ser compatible con SGML."

   Como ya hemos reseñado anteriormente, XML es un subconjunto de propósito especial de SGML. Una de las ventajas de esta funcionalidad es que las herramientas de software de SGML se pueden adaptar muy fácilmente para trabajar con XML.

4. "Debe ser fácil crear programas que procesen documentos de XML."

   Si XML ha de ser práctico, la creación de navegadores y demás programas que procesen documentos XML tendrá que ser muy simple. De hecho, la causa principal de que se formara el subconjunto XML de SGML fue la

complejidad existente para generar programas que procesaran documentos SGML.

5. “El número de funcionalidades opcionales de XML deberá mantenerse en un mínimo absoluto, preferiblemente cero.”

   La existencia de un número mínimo de funcionalidades opcionales en XML hace que sea más sencillo crear programas que procesen documentos XML. La abundancia de opcionales en SGML fue una de las principales causas por las que fue calificado como poco práctico para definir documentos web. Entre las funcionalidades opcionales de SGML estaban la redefinición de los caracteres delimitadores en los marcadores (normalmente <y>), y la omisión del marcador de fin cuando el procesador pudiera averiguar dónde terminaba un elemento. Un programa robusto que procesase documentos SGML tendría que tener en cuenta todas las funcionalidades opcionales, incluso aquellas que apenas se utilicen.

6. “Los documentos XML deben ser inteligibles para los humanos y razonablemente claros.”

   XML está diseñado para convertirse en la lengua para el intercambio de información entre los usuarios y los programas de todo el mundo. El que sea inteligible por los humanos permite alcanzar este objetivo, posibilitando a las personas (y a los programas de software) la lectura y la escritura de documentos XML. Su legibilidad distingue a XML de la mayoría de los formatos propietarios, utilizados en las bases de datos y en los documentos de los procesadores de texto.

   Los humanos pueden leer fácilmente un documento XML, dado que está escrito en texto legible y tiene una estructura lógica de tipo árbol. Podemos incrementar la legibilidad de XML utilizando nombres significativos para los elementos, atributos y entidades de nuestros documentos y añadiendo comentarios adicionales apropiados.

7. “El diseño de XML deberá prepararse rápidamente.”

   XML será un estándar viable únicamente si la comunidad de programadores y usuarios lo adopta. Este estándar necesita por lo tanto ser completado antes de que esta comunidad comience a adoptar estándares alternativos.

8. “El diseño de XML deberá ser formal y conciso.”

La especificación de XML está escrita en un lenguaje formal, utilizado para definir lenguajes informáticos y que se conoce como notación EBNF (Extended Backus-Naur Form). Este lenguaje formal, aunque difícil de leer a primera vista, resuelve las ambigüedades y en último término facilita la creación de documentos XML y especialmente del software de procesamiento de XML, colaborando aún más a la adopción de XML.

9. "Los documentos XML deberán ser fáciles de generar."

   Para que XML se convierta en un lenguaje práctico de marcado para los documentos Web, no sólo debe ser fácil la creación de programas de procesamiento de XML, sino que también los propios documentos XML tendrán que poder crearse de manera sencilla.

10. "La concisión en los marcadores XML tiene una importancia mínima."

    Para conseguir el objetivo 6 (los documentos deberán ser inteligibles para los humanos y razonablemente claros), los marcadores XML no deberán ser tan concisos que lleguen a convertirse en crípticos.

2.3.3 Arquitectura de XML

A continuación se presenta someramente la anatomía de un documento XML en relación con las características empleadas en la realización del presente proyecto.

En XML existen básicamente dos tipos de documentos:

- **Bien formados**: Son todos los que cumplen las especificaciones del lenguaje respecto a las reglas sintácticas que posteriormente se explicarán, sin estar sujetos a unos elementos fijados en una DTD. Aunque posteriormente se explicará con detalle, en primera instancia se puede decir que una DTD es una definición de los elementos que puede incluir un documento XML, la forma en deben hacerlo (el orden de los elementos) y los atributos que se les puede dar. De hecho, los documentos XML deben tener una estructura jerárquica muy estricta que los documentos bien formados deben cumplir.
- **Válidos:** Además de estar bien formados, los documentos válidos siguen una estructura y una semántica determinada por una DTD: sus elementos y la estructura que define el DTD, además de los atributos, deben ajustarse a lo que la DTD expone.

El procesamiento de un documento XML comienza comprobando que el documento está bien formado, y después, si incluye o referencia una DTD, comprobando que se siguen sus reglas.

El ejemplo de documento XML que se utilizará para ir explicando el funcionamiento de XML está contenido en el archivo *"definiciones.xml"*, y contiene la clase *definiciones* recogida en el Modelado del Contenido de la Norma del presente proyecto. Se presenta a continuación parte de su código.

