PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR
SEDE AMBATO

ESCUELA DE INGENIERÍA DE SISTEMAS

DISERTACIÓN DE GRADO PREVIA LA OBTENCIÓN DEL TÍTULO DE INGENIERO DE SISTEMAS

"Estudio del Metalenguaje XML (Extensible Markup Language), orientado al desarrollo de una aplicación Web de actualización y consulta de notas, para los alumnos de la PUCESA"

ALEX NAPOLEÓN ROSERO BURBANO

DIRECTOR DE DISERTACIÓN

Ing. Janio Jadán, M. Sc.

Ambato, 2002

PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR
SEDE AMBATO

ESCUELA DE INGENIERÍA DE SISTEMAS

DISERTACIÓN DE GRADO PREVIA LA OBTENCIÓN DEL TÍTULO DE INGENIERO DE SISTEMAS

"Estudio del Metalenguaje XML (Extensible Markup Language), orientado al desarrollo de una aplicación Web de actualización y consulta de notas, para los alumnos de la PUCESA"

Director : _____
Ing. Janio Jadán, M.Sc.

Revisores: _____
Ing. Patricio Medina

_____
Ing. Natasha Bayas

Alex Napoleón Rosero Burbano

Ambato, 2002

Dedicatoria

A esos seres maravillosos, mis padres, por todo el amor y sacrificio entregados.

A mis hermanas, por su apoyo y compañía durante el transcurso de mi vida.

A mis verdaderos amigos, por su derroche de aprecio, ayuda y alegría.

A la memoria de mi abuelita Enriqueta, por la belleza y grandeza de su corazón.

A la memoria de mi mejor amigo, Luis, por haber compartido conmigo la época más bonita de mi vida.

"Que Dios los colme de bendiciones"

Agradecimiento

Desde el fondo de mi corazón, el más profundo agradecimiento a mis padres.

ÍNDICE

CAPÍTULO I 1

1 GENERALIDADES XML 1

1.1 Historia 1

1.2 El lenguaje Markup. 3

1.3 ¿Cómo Trabaja el Markup? 4

1.4 Lenguajes y Metalenguajes 5

1.5 Definición 6

1.6 Separación de Datos, Presentación y Proceso 6

1.7 Objetivos en el Diseño del XML 7

1.8 Principales Características del XML 10

1.9 Estándares XML relacionados entre sí 11

1.10 Documentos Autodescriptivos 12

1.10.1 Documentos Bien Formados 13

1.10.2 Documentos Válidos 14

1.11 El Procesador XML 14

1.12 El Elemento Raíz 15

1.13 Nodos Padres, nodos hijos 15

1.14 Anatomía de un Error 17

CAPÍTULO II 19

2 ESTRUCTURA Y SINTAXIS XML. 19

2.1 Estructura Lógica XML 19

2.1.1 El Prólogo 20

2.1.2 El Elemento Documento. 23

2.2 Sintaxis XML 24

3.1.2 Concepto 40
3.1.3 DTD Interno 40
3.1.4 DTD Externo 41
3.1.5 Definición de un DTD 42
3.1.6 Estructura Básica de un DTD 43
3.1.7 Declaración de Elementos 45
3.1.7.1 Regla ANY 45
3.1.7.2 Regla EMPTY 46
3.1.7.3 Declaración Mixta 46
3.1.7.4 Declaración Múltiple 47
3.1.7.5 Regla #PCDATA 47
3.1.7.6 Agrupamiento, Ocurrencias y Símbolos Elemento 48
3.1.8 Declaración de Atributos 49
3.1.9 Declaración de Entidades 52
3.1.9.1 Entidades Generales o Internas 52
3.1.9.2 Entidades Externas 53
3.1.9.3 Entidades Parámetros 55
3.1.10 Otras Palabras Claves DTD 55
3.1.10.1 IGNORE e INCLUDE 56
3.1.10.2 Comentarios 57
3.1.11 Combinación de DTD Externo e Interno 57
3.1.12 DTD ya definido 59
3.1.13 Limitaciones de un DTD 59
3.2 Documentos esquemas (SCHEMAS) 60
3.2.1 Introducción 60

3.2.2 Definición 61
3.2.3 Componentes de un Documento Esquema 61
3.2.4 Definición de Tipos Complejos 64
3.2.4.1 Definición de Tipos Existentes 65
3.2.5 Definición de Tipos Simples 66
3.2.6 Facetas 67
3.2.6.1 La Faceta Pattern 67
3.2.6.2 La Faceta Enumeración 68
3.2.7 Atributos 69
3.2.8 Atributo Contenido 71
3.2.9 Grupos 72
3.2.10 Anotaciones 73
3.2.11 resúmen de DTD y Documentos Esquemas 74

CAPÍTULO IV 75

4 EL MODELO DE OBJETOS DEL DOCUMENTO (DOM) 75

4.1 Definición 75
4.2 Origen del Modelo de Objetos de Documento 76
4.3 Niveles del Modelo de Objetos de Documento 76
4.4 Representación de un Documento 77
4.5 Intercambio de Datos con Plataforma Neutral 79
4.6 Lo que el Modelo de Objetos de Documentos no es 80
4.7 Recuperación de la Información de un Documento 80
4.8 El Analizador (parser) MSXML 81
4.8.1 Interfaz de Documentos 82
4.8.2 Interfaz de Nodos 84

4.8.3 Interfaz de Listas de Nodos 86
4.8.4 Interfaz de Mapas de Nodos 87
4.8.5 Interfaz de Manipulación de Errores 88
4.9 Funcionamiento del Modelo de Objetos de Documento 89
4.9.1 Creación del Analizador (parser) 89
4.9.2 Carga y Validación del Documento 90
4.9.3 Lectura a través del documento cargado. 93
4.9.4 Creación y manipulación de documentos y su contenido. 95
4.9.5 Retorno del documento a un usuario cliente 100
CAPÍTULO V 101
5 FORMATO Y TECNOLOGÍAS AFINES A XML 101
5.1 Hojas de Estilo en Cascada (CSS - CASCADE STYLESHEET) 101
5.1.1 Introducción 101
5.1.2 Definición 102
5.1.3 Fundamentos y Sintaxis de las Hojas de Estilo en Cascada 102
5.1.3.1 Selectores 102
5.1.3.2 Selectores Especiales 106
5.1.3.3 Unidades utilizadas en las Hojas de Estilo en Cascada 107
5.1.3.4 Algunas propiedades utilizadas en las Hojas de Estilo en Cascada .. 108
5.1.3.5 Herencia 113
5.1.3.6 Cascada 113
5.1.3.7 Comentarios 116
5.1.4 ¿Cómo asociar una hoja de estilos en cascada a un documento XML? 116
5.1.4.1 Conectando a un Stylesheet externo. 116
5.1.4.2 Incrustando Hojas de Estilo en Cascada 117

5.1.4.3 Unión de Hojas de Estilo en Cascada 118
5.1.5 Ejemplo del uso de Hojas de Estilo en Cascada 119
5.1.6 Ventajas y Beneficios del uso de Hojas de Estilo en Cascada 124
5.1.7 Desventaja del uso de Hojas de Estilo en Cascada 126
5.2 XSL (Extensible Stylesheet Language) 126
5.2.1 Introducción 126
5.2.2 Definición 127
5.2.3 ¿Cómo trabaja el XSL? 128
5.2.4 ¿Cómo se conecta un XSL a un documento XML? 128
5.2.5 XSL Orientado a Transformaciones (XSLT) 129
5.2.5.1 Estructura de un XSLT 132
5.2.5.2 Patrones XSLT 138
5.2.5.3 Elementos XSL utilizados en Hojas de Estilo de Transformación. .. 141
5.2.5.4 Ordenamiento de datos a través de una hoja de estilos XSLT 143
5.2.6 Transformación de documentos XML a través del DOM (programación) utilizando una hoja de estilos XSL 144
5.2.7 Cuadro Comparativo entre CSS y XSL 145
5.3 Xpath (Lenguaje de búsqueda) 147
5.3.1 Definición 147
5.3.2 Sintaxis Básica de XPath 147
5.3.3 Entendiendo el funcionamiento del Nodo Contexto 148
5.3.4 Ejes 151
5.3.5 Nodos-Prueba 152
5.3.6 Predicados 153
5.3.7 Funciones XPath 154

5.4 XLINK (Lenguaje de enlace) 155
5.4.1 Definición 155
5.4.2 Limitaciones de los enlaces HTML 155
5.4.3 Declaración del Namespace XLink 157
5.4.4 Algunos Atributos XLink 157
5.4.5 Tipos de Enlaces 159
5.4.5.1 Enlace Simple 159
5.4.5.2 Enlace Extendido 160
5.5 XPOINTER (Lenguaje de localización) 162
5.5.1 Definición 162
5.5.2 Apuntadores en HTML 163
5.5.3 Utilización de Xpointer 164
CAPÍTULO VI 169
6 HERRAMIENTAS RELACIONADAS CON XML 169
6.1 Introducción 169
6.2 HTML (HyperText Markup Language) 169
6.2.1 Definición 169
6.2.2 Estructura Básica de un documento HTML 170
6.2.2.1 Encabezamiento <HEAD> 171
6.2.2.2 Cuerpo <BODY> 172
6.3 XHTML 181
6.3.1 Definición 181
6.3.2 Mejoras 182
6.3.3 Diferencias con HTML 183
6.3.4 Novedades incluidas en XHTML 186

6.4 SGML (Standard generalized markup language) 187
6.4.1 Definición 187
6.4.2 Ventajas del SGML 188
6.4.3 Características de un Documento SGML 189
6.4.4 Estructura de un Documento SGML 189
6.4.4.1 Prólogo SGML. 190
6.4.4.2 Los Elementos. 190
6.4.4.3 Los Atributos. 191
CAPÍTULO VII 192
7 SOFTWARE XML 192
7.1 Introducción 192
7.2 Editores XML 193
7.2.1 XML SPY 193
7.2.2 XMetal 193
7.2.3 XML Styler 194
7.2.4 XML Notepad 194
7.2.5 XPublish 194
7.3 Parsers XML 195
7.3.1 Lista de Parsers No Validadores : 195
7.3.1.1 XP 195
7.3.1.2 Lark 196
7.3.1.3 HXA 196
7.3.2 Lista de Parsers Validadores 196
7.3.2.1 MSXML 196
7.3.2.2 Parser XML de IBM para Java 197

7.3.2.3 DXP 197
7.4 Software de Administración de Contenido y Base de Datos 197
7.4.1 Tamino 197
7.4.2 DataChannel XML 198
7.4.3 Balise 198
7.4.4 Konstructor 199
CAPÍTULO VIII 200
8 APLICACIÓN PRÁCTICA (Consulta y Actualización de Notas) 200
8.1 Descripción Global del Proyecto 200
8.2 Análisis del Software 201
8.2.1 Flujo de Información 201
8.3 Diseño del sistema 205
8.3.1 Selección y descripción de herramientas 205
8.3.1.1 Visual Interdev versión 6.0 205
8.3.1.2 Microsoft Access 2000 207
8.3.1.3 Internet Information Services 207
8.3.1.4 Tecnología Active Server Pages (ASP) 209
8.3.1.5 ActiveX Data Objects (ADO) 214
8.3.2 Diseño de la base de datos 220
8.3.2.1 Descripción de la Base de datos 220
8.4 Estructura de la Aplicación 224
8.5 XML en la Aplicación 225
8.5.1 Obtención de XML a partir de una página ASP, utilizando ADO (Access Data Object) y aplicando xslt 225
8.5.2 Utilización de una Isla de Datos 229

CONCLUSIONES 232
RECOMENDACIONES 234
BIBLIOGRAFÍA 236
ANEXOS 238

PREFACIO

Desde tiempos lejanos hasta la actualidad, la información ha constituido el núcleo de millones y millones de actividades, las cuales han sido desarrolladas dentro de un sinnúmero de campos y ciencias, con el propósito de cumplir metas y dar a luz trascendentales cambios y resultados, ya sea en beneficio de algunos o de toda la humanidad.