Ejemplo: *"definiciones.xml"*

```
<?xml version="1.0"?>
<!DOCTYPE DEFINICIONES
  [
  <!ENTITY a "&#225;">
  <!ENTITY e "&#233;">
  <!ENTITY i "&#237;">
  <!ENTITY o "&#243;">
  <!ENTITY u "&#250;">
  <!ENTITY n "&#241;">
  <!ELEMENT DEFINICIONES (DEFINICION)*>
<!ELEMENT    DEFINICION
(ACTIVIDAD,QUE,ELEMENTOS*,PDEFINICION)>
  <!ELEMENT ACTIVIDAD (#PCDATA)>
  <!ELEMENT QUE (#PCDATA)>
  <!ELEMENT ELEMENTOS (#PCDATA)>
  <!ELEMENT PDEFINICION (#PCDATA)>
  ]
>
<DEFINICIONES>
  <DEFINICION>
        <ACTIVIDAD>diseno</ACTIVIDAD>
```

<QUE>instalacion</QUE>

<ELEMENTOS>instalacion</ELEMENTOS>

<ELEMENTOS>puntodetoma</ELEMENTOS>

<ELEMENTOS>conducciondealimentacion</ELEMENTOS>

<ELEMENTOS>reddedistribucion</ELEMENTOS>

<PDEFINICION>La instalaci&o;n estará compuesta por: punto de toma en una conducción o depósito, conducción de alimentación y red de distribucción.</PDEFINICION>

</DEFINICION>

…

</DEFINICIONES>

Características Generales

En cuanto a la sintaxis del documento hay que reseñar una serie de detalles importantes:

- Los documentos XML son sensibles a mayúsculas, esto es, en ellos se diferencia las mayúsculas de las minúsculas. Por ello, <DEFINICION> sería una etiqueta diferente a <Definicion>.
- Todos los espacios se tienen en cuenta dentro de las etiquetas y los elementos.
- Hay algunos caracteres reservados que forman parte de la sintaxis de XML: <, >, &, “ y ‘. En su lugar, cuando queramos representarlos habrá que utilizar las entidades, que se describen más adelante.

Es importante diferenciar entre etiquetas y elementos: los elementos son las entidades en sí, lo que tiene contenido, mientras que las etiquetas sólo describen a los elementos. Un documento XML está compuesto por elementos, y en su sintaxis éstos se nombran mediante etiquetas.

Declaración de XML

En ella se indica que el documento está en XML. Aunque es opcional, conviene incluirla en todos los documentos. El atributo de “version” indica la versión de XML usada en el documento. La actual es la versión 1.0.


```
<?xml version="1.0"?>
<!DOCTYPE DEFINICIONES
        [
        <!ENTITY a "&#225;">
        <!ENTITY e "&#233;">
        <!ENTITY i "&#237;">
        <!ENTITY o "&#243;">
        <!ENTITY u "&#250;">
```

DTD: Definición de Tipos de Documento

Como se comentó anteriormente, los documentos XML pueden ser válidos o bien formados. Los válidos son los que su gramática está definida en las DTD.

Las DTD no son más que definiciones de los elementos que se puede incluir un documento XML, la forma en que deben hacerlo (qué elementos van dentro de otros) y los atributos que se les puede dar.

Hay varios modos de incluir una a DTD en un documento XML. En el presente proyecto se ha decidido incluir la DTD dentro del propio documento DTD por una razón muy sencilla: cada DTD que se ha formulado es válida únicamente para el documento para la que es definida, es decir, la DTD de *"definiciones.xml"* es distinto de la de *"procedimientos.xml"*, por lo que no tiene sentido recurrir a un DTD externo cuando sólo se va a utilizar en dicho documento.

A continuación se presenta el contenido DTD del ejemplo que nos ocupa.

```
<?xml version="1.0"?>
<!DOCTYPE DEFINICIONES
        [
        <!ENTITY a "&#225;">
        <!ENTITY e "&#233;">
        <!ENTITY i "&#237;">
        <!ENTITY o "&#243;">
        <!ENTITY u "&#250;">
        <!ENTITY n "&#241;">
        <!ELEMENT DEFINICIONES (DEFINICION)*>
        <!ELEMENT DEFINICION (ACTIVIDAD,QUE,ELEMENTOS*,PDEFINICION)>
        <!ELEMENT ACTIVIDAD (#PCDATA)>
        <!ELEMENT QUE (#PCDATA)>
        <!ELEMENT ELEMENTOS (#PCDATA)>
        <!ELEMENT PDEFINICION (#PCDATA)>
        ]
>
<DEFINICIONES>
        <DEFINICION>
...
```

DTD: Definición de Tipos de Documento

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

La definición de los elementos es bastante intuitiva: después de la cláusula <!ELEMENT se incluye el nombre del elemento (el que luego se indicará en la etiqueta) y después diferentes cosas en función del elemento:

- Entre paréntesis, si el elemento es no vacío, se indica el contenido que puede tener el elemento: la lista de elementos hijos o que descienden de él si los tiene, separados por comas; o el tipo de contenido, normalmente #PCDATA, que indica datos de tipo texto, que son los más habituales.
- Si es un elemento vacío, se indica con la palabra EMPTY.0

A la hora de indicar los elementos descendientes (los que están entre paréntesis), estos van seguidos de unos caracteres especiales : +, * … Sirven para indicar qué uso se permite hacer de estos elementos dentro del documento. En el proyecto se ha utilizado dos tipos de descendientes:

- Los que contienen el carácter "*", que indica uso opcional y múltiple, esto es, puede haber ninguna ocurrencia, una o varias, como es el caso del elemento DEFINICION, que indica que en el documento DEFINICIONES,

que recoge las definiciones que aparecen en la norma, contendrá cualquier número de definiciones.