En los tiempos modernos, la tecnología constituye el motor que mueve las riendas de la humanidad, y el tratamiento de la información representa sin duda alguna la principal actividad, en torno a la cual la tecnología trabaja, en busca de mejores y óptimas soluciones para el desarrollo global del planeta y sus habitantes.

El cotidiano tratamiento de la información y el aumento de la misma han sido causas más que suficientes para el incremento de alternativas tecnológicas que mantengan la estabilidad y el normal flujo de dicha información, dentro de las diferentes actividades que alrededor de ésta suceden. Desde la creación de la primera red, por los años 50 y 60, pasando por la evolución de la red de redes (Internet) en los 90 y llegando actualmente a tener en nuestras manos un sinnúmero de sofisticadas herramientas tanto de hardware como de software, el manejo de la información (datos) es el punto central alrededor del cual éstas y muchas más tecnologías han nacido.

La creación y constante evolución del Internet, han transformado la visión del mundo y su proyección hacia el futuro ya no es la misma desde que ésta herramienta apareció. La cantidad de datos que a través del Internet fluyen, constituye información esencial para

como desplegamos las páginas HTML hoy en día. El estándar de datos es XML y las extensiones XML.

CAPÍTULO I

1 GENERALIDADES XML

1.1 HISTORIA

La versión 1.0 del Lenguaje XML (Extensible Markup Language), es una recomendación del W3C (World Wide Web Consortium) desde Febrero de 1998, pero se ha trabajado en ella desde un par de años antes. 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 el 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. Está además asociado a la recomendación del W3C DOM (Document Object Model), aprobado también en 1998. Éste no es más que un modelo de objetos (en forma de API) que permite acceder a las diferentes partes que pueden componer un documento XML o HTML.

SGML proporciona un modo consistente y preciso de aplicar etiquetas para describir las partes que componen un documento, permitiendo además el intercambio de documentos entre diferentes plataformas. Sin embargo, el problema que se atribuye a SGML es su excesiva dificultad; basta pensar que la recomendación ocupa unas 400 páginas.

Así que, manteniendo su misma filosofía, de él se derivó XML como subconjunto simplificado, eliminando las partes más engorrosas y menos útiles. Como su padre, y este es un aspecto extremadamente importante, XML es un metalenguaje, es decir: un lenguaje para definir lenguajes. Los elementos que lo componen pueden dar

información sobre lo que contienen, no necesariamente sobre su estructura física o presentación, como ocurre en HTML. Usando SGML, por otro lado, se definió precisamente el HTML, el lenguaje que es tan conocido.

HTML es simplemente un lenguaje, mientras que XML como se ha dicho es un metalenguaje. Y esa es la diferencia fundamental entre estas dos plataformas, de la que derivan todas las demás.

Desde el año 1998 XML ha tenido un crecimiento exponencial, ya que desde entonces ha tenido apariciones en los medios de comunicación de todo tipo, menciones en páginas web, soporte software, tutoriales, etc.

Los programas que lo soportan han crecido del mismo modo, y al día de hoy no hay empresa de software que no anuncie la compatibilidad de sus productos más vendidos con este nuevo estándar: Microsoft (Office 2000), Oracle (Oracle 8i, Web Application Server) o Lotus (Notes) son tres claros ejemplos de ello. Aún más increíble es pensar que hay empresas que se han creado entorno a él, u otras que han movido su actividad hacia su ámbito (de SGML a XML, por ejemplo, ArborText).

Por lo expuesto anteriormente, surge la interrogante: ¿será XML el sustituto de HTML, que tan bien se conoce?. No, esa es la respuesta, ya que básicamente XML no ha nacido sólo para su aplicación en Internet, sino que se propone como lenguaje de bajo nivel (en el ámbito de aplicación, no de programación) para intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de

texto, hojas de cálculo, y casi cualquier cosa que se pueda pensar. Pues bien, no lo sustituirá, pero, aplicado en Internet, sí va a mejorar algo de lo que HTML adolece: establece un estándar fijo al que atenerse, y separa el contenido de su presentación. Esto significa que a partir de su aplicación definitiva, para ver un documento web los usuarios no estarán sujetos a la parte del estándar de hojas de estilo (CSS) que soporte el Navigator de Netscape o el Internet Explorer de Microsoft, ni al lenguaje de script del servidor y al modelo de objetos definido por Microsoft, Nestcape, etc. Además el usuario tampoco estará atado a la plataforma: se podrá ver la misma información desde una PC, un navegador textual, una lavadora, un microondas o un reloj con acceso a Internet, con presentaciones adecuadas a cada entorno.

Se puede suponer de este modo que XML constituye la capa más baja dentro del nivel de aplicación, sobre el que se puede montar cualquier estructura de tratamiento de documentos, hasta llegar a la presentación.

1.2 EL LENGUAJE MARKUP.

El Markup generalmente se presta para dos propósitos: para determinar el formato o para fijar la estructura o el significado. Por consiguiente un Lenguaje de Marcas (Markup Language) es un conjunto estandarizado de etiquetas de marcado que se conforman para definir la sintaxis y la gramática.

Probablemente la primera ocasión en que se escuchó Markup, fue cuando se utilizó HTML, pero el concepto y el uso de Markups son mucho más antiguos, así por ejemplo Word Perfect y los archivos con formato RTF ya utilizaban etiquetas.