- Los que no contienen ningún carácter especial, como ACTIVIDAD, que indica que la multiplicidad del elemento ha de ser obligatoriamente de uno.

Así, en el ejemplo que se utiliza, se recoge la siguiente declaración:

```
<!ELEMENT    DEFINICION
  (ACTIVIDAD,QUE,ELEMENTOS*,PDEFINICION)>
```

En ella se indica que cada elemento DEFINICION puede contener los siguientes elementos: un elemento ACTIVIDAD, un elemento QUE, ninguno, uno o varios ELEMENTOS, y un elemento PDEFINICION.

Como se comentó anteriormente, un documento XML presenta una jerarquía muy determinada, definida en la DTD si el documento es valido, pero siempre inherente al documento en cualquier caso, por lo que se puede presentar como un árbol de elementos. Existe un elemento raíz, que siempre debe ser único que se llamará con el nombre que se ponga en la definición del <!DOCTYPE si está asociado a una DTD o cualquiera que se desee en caso contrario. Y de él descienden las ramas de sus respectivos elementos descendientes o hijos. De este modo la representación en forma de árbol de nuestro documento XML de ejemplo sería la siguiente:

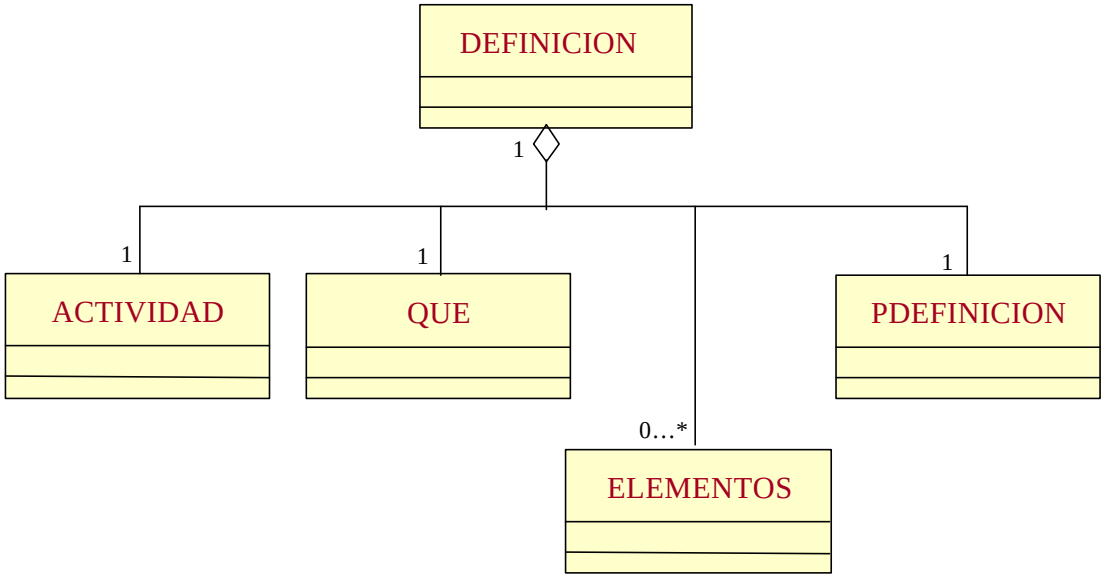


Fig. 2.1. Ejemplo de DTD: Definiciones

La DTD, por ser precisamente la definición de esa jerarquía, describe precisamente la forma de ese árbol. La diferencia está en que la DTD define la

forma del árbol de elementos, y un documento XML válido puede basarse en ella para estructurarse, aunque no tienen que tener en él todos los elementos, si la DTD no obliga a ello. Un documento XML bien formado sólo tendrá que tener una estructura jerarquizada, pero sin tener que ajustarse a ningún DTD concreto.

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

- El nombre de elemento del que estamos declarando los atributos.
- El nombre del atributo.
- Los posibles valores del atributo.
- De forma opcional y entrecomillado, un valor por defecto del atributo.

De esta forma, hay ciertas tablas que se han implementado aprovechando la potencia que otorga la utilización de atributos. Así, la implementación de la Tabla 2 de la sección de Cálculo de la norma se ha realizado mediante el uso de los mismos. Una línea de esta implementación se muestra a continuación:

```
<FILA desde="0" hasta="1000" dotacion="630" caudal="0.030"></FILA>
```

En ella se ha utilizado una lista de cuatro atributos. Los atributos “desde” y “hasta” indican el rango de validez de los datos de salida de la misma, “dotación” y “caudal”. De esta manera la entrada a dicha tabla será refiriéndose al atributo necesario en cada momento.

Las Entidades o Entities

Mediante estos elementos es posible dotar de modularidad a los documentos XML. Se pueden definir dentro de las propios DTDs anteriormente expuestas.