Los lenguajes de marcas no son equivalentes a los lenguajes de programación, aunque se definan igualmente como "lenguajes". Son sistemas complejos de descripción de información, normalmente documentos, que si se ajustan a SGML, se pueden controlar desde cualquier editor ASCII.

Se puede decir que existen tres utilizaciones básicas de los lenguajes de marcas: los que sirven principalmente para describir su contenido, los que sirven más que nada para definir su formato y los que realizan las dos funciones indistintamente. Las aplicaciones de bases de datos son buenas referencias del primer sistema, los programas de tratamiento de textos son ejemplos típicos del segundo tipo, y aunque no lo parezca, el HTML es la muestra más conocida del tercer modelo.

1.3 ¿CÓMO TRABAJA EL MARKUP?

En un nivel rudimentario tanto el Markup antiguo como el nuevo trabajan de la misma manera. Generalmente abarcan una librería de pares de etiquetas que se sitúan alrededor de palabras o caracteres para alterar su apariencia o significado. Esto es lo que se denomina un elemento. Por ejemplo aquí tenemos un pequeño recorte de un documento RTF:

{\f0 Hola ! Yo uso } {\b\f0\fs28 markup}

Interpretado por un procesador RTF, esto quedaría así:

Hola ! Yo uso **markup**

1.4 LENGUAJES Y METALENGUAJES

SGML es un estándar para el manejo de información adoptada por la ISO en el año de 1986, sin embargo sus raíces nacieron mucho más antes. Fue diseñado para proveer la manera de crear documentos con plataformas independientes y aplicaciones independientes que guarden formatos, indexen y conecten información. Con un mecanismo gramatical, los usuarios pueden definir la estructura de sus documentos a través de la creación de etiquetas.

SGML es un metalenguaje, es decir puede ser usado para crear otros lenguajes, de igual forma XML también es un metalenguaje, ya que se puede usar para crear estructuras que describan el contenido sin tomar en cuenta, como éste va ha ser mostrado. Por otro lado HTML, es puramente un lenguaje y no puede crear conjuntos de sí mismo, además describe como el documento va ha ser mostrado, pero no dice nada de lo que el documento es.

El mundo necesita una manera de describir información, y los creadores de XML intentan llenar el vacío con el uso de un metalenguaje basado en SGML, pero dejando a un lado su complejidad. XML es más fácil de aprender y ponerlo en práctica que el SGML, el cual resulta mucho más tedioso y dificultoso.

Los creadores del XML además han incluido ciertas características que el SGML no posee, como por ejemplo la capacidad que tiene el XML de trabajar en conjunto con el HTML para integrar y desplegar datos.

1.5 DEFINICIÓN

El Lenguaje de Marcaje Extensible (Extensible Markup Language, XML) es un lenguaje de meta-marcaje (meta-markup) que proporciona un formato para describir datos estructurados. Esto facilita declaraciones más precisas de contenido y resultados de búsquedas con más significado entre muchas plataformas. Además, el XML habilitará una nueva generación de aplicaciones, manipulación y visualización de datos basadas en Web.

El XML es valioso para la Internet, así como para los grandes ambientes de intranets corporativas porque proporciona interoperabilidad utilizando un formato basado en estándares, abierto y flexible con nuevas formas de accesar a bases de datos propietarias y proporcionar datos a los clientes de Web. Las aplicaciones se pueden construir más rápidamente, son más fáciles de mantener y pueden proporcionar varias vistas de los datos estructurados.

1.6 SEPARACIÓN DE DATOS, PRESENTACIÓN Y PROCESO

El poder y la belleza del XML consisten en la separación entre la interfaz de usuario y los datos estructurados. El HTML especifica como visualizar datos en un navegador, el XML define el contenido. Por ejemplo, en HTML se utiliza etiquetas para decirle al navegador que despliegue en negritas o itálicas; en XML solo se utiliza etiquetas para describir los datos, tales como el nombre de la ciudad, temperatura y presión barométrica. En XML se utiliza hojas de estilo tales como el Lenguaje de Estilo Extensible (Extensible Style Language, XSL) y las Hojas de Estilo en Cascada (Cascading Style Sheets, CSS) para presentar los datos en un navegador. El XML

separa los datos de la presentación y el proceso, permitiendo desplegar y procesar la información tal como se desee, al aplicar diferentes hojas de estilo y aplicaciones.

Esta separación de datos de la presentación permite una integración de datos perfecta de fuentes diversas. La información de clientes, ordenes de compra, resultados de investigaciones, pagos de facturas, registros médicos, datos de catálogo y cualquier otra información se puede convertir a XML, permitiendo a los datos ser intercambiados en línea tan fácilmente como las páginas de HTML despliegan datos hoy. Los datos codificados en XML pueden ser transmitidos sobre la Web hasta el escritorio. No es necesario retroajustar información en formatos propietarios almacenados en bases de datos o documentos de mainframes, y debido a que se usa el HTTP para transmitir el XML sobre la red, no se necesitan cambios para esta función.