Así se han utilizado entidades para definir en el presente ejemplo las vocales acentuadas y las “ñ”, ya que XML da problemas en la utilización de acentos y de la letra ñ. Siguiendo con el ejemplo anterior, se ha tenido que utilizar la definición de entidades para poder reflejar estos acentos y esta letra en el documento “*definiciones.xml”*. Se han utilizado únicamente en las partes que recogen texto de la norma suceptible de ser presentado posteriormente por pantalla, como <PDEFINICION>, ya que a nivel interno, en vez de utilizar por ejemplo la palabra “diseño” se ha utilizado “diseno” para facilitar la implementación. De esta forma, al comienzo de cada archivo de “definiciones.xml”, “procedimientos.xml”, “ubicación.xml”… encontramos la definición de entidades que se muestra a continuación:

```
<?xml version="1.0"?>
<!DOCTYPE DEFINICIONES
            [
            <!ENTITY a "&#225;">
            <!ENTITY e "&#233;">
            <!ENTITY i "&#237;">
            <!ENTITY o "&#243;">
            <!ENTITY u "&#250;">
            <!ENTITY n "&#241;">
            <!ELEMENT DEFINICIONES (DEFINICION)*>
            <!ELEMENT DEFINICION (ACTIVIDAD,QUE,ELEMENTOS*,PDEFINICION)>
            <!ELEMENT ACTIVIDAD (#PCDATA)>
            <!ELEMENT QUE (#PCDATA)>
            <!ELEMENT ELEMENTOS (#PCDATA)>
            <!ELEMENT PDEFINICION (#PCDATA)>
            ]
>
<DEFINICIONES>
            <DEFINICION>
...
```

Entidades

Así, a la hora de la implementación, en vez de utilizar en <PDEFINICION> la palabra “instalación”, se utiliza “instalaci&o;n”, que una vez procesada por el navegador, dará la salida correcta “instalación”.

```
<PDEFINICION>La instalaci&o;n estar&#225; compuesta por: punto de toma en una conduci&#243;n o
dep&#243;sito, conduci&#243;n de alimentaci&#243;n y red de distribucci&#243;n.</PDEFINICION>
```

Entidades

2.3.4 Visualización de Documentos XML

Los documentos XML pueden ser tratados de cuatro maneras diferentes para su procesamiento y visualización. Estas se relacionan brevemente a continuación:

- Visualización de documentos XML utilizando hojas de estilo en cascada (CSS, Cascading Style Sheets).

  Un documento XML con una hoja de estilo en cascada asociada puede ser abierto directamente por el navegador. No es necesario utilizar una página HTML para acceder y mostrar los datos. Es posiblemente el método más

sencillo para mostrar un documento XML ya que el lenguaje CSS es familiar al que se emplea en las páginas web.

Sin embargo, comparadas con otros métodos de representación de XML, las hojas de estilo están muy limitadas. Aunque una hoja de estilo en cascada proporciona un alto nivel de control sobre la forma que el explorador dará al contenido de los elementos de un documento XML, no nos permite reordenar o modificar este contenido. Tampoco podemos acceder a los atributos, entidades, instrucciones de procesamiento y otros componentes, ni procesar la información que contengan estas unidades, de ahí que debido a los requisitos que debe cumplir el sistema a diseñar, esta manera de tratar los documentos no sea la adecuada.

- Representación de documentos XML mediante asociación de datos.

  La asociación de datos vincula un documento XML a una página HTML, y después asocia los elementos estándares HTML, como SPAN o TABLE, a elementos XML individuales. Los elementos HTML mostrarán automáticamente el contenido de los elementos XML a los que estén asociados.

  La asociación de datos funciona únicamente con aquellos documentos XML que estén estructurados simétricamente, igual que una base de datos típica (es decir, un documento cuyos elementos puedan interpretarse como un conjunto de registros y campos). En su formato más simple, un documento de este tipo estaría formado por un elemento raíz, que contendría una serie de elementos del mismo tipo (los registros), cada uno de los cuales tendría el mismo número de elementos hijo, los cuales contendrían todos los datos de caracteres. Es debido a esta restricción de simetría por lo que este método de representación no fue considerado en la realización del presente proyecto.

- Visualización de documentos XML utilizando hojas de estilo XSL.

  Al igual que sucede con las hojas de estilo en cascada, una hoja de estilo XSL está vinculada a un documento XML e indica al explorador cómo visualizar los datos XML, lo que permite abrir el documento XML directamente en el explorador sin necesidad de utilizar una página HTML como intermediario.

  Para visualizar XML, sin embargo, una hoja de estilo XSL es considerablemente más potente y flexible que una hoja de estilo CSS. Mientras que CSS permite simplemente especificar el formato de cada

elemento XML, una hoja XSL proporciona un control completo sobre la salida. Específicamente, XSL permite seleccionar de forma precisa los datos XML que se quieren visualizar, presentar dichos datos en cualquier orden o disposición y añadir o modificar información con total libertad. XSL proporciona acceso a todos los componentes XML (como elementos, atributos, comentarios e instrucciones de procesamiento), permite ordenar y filtrar fácilmente los datos XML, permite incluir scripts en la hoja de estilo y proporciona un conjunto de métodos de utilidad que pueden invocarse para trabajar con la información.

Una hoja de estilo proporciona, por tanto, acceso a toda la potencia funcional y de formato de HTML, además de proporcionar acceso a los datos y a las características de transformación proporcionadas por el propio XSL.

XSL, sin embargo, es algo más complejo que CSS, y requiere conocimientos previos de HTML. Es, a su vez, una tecnología más reciente que CSS y, por tanto, dispone de menos soporte entre los exploradores actuales.

- Visualización de documentos XML utilizando scripts DOM.

  El denominado Modelo de Objeto de Documento XML (Document Object Model, DOM) está compuesto por un conjunto de objetos de programación que representan los diferentes componentes de un documento XML. Las propiedades y métodos de estos objetos permiten utilizar scripts (en JavaScript) para visualizar el documento XML desde dentro de una página HTML.

  Reciben el nombre de script los programas escritos en lenguajes interpretados, es decir, en aquellos lenguajes como JavaScript que no necesitan ser compilados, sino que es el propio navegador el que se encarga de traducir – interpretar – dicho código. Dichos scripts se introducen directamente en el documento HTML.

  Aunque DOM requiere algo más de trabajo que otras técnicas, almacena los datos XML en una estructura de tipo árbol jerárquico que refleja la estructura jerárquica del documento XML. Por tanto, se puede usar DOM para visualizar cualquier tipo de documento XML, independientemente de si está o no estructurado como un conjunto de registros, y también se puede usar para acceder a cualquier componente de un documento XML, incluyendo

elementos, atributos, instrucciones de procesamiento, comentarios y declaraciones de notación y de entidad.

El almacenamiento de tipo jerárquico que proporciona DOM ha sido la característica más importante a tener en cuenta para ser elegido este método como herramienta para la ejecución del proyecto. Hay que tener en cuenta el enorme potencial para conseguir una búsqueda rápida y eficaz que posee una estructura jerárquica en forma de árbol, que permitirá una mejor gestión de los recursos implicados en la búsqueda. Por otra parte, la visualización de documentos XML mediante DOM implica la utilización de scripts en JavaScript. La generación de estos scripts se realiza de una manera sencilla prestando a cambio una enorme capacidad y versatilidad en cuanto al tratamiento de la información guardada en los documentos de XML. Así por ejemplo, la utilización de los mismos permite una fácil gestión de las tablas que se encuentran en la norma NTE-IFA, de manera que permiten aplicar sencillos búcles de búsqueda y de tratamiento de datos.

Hay que tener en cuenta, por otra parte, que HTML está bastante limitado en cuanto a la interactividad con el usuario, necesaria por otra parte en la implementación que se ha llevado a cabo. Una forma de paliar esta escasa interactividad de HTML es mediante la utilización de JavaScript.

Por todo ello, se consideró la visualización utilizando scripts (en JavaScript) DOM como la más idónea para el proyecto en cuestión, y de ahí que pasemos seguidamente a tratar más en profundidad este método.

2.3.4.1.- Visualización de documentos XML utilizando scripts DOM

Para acceder a un documento HTML utilizando DOM, es preciso vincular el documento DOM a la página HTML, es decir, indicar qué documento o documentos XML van a ser utilizados en dicha página HTML. Para ello se inserta un identificador del archivo a utilizar. Este identificador se consigue empleando un elemento de HTML denominado XML. Por ejemplo, para vincular el documento "*definiciones.xml"*, que contiene los enunciados de tipo definición que contiene la norma, a una página HTML, como puede ser la correspondiente al caso de uso **"Consulta por Contenido",** se utiliza la siguiente sintaxis:

```
<BODY>
<XML ID="XMLdocumento" SRC="definiciones.xml"></XML>
```

</BODY>

donde el identificador ID=”XMLdocumento” permite acceder al documento XML en el script.

En DOM, los objetos de programación que representan el documento XML se conocen con el nombre de nodos. Cuando el explorador procesa un documento XML vinculado y lo almacena dentro de un modelo DOM, crea un nodo por cada uno de los componentes básicos del documento XML, como pueden ser los elementos, atributos e instrucciones de procesamiento.

El modelo DOM utiliza diferentes tipos de nodos para representar diferentes tipos de componentes XML. Por ejemplo, un elemento se almacena en un nodo de elemento y un atributo en un nodo de atributo. Continuando con el ejemplo del archivo “*definiciones.xml*”, la estructura DOM de éste quedaría como sigue:

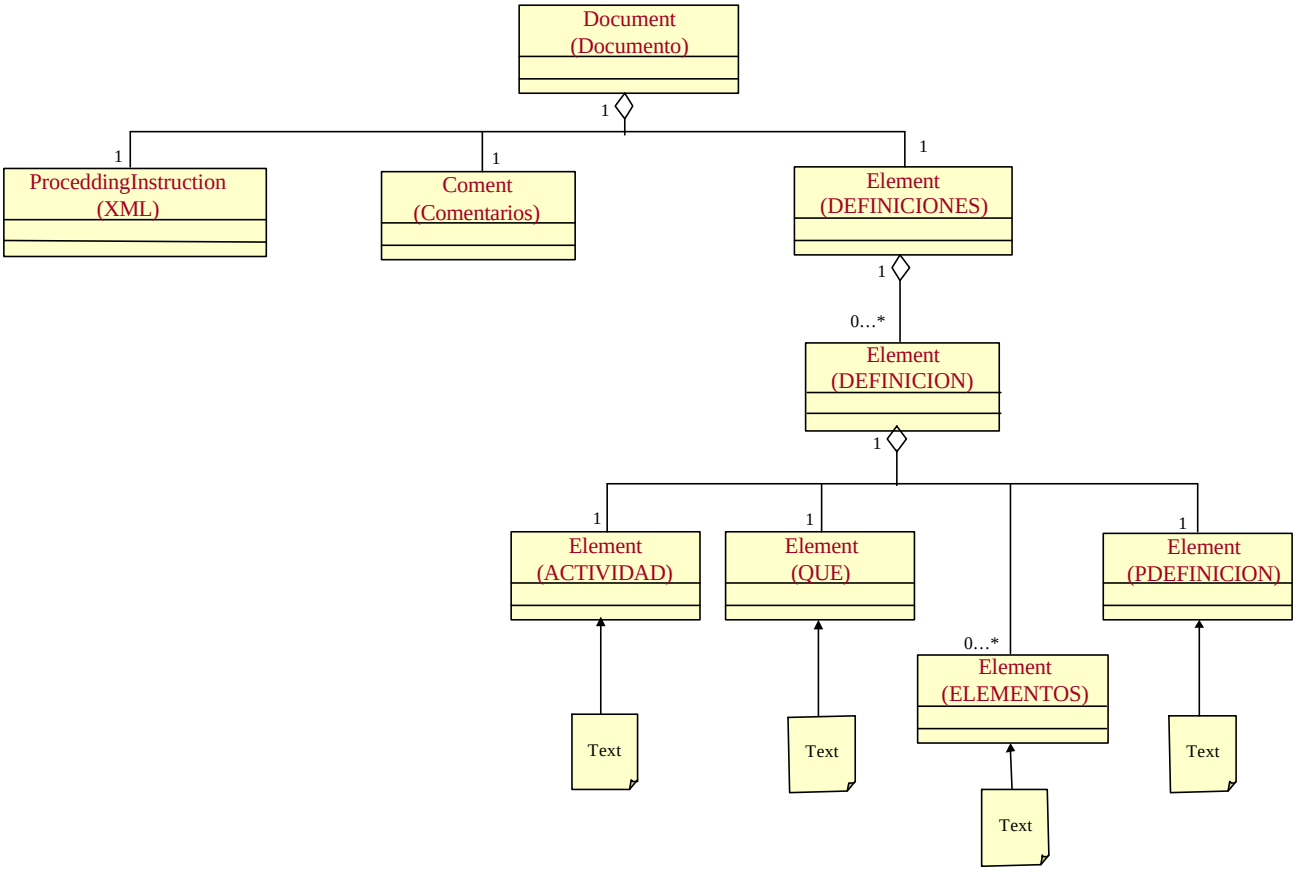


Fig. 2.2. Ejemplo de Estructura DOM

Pueden obtenerse los valores que poseen los nodos a partir de la propiedad *nodeValue* del nodo. Si un componente XML tiene un valor asociado (por ejemplo, un atributo), el valor se almacenará en el valor del nodo. Si en

componente XML no tiene un valor, el modelo DOM asigna *null* al valor del nodo.

El modelo DOM organiza los nodos del documento XML en una estructura jerárquica de árbol que refleja la estructura jerárquica del propio documento. DOM crea un único nodo *Document* que representa al documento XML completo y que sirve como raíz de esa jerarquía.

XML proporciona una serie de propiedades a los nodos que sirven para acceder a los mismos de distintas formas. Estas propiedades se utilizan en aplicaciones JavaScript insertadas en cada una de las páginas que componen la implementación de la aplicación final. A continuación se muestran algunas de estas propiedades que han sido utilizadas en el desarrollo del presente proyecto:

Propiedad	***Descripción***	***Ejemplo***
ChildNodes	Una colección NodeList de todos los nodos hijo de este nodo que no sean de tipo Attribute	FirstNode= Elem.childNodes(0);
FirstChild	El primer nodo hijo de este nodo que no sea de tipo Attribute	FirstChildNode= Elem.firstNode;
LastChild	El último nodo hijo de este nodo que no sea de tipo Attribute	LastChildNode= Elem.lastChild;
NextSibling	El siguiente nodo situado al mismo nivel que este nodo	NextElement= Elem.nextSibling;
ParentNode	El nodo del cual es hijo este nodo	ParentElement= Elem.parentNode;
Text	El contenido completo de texto de este nodo y de todos los nodos descendientes de tipo Element.	Texto= Elem.text;

GetElements ByTagName	Devuelve una colección NodeList (lista de nodos) de todos los elementos del documento que contengan el tipo especificado. Si se pasa "*" devuelve todos los elementos.	Lista= getElementsByTagName ("QUE");

Por otra parte, también se ha utilizado una serie de propiedades del objeto NodeList (lista de nodos) que se resumen a continuación:

Propiedad	***Descripción***	***Ejemplo***
Item (índice)	Devuelve el nodo situado en la posición indicada por el índice que se le pase, donde 0 indica el primer nodo.	Nodo2= Elem.ChildNodes (1);
NextNode	Devuelve el siguiente nodo de la colección, de acuerdo con lo que dicte el puntero interno.	FirstNode= Elem.childNodes.nextNode;
Length	Devuelve el número de nodos contenidos en la colección.	Nodos= Elem.childNodes.length;

2.4 HTML y XML

El uso conjunto de HTML y XML podría parecer en primera instancia redundante. Sin embargo, su uso combinado más que recomendable, resulta casi obligatorio.