1.7 OBJETIVOS EN EL DISEÑO DEL XML

Los creadores del XML, buscaban que este sea usado en más y otros lugares que solo la Web. Su intención fue que el XML debía convertirse en una manera de compartir datos entre diferentes y variadas aplicaciones. Por esta razón establecieron los siguientes objetivos de diseño para esta especificación del W3C:

a) XML será directamente usable en el Internet

Esto no significa que XML sea limitado únicamente para Internet, mas bien se pretende que la complejidad del SGML sea removida, manteniendo la simpleza del XML, usándolo en el Internet.

b) XML soportará una amplia variedad de aplicaciones

XML puede ser usado en la Web o en un tradicional ambiente Cliente / Servidor. No hay una tecnología específica detrás de esto, por lo tanto cualquier tecnología puede usarlo.

c) XML será compatible con SGML

XML es un subconjunto del SGML, lo que significa que lo que se haga en XML, es válido en SGML.

d) Será fácil escribir programas que procesen Documentos XML

Solo haciéndolo fácil y permitiendo que los programadores escriban sus propios analizadores (parsers) en cualquier lenguaje, el XML ganará aceptabilidad.

e) El número de Características Opcionales en XML debe ser el mínimo, o sea cero

SGML, posee otras opciones, muchas de las cuales no son usadas, y si son utilizadas para realizar cierta aplicación, luego no serán entendibles para otra aplicación SGML. Si el XML tiene cero opciones todo procesador XML puede analizar un documento XML.

f) Los Documentos XML, deben ser legibles y razonablemente claros

XML está diseñado para describir la estructura del contenido. Si se mira un documento XML, se podrá deducir que es una factura de una venta, por ejemplo. Pero no se podrá saber de que manera esa factura será mostrada al usuario final.

g) El Diseño XML debe ser preparado rápidamente

Los creadores del XML, se propusieron terminar rápidamente el producto final XML, ya que ellos temían que otro grupo de desarrolladores saque al mercado esta herramienta con fines de lucro.

h) El Diseño del XML, debe ser formal y conciso

La especificación XML usa el estándar EBNF (Extended Backus-Naur Form) para formalizar su redacción. EBNF es un formato usado para describir lenguajes de programación y mantener sus especificaciones concisas.

i) Un Documento XML, será fácil de crear

Un documento XML, puede ser creado en un simple editor de texto. Sin embargo existen muchos editores XML en el mercado.

j) No podrá existir ambigüedad en los documentos XML

En XML, no puede existir ambigüedad. SGML y HTML, permiten técnicas de truncamiento como por ejemplo el no cerrar etiquetas de algunos elementos. Esto hace

que sea complicado de leer. Si se cierra cada etiqueta abierta, no existe la posibilidad de confusión.

1.8 PRINCIPALES CARACTERÍSTICAS DEL XML

- Es una arquitectura más abierta y extensible. No se necesita versiones para que puedan funcionar en futuros navegadores. Los identificadores pueden crearse de manera simple y ser adaptados en Internet / Intranet por medio de un validador de documentos (parser).
- Mayor consistencia, homogeneidad y amplitud de los identificadores descriptivos del documento con XML (los RDF Resource Description FrameWork), en comparación a los atributos de la etiqueta <META> del HTML.
- Integración de los datos de las fuentes mas dispares. Se podrá hacer el intercambio de documentos entre las aplicaciones tanto en el propio PC como en una red local o extensa.
- Datos compuestos de múltiples aplicaciones. La extensibilidad y flexibilidad de este lenguaje permitirá agrupar una variedad amplia de aplicaciones, desde páginas web hasta bases de datos.
- Gestión y manipulación de los datos desde el propio cliente web.
- Los motores de búsqueda devolverán respuestas más adecuadas y precisas, ya que la codificación del contenido web en XML consigue que la estructura de la información resulte más accesible.
- Se desarrollarán de manera extensible las búsquedas personalizables y subjetivas para robots y agentes inteligentes. También conllevará que los clientes web puedan

ser más autónomos para desarrollar tareas que actualmente se ejecutan en el servidor.

- Se permitirá un comportamiento más estable y actualizable de las aplicaciones web, incluyendo enlaces bidireccionales y almacenados de forma externa (El famoso epígrafe "404 file not found" desaparecerá).
- Exportabilidad a otros formatos de publicación (papel, web, cd-rom, etc.). El documento maestro de la edición electrónica podría ser un documento XML que se integraría en el formato deseado de manera directa.

1.9 ESTÁNDARES XML RELACIONADOS ENTRE SÍ

La iniciativa XML consta de un conjunto de estándares relacionados entre sí:

- **Extensible Markup Language (XML)**, es una recomendación, la fase final del proceso de aprobación del W3C. Esto significa que el estándar es estable y que los desarrolladores de Web y de herramientas pueden adoptarlo plenamente.
- **Namespaces en XML**, es una recomendación que describe la sintaxis y la compatibilidad de los espacios de nombres para los analizadores XML que los reconocen.
- **Document Object Model (DOM) Especificación Nivel 1**, es una recomendación que ofrece un estándar para el acceso mediante programación a los datos estructurados (a través de scripts), de modo que los desarrolladores puedan interactuar de forma coherente con los datos basados en XML y computarlos.

- **Extensible Stylesheet Language (XSL)**, esta especificación tiene dos secciones modulares: el lenguaje de transformación XSL y los objetos de formato XSL. El lenguaje de transformación puede servir para transformar el XML de cara a su presentación. Puesto que las dos partes de XSL son modulares, el lenguaje de transformación se puede utilizar de forma independiente para transformaciones con fines generales, incluida la conversión de XML en HTML bien formado. CSS (Cascading Style Sheets) se puede aplicar a datos XML de estructura simple, pero no puede modificar el orden de presentación de la información.
- **XML Linking Language (Xlink - XLL)** y su compañero **XML Pointer Language (XPointer)**. XLL es un lenguaje de vinculación XML que ofrece vínculos en XML parecidos a los de HTML, pero más potentes. Con XLL, los vínculos pueden tener varias direcciones y pueden existir en el nivel de los objetos, no sólo en el nivel de las páginas. Internet Explorer 5.0 no tiene incorporada la compatibilidad con XLL. Por otro lado Xpointer describe cómo se puede apuntar a un lugar específico de un determinado documento XML. Resumiendo, Xlink determina el documento al que se desea acceder y Xpointer marca el lugar exacto de dicho documento.

1.10 DOCUMENTOS AUTODESCRIPTIVOS

Un documento XML da la facilidad de describir nuestros elementos, mediante las etiquetas, por ejemplo: <NombreCliente>Alex</NombreCliente> o también <Direccion>Calle Corazón</Direccion>, esto brinda facilidad de entendimiento al leer documentos XML. Hay básicamente 2 criterios por los cuales debe ser conformado un documento: este debe ser: Bien Formado y debe ser Válido.

1.10.1 DOCUMENTOS BIEN FORMADOS

Bien Formado significa que un documento cumple con las estructuras básicas que un documento XML tiene. Las Reglas básicas para tener un documento bien formado son:

- Así como el SGML y HTML, el XML usa el menor que (<) y el mayor que (>) como sus delimitadores estándar.
- Toda etiqueta abierta debe tener una etiqueta de cerrado. La excepción es un elemento vacío (uno que no tiene valor) en este caso la etiqueta de apertura y de cerrado puede ser la misma, excepto cuando la de cerrado está precedida por /. Por ejemplo <NombreCliente/>.
- Los atributos de las etiquetas deben estar entre comillas. En HTML se puede tener: <FONT SIZE = 3> y el 3 no necesita estar entre comillas. En XML la misma sentencia podría ser escrita de esta manera <FONT SIZE = "3">.
- Los elementos deben ser apropiadamente anidados. Todo documento XML, tiene un elemento raíz y todos los demás elementos son hijos del elemento raíz. Por ejemplo:

```
<Cliente>
        <NombreCliente saludo= "Sr">Alex</ NombreCliente >
</Cliente>
```

El elemento raíz es <*Cliente*> y el primer hijo es <*NombreCliente*>.

- Los elementos son case-sensitivity, es decir se distingue de diferente forma las minúsculas como las mayúsculas. Por ejemplo <*NombreCliente*> no es lo mismo que <*NOMBRECLIENTE*>, XML las trata como dos etiquetas distintas.

1.10.2 DOCUMENTOS VÁLIDOS

Un documento válido es aquel cuya estructura está conforme a las reglas definidas por el DTD (Document Type Definition) o por el Documento Esquema (Schema). Estos definen lo que cada elemento puede contener y la estructura de organización del documento XML. Los DTDs tienen su propia y única sintaxis para describir un documento XML, mientras que el Documento Esquema usa estándar XML para definir el documento.

Cuando el parser analiza el documento XML con el DTD o el Documento Esquema, todos los valores deben obedecer y cumplir las reglas determinadas. XML es por consiguiente validado por esas reglas.

1.11 EL PROCESADOR XML

Un Procesador XML (Parser) es aquel que analiza gramaticalmente un documento XML. Si no se incluye un DTD o un Documento Esquema (Schema), el único criterio que será analizado será el de documento bien formado. Microsoft Internet Explorer 5.0 tiene un procesador XML nativo, sin embargo técnicamente hablando no es un "procesador puro" ya que es simplemente añadido con IE 5.0. El parser de IE 5.0 no valida un documento XML.

Microsoft Internet Explorer 5.0 posee soporte para XML, esto significa que se puede pasar documentos válidos y bien formados directamente al browser IE 5.0. Cuando IE 5.0 encuentra un documento con la extensión ".xml" automáticamente analiza el documento. El soporte nativo para XML del IE 5.0 es más por conveniencia que por

necesidad. IE 5.0 únicamente posee una manera rápida para ver un archivo XML dentro del browser.

1.12 EL ELEMENTO RAÍZ

El siguiente es un ejemplo de un documento XML básico:

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

<PRIMER_NOMBRE>Alex</PRIMER_NOMBRE>

<PRIMER_APELLIDO>Rosero</PRIMER_APELLIDO>
        </NOMBRE>
        <DIRECCION>
                <CALLE_INFO>Calle Corazon</CALLE_INFO>
                <CIUDAD>Ambato</CIUDAD>
                <PAIS>Ecuador</PAIS>
        </DIRECCION>
        <TELEFONO>852273</TELEFONO>
</CONTACTO>
```

En el ejemplo citado anteriormente, el elemento *<CONTACTO>* es el Elemento Raíz o Elemento Documento. Todo documento XML debe tener un Elemento Documento.

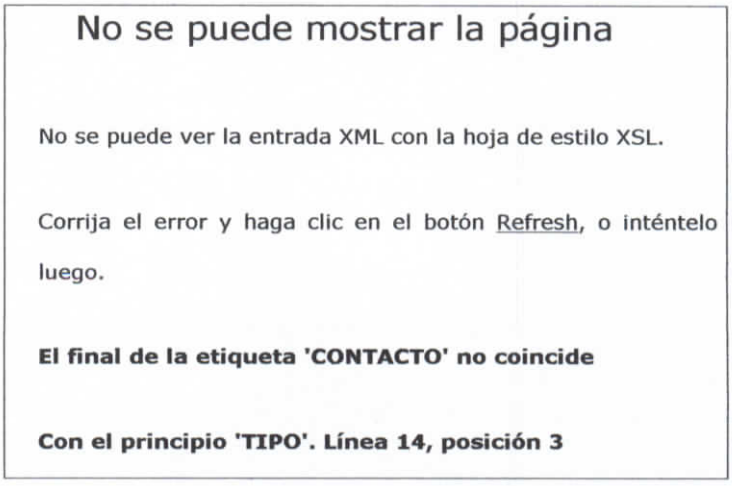
1.13 NODOS PADRES, NODOS HIJOS

Generalmente hablando, los nodos hijos ayudan a describir a sus padres, y se los puede usar cuantas veces sea necesario en el documento. Pero se debe tener cuidado en no complicarse. Un buen documento XML debe ser fácil de entender en su estructura, independientemente de cuan largo sea.

¿Qué pasaría si se analiza el siguiente documento XML?.