En una primera aproximación, se puede decir que mediante XML también se puede definir HTML, de manera que tendríamos el siguiente diagrama:

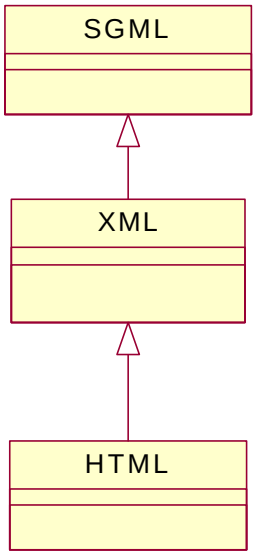


Fig. 2.3. SGML, XML y HTML

De hecho, HTML es simplemente un lenguaje, mientras que XML es un metalenguaje, es decir, un lenguaje para definir lenguajes. Esta es la diferencia fundamental entre ambos.

XML no puede ser considerado el sustituto de HTML, ya que básicamente XML no ha nacido sólo para su aplicación en Internet, sino que se propone como lenguaje de bajo nivel (es decir, a nivel de aplicación, no de programación) para intercambio de información estructurada entre diferentes plataformas.

Por otra parte, en Internet Explorer 5 se puede abrir un documento XML con una hoja de estilo asociada, directamente con el explorador, sin tener que utilizar una página HTML. Sin embargo, los otros dos métodos principales para visualizar documentos XML (asociación de datos y scripts DOM) utilizan páginas web HTML como vehículo para mostrar documentos XML. Incluso utilizando una hoja de estilo, se emplearía en último término HTML para indicar al explorador cómo dar formato a los datos XML.

Así pues, XML empleado junto a HTML, mejora a este último en dos sentidos: establece un estándar al que atenerse, y por otra parte, separa contenido y presentación. De esta manera, utilizando conjuntamente XML y HTML ampliamos enormemente la capacidad de las páginas web para:

- Suministrar virtualmente cualquier tipo de documento.

- Ordenar, filtrar, reorganizar, localizar y manipular información de cualquier manera.
- Presentar información altamente estructurada.

En definitiva, XML fue diseñado para interoperar con HTML.

2.5 JavaScript

El lenguaje HTML permite dotar las páginas Web de una atractiva información visual, no obstante le falta cierto grado de interactividad para el usuario, es decir, en la mayoría de casos, una página Web es un mero escaparate.

Para dotar una página Web de más interactividad podemos insertar, por ejemplo, applets de Java. Un applet de Java es una miniaplicación construida en este lenguaje. Sin embargo Java es un lenguaje bastante complicado, en especial si no se tienen conocimientos de C++, o programación orientada a objetos.

Para llenar el vacío existente entre la sencillez y poca interactividad de HTML, y el grado de interactividad y dificultad de Java, las empresas de software se dispusieron a desarrollar órdenes fáciles de utilizar.

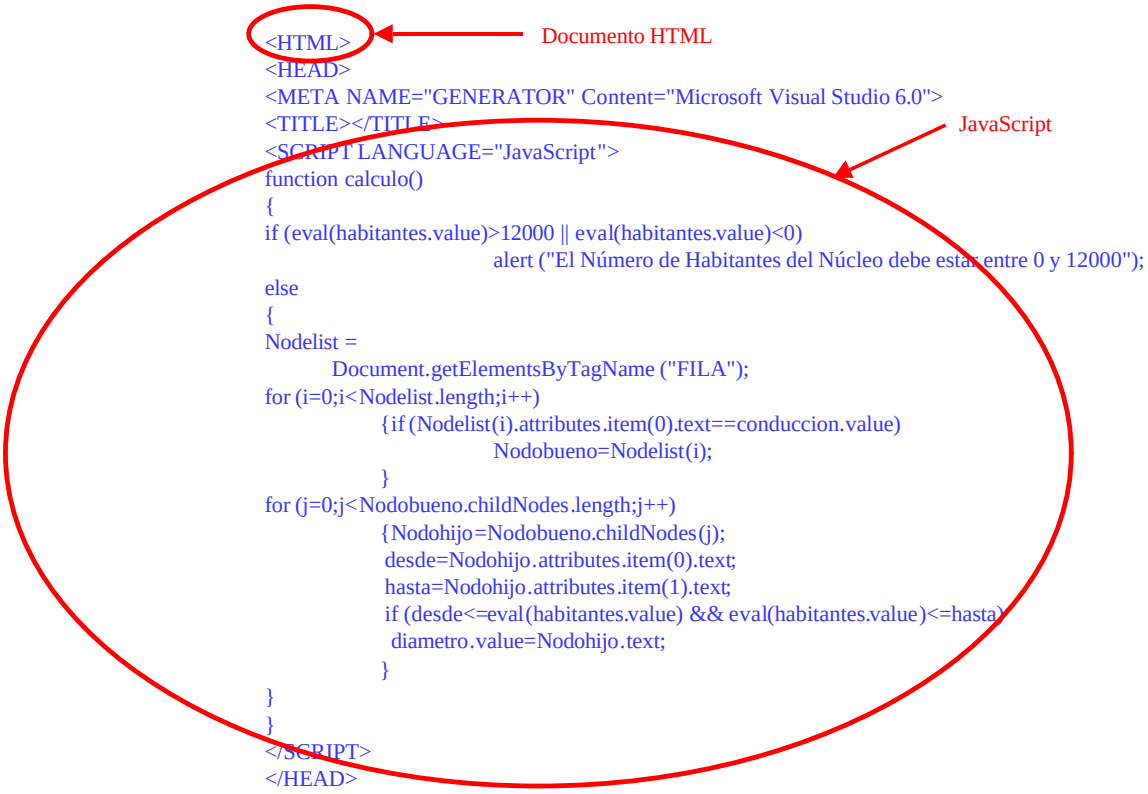
Así Netscape Communications introdujo el lenguaje LiveScript. Poco después la empresa Sun Microsystems, creadora del lenguaje Java, se unió a Netscape para conseguir que LiveScript fuese adoptado como lenguaje estándar de internet para la escritura de órdenes de la Web.

Puesto que LiveScript tenía muchas semejanzas con Java, el lenguaje fue renombrado JavaScript. JavaScript es pues un lenguaje de programación dirigido a los creadores de páginas Web.

Además de paliar la poca interactividad que posee el HTML, la utilización de XML en el desarrollo del proyecto invita a la utilización de JavaScript. Efectivamente, como se reflejó en anteriores líneas, una de las posibles formas de visualización de los documentos XML era mediante DOM, que requería en su sistemática la utilización de scripts en JavaScript.

Todo ello justifica sobradamente la utilización de JavaScript en el presente proyecto.

Los scripts de JavaScript irán incrustados dentro de las páginas Web en las que actúan, tal y como se muestra en el siguiente ejemplo:




```
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
<TITLE></TITLE>
<SCRIPT LANGUAGE="JavaScript">
function calculo()
{
if (eval(habitantes.value)>12000 || eval(habitantes.value)<0)
                        alert ("El Número de Habitantes del Núcleo debe estar entre 0 y 12000");
else
{
Nodelist =
        Document.getElementsByTagName ("FILA");
for (i=0;i<Nodelist.length;i++)
                {if (Nodelist(i).attributes.item(0).text==conduccion.value)
                        Nodobueno=Nodelist(i);
                }
for (j=0;j<Nodobueno.childNodes.length;j++)
                {Nodohijo=Nodobueno.childNodes(j);
                 desde=Nodohijo.attributes.item(0).text;
                 hasta=Nodohijo.attributes.item(1).text;
                 if (desde<=eval(habitantes.value) && eval(habitantes.value)<=hasta)
                  diametro.value=Nodohijo.text;
                }
}
}
</SCRIPT>
</HEAD>
<BODY onload=calculo() bgColor=papayawhip>
<XML ID="Document" SRC="tabla6.xml"></XML>
```

Las dos principales características de JavaScript son, por un lado, que es un lenguaje basado en objetos, y por otro, JavaScript es un lenguaje orientado a eventos (debido al tipo de entornos en los que se utiliza, como Windows). Esto implica que gran parte de la programación en JavaScript se centra en describir objetos (con sus variables instancia y métodos de clase) y escribir funciones que correspondan a movimientos del ratón, como pulsación de teclas, apertura y cierre de ventanas, o carga de una página, entre otros eventos.

JavaScript no es un lenguaje de propósito general, es decir, no permite un control absoluto sobre los recursos del ordenador, tal y como permiten otros lenguajes. Cada programa en Javascript sólo tiene acceso al documento HTML en el que va inmerso, y si acaso, a las ventanas en las que se ejecuta el navegador dentro del cuál se está ejecutando el programa en Javascript.

Por otro lado, tampoco es un lenguaje orientado a objetos, ya que, por ejemplo, no existe el concepto de clase. Es basado en objetos, de modo que se trabaja directamente con instancias de objetos.

Es conveniente dejar bien clara la diferencia entre Java y JavaScript. Aunque pudiera pensarse que son una misma cosa, son dos lenguajes totalmente diferentes, como ya se ha indicado anteriormente.

Señalamos a continuación algunas de las diferencias más notables entre ambos lenguajes:

- Java es un lenguaje de propósito general, y por lo tanto se puede utilizar para programar aunque no sea para la Web. JavaScript es específico de la Web.
- JavaScript es un lenguaje interpretado y ejecutado por el cliente. Java es un lenguaje compilado por el servidor y que puede ser ejecutado en el cliente.
- JavaScript es basado en objetos. Java es orientado a objetos. Los objetos usados en JavaScript existen cuando empieza a ejecutarse el programa. No existen clases ni por tanto ninguna de las características que de ello derivan, como la herencia.
- El código del programa en JavaScript va inscrito en la página HTML en la que se ejecuta.
- JavaScript es un lenguaje poco tipificado, es decir, no es necesario declarar el tipo de los datos y se realizan conversiones de unos a otros automáticamente cuando es necesario y es viable. Java es, por el contrario, fuertemente tipificado, como C o Pascal.