```
<CONTACTO>
    <TIPO>Personal
        <NOMBRE>
            <PRIMER_NOMBRE>Alex</PRIMER_NOMBRE>

<PRIMER_APELLIDO>Rosero</PRIMER_APELLIDO>
        </NOMBRE>
        <DIRECCION>
            <CALLE_INFO>Calle Corazon</CALLE_INFO>
            <CIUDAD>Ambato</CIUDAD>
            <PAIS>Ecuador</PAIS>
        </DIRECCION>
        <TELEFONO>852273</TELEFONO>
</CONTACTO>
```

En este ejemplo estamos quebrando una regla esencial del XML: ¡Cerrar todas las etiquetas!. El browser nos indica que deacuerdo con el procesador XML, la etiqueta *<CONTACTO>* no se encuentra apropiadamente cerrada. El error mostrado en el browser será el siguiente:



Dependiendo de cómo se desee estructurar la información, se puede solucionar esto de dos maneras:

- Cerrar la etiqueta *<TIPO>* después del valor, así: <TIPO>Personal</TIPO>
- Colocar la etiqueta de cerrado *</TIPO>* antes de la etiqueta de cerrado *</CONTACTO>*.

1.14 ANATOMÍA DE UN ERROR.

El procesador XML analiza un documento XML, detectando un nodo y luego buscando sus nodos hijos. Si encuentra un nodo hijo, analiza de la misma manera el siguiente nodo, así: en el documento anterior, empieza con el Elemento Raíz *<CONTACTO>*, y chequea sus nodos hijos, encontrando el nodo *<TIPO>*, luego chequea *<TIPO>*, encontrando el nodo abierto *<NOMBRE>*, luego chequea *<NOMBRE>* y encuentra sus hijos, y así sucesivamente. Al haber encontrado la etiqueta *<TIPO>*, el procesador XML, está esperando un dato texto, elementos hijos o la etiqueta de cerrado *</TIPO>*, pero al contrario de todo eso lo que ha encontrado es la etiqueta de cerrado *</CONTACTO>*, razón por la cual el procesador arroja un error.

En otro caso, si se omite las comillas de un atributo, por ejemplo:

```
<?xml version ="1.0"?>
<CONTACTO>
        <NOMBRE  TIPO = Personal>
        ...
        ...
        ...
</CONTACTO>
```

El error mostrado en el browser, será el siguiente:

No se puede mostrar la página

No se puede ver la entrada XML con la hoja de estilo XSL.

Corrija el error y haga clic en el botón Refresh, o inténtelo luego.

Se esperaba una cadena literal, pero no se encontraron las comillas de apertura. Línea 3, posición 15

CAPÍTULO II

2 ESTRUCTURA Y SINTAXIS XML.

2.1 ESTRUCTURA LÓGICA XML

La estructura lógica, puede ser descrita como el armazón del documento XML. Todo documento XML, tiene tres partes lógicas: el Prólogo, el Elemento Documento, y el Epílogo. El Prólogo y el Epílogo son opcionales. El Prólogo es usado para decirle al analizador (parser) como interpretar el Elemento Documento. El propósito del Epílogo no está claro, debido a que en XML no hay una etiqueta de final de documento, la mayoría de analizadores XML, usan la etiqueta de cerrado del elemento raíz. Por consiguiente es improbable que el Epílogo sea interpretado correctamente.

La estructura básica de un documento XML es la siguiente:

```
<?xml version ="1.0"?>
<!-- la línea de arriba corresponde al prólogo -->

<!-- las líneas de abajo están contenidas dentro del elemento: CONTACTO
-->
        <!-- el cual es el Elemento Documento (Elemento Raíz)-->

<CONTACTO>
        <NOMBRE>
                <PRIMER_NOMBRE>Alex</PRIMER_NOMBRE>
                <PRIMER_APELLIDO>Rosero</PRIMER_APELLIDO>
        </NOMBRE>
        <DIRECCION>
                <CALLE_INFO>Calle Corazon #119</CALLE_INFO>
                <CIUDAD>Ambato</CIUDAD>
                <PAIS>Ecuador</PAIS>
        </DIRECCION>
        <TELEFONO>852273</TELEFONO>
</CONTACTO>

<!-- El epílogo podría ir aquí -->
```

Incluso en HTML, se tiene los elementos (etiquetas) <*html*>, <*head*> y <*body*> los cuales tienen cierta relación uno con el otro. Los elementos <*head*> y <*body*> deben estar contenidos dentro del elemento <*html*>. El elemento <*body*> usualmente sigue al elemento <*head*>. Toda la información que queramos desplegar al usuario debe estar contenida en el elemento <*body*>. La excepción a esto es el elemento <*title*>. Esta etiqueta es usada para desplegar texto en la barra del browser y aunque es una práctica estándar colocarla dentro del elemento <*head*>, IE 5.0 y Netscape Navigator pueden interpretarla sin importar en que parte del documento se encuentre.

2.1.1 EL PRÓLOGO

El Prólogo es opcional, sin embargo este entrega al analizador (parser) XML la información acerca del documento y su contenido, por consiguiente debe ser siempre incluido. El Prólogo consta de 2 partes:

a) La Declaración XML.

Esta parte del Prólogo hace exactamente lo que su nombre dice, declara que se está tratando con un documento XML y qué versión del XML es conforme para ese documento. Una simple Declaración XML luce así:

<?xml versión="1.0"? >

Esto le dice al analizador (parser), que el documento XML debe acceder con la versión 1.0 de la especificación XML. El atributo versión es requerido cuando usamos la Declaración XML. Hay que usar siempre minúsculas al momento de declararla.

Las etiquetas <?....?> señalan una instrucción de proceso y le dicen al analizador (parser) que debe poner atención en esta etiqueta, es como que le señalara que algo importante tiene que hacer. La Declaración XML tiene además dos atributos opcionales, así:

<?xml versión= "1.0" encoding="UTF-16" standalone="yes"? >

El atributo *encoding*, señala al analizador (parser) qué codificación de caracteres debe usar. Por defecto los parsers XML usan "UTF-8" o "UTF-16" (Unicode Transformation Format) dependiendo del formato de la Declaración XML. UTF-16 es mucho más útil y tiene total capacidad de lectura de caracteres unicode de 16 bits. El atributo *encoding* es el único en XML que no es case sensivity.

El atributo *standalone*, le indica al parser que el documento XML puede ser validado por si mismo o necesita un DTD externo o un Documento Esquema (Schema). Cuando el valor es "yes" (es el valor por defecto si el atributo no esta indicado) toda declaración está incluida en el documento XML. Cuando el valor es "no" un DTD o Documento Esquema externo es requerido.

b) La Declaración del Tipo de Documento

La Declaración del Tipo de Documento establece las reglas gramaticales para el documento o dirige a un archivo externo donde dichas reglas pueden ser encontradas. La Declaración del Tipo de Documento es opcional, pero si es incluida, esta debe aparecer inmediatamente después de la Declaración XML y antes del Elemento

Documento. Si esto es excluido, el documento puede ser bien formado, pero nunca puede ser validado, ya hay con que validar el documento.

Hay que tomar muy en cuenta que el Prólogo contiene La Declaración del Tipo de Documento y no la Definición del Tipo de Documento; la primera contiene una referencia a la segunda. La Declaración del Tipo de Documento es requerida en todo documento válido. Adicionalmente los comentarios pueden ser incluidos en el Prólogo.

Si se tiene el siguiente Prólogo:

```
<?xml versión="1.0"?>
<!DOCTYPE Contactos SYSTEM "Contactos.dtd">
```

se determina que la Declaración del Tipo de Documento se dirige al archivo externo "Contactos.dtd" el cual contiene una Definición del Tipo de Documento.

Hay que notar el uso del delimitador de etiqueta !, el cual se usa para señalar una referencia de entrada. En esta instancia, la referencia es para una entrada DOCTYPE. Esto le dice al procesador XML, que el documento elemento es llamado Contactos, y el archivo DTD actual puede ser encontrado en el sistema local (por la directiva SYSTEM) y su nombre es "Contactos.dtd". Los DTDs son opcionales, pero ellos son requeridos cuando se necesita validar documentos XML.

Otro método para aplicar reglas a un documento XML, es con un Documento Esquema (Schema). Los Documentos Esquemas son similares a los DTDs en los cuales se puede definir la estructura y el contenido correcto de las etiquetas XML. La gran ventaja de

los Documentos Esquemas es que, a diferencia de los DTDs, ellos usan sintaxis XML para describir el documento.

2.1.2 EL ELEMENTO DOCUMENTO.

Todos los datos en un documento XML están contenidos en un único elemento, cuyo nombre está dado por el autor del documento, por ejemplo:

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

<CONTACTO_INFO>
    <CONTACTO TIPO="negocios">
      <NOMBRE>
            <PRIMER_NOMBRE>Alex</PRIMER_NOMBRE>
            <PRIMER_APELLIDO>Rosero</PRIMER_APELLIDO>
      </NOMBRE>
      <DIRECCION>
            <CALLE_INFO>Calle Corazon #119</CALLE_INFO>
            <CIUDAD>Ambato</CIUDAD>
            <PAIS>Ecuador</PAIS>
      </DIRECCION>
      <TELEFONO>852273</TELEFONO>
    </CONTACTO>

    <CONTACTO TIPO="personal">
      <NOMBRE>
            <PRIMER_NOMBRE>Fabian</PRIMER_NOMBRE>
             <PRIMER_APELLIDO>Altamirano</PRIMER_APELLIDO>
      </NOMBRE>
      <DIRECCION>
            <CALLE_INFO>Calle Venecia</CALLE_INFO>
            <CIUDAD>Quito</CIUDAD>
            <PAIS>Ecuador</PAIS>
      </DIRECCION>
      <TELEFONO>02555399</TELEFONO>
    </CONTACTO>
</CONTACTO_INFO>
```

El Elemento Documento en el documento anterior es *<CONTACTO_INFO>*. No se puede tener más de un Elemento Documento en el mismo documento, pero como se señaló anteriormente el Elemento Documento puede tener tantos hijos como necesite. Esto se puede comparar con la etiqueta *<HTML> ...</HTML>*. Todos los elementos de cualquier documento XML, están contenidos dentro del Elemento Documento o Elemento Raíz.

Cabe señalar que todo documento XML, aparte de tener una estructura lógica, posee también una estructura física, la cual corresponde a los datos que llenan el documento. Es decir, que cuando un documento es completado con todos los datos que se espera, ese instante se está creando su estructura física, la cual puede abarcar información de todo tipo, siempre y cuando sea válida.

2.2 SINTAXIS XML

El HTML no requiere un estricto apegamiento a sus reglas y sintaxis, pero esto no sucede con XML. Las reglas sintácticas XML son pocas y fáciles de aprender y difícilmente olvidables. Las reglas HTML, pueden ser olvidadas por el tratamiento con el diseño y formato, permitiendo hacer ciertas suposiciones cuando las reglas no están siendo cumplidas. El HTML está lleno de ambigüedades y cambiando a cada instante.

El reciente aprobamiento del estándar XHTML, ha definido a éste con reglas estrictas, el XHTML es una implementación HTML en XML. La principal diferencia entre XHTML y HTML es que las reglas del primero son un poco más estrictas que las del segundo.

Reglas en XHTML:

- Las etiquetas son case sensitive.
- Todas las etiquetas deben estar cerradas
- Los valores de los atributos deben estar encerrados entre comillas
- La etiqueta <*HEAD*> es requerida.

Todas estas reglas a excepción de la última son aplicadas al XML.

2.2.1 ETIQUETAS

Las etiquetas son nombres de elementos, delimitados entre los signos menor que y mayor que (<....>), en donde además se usa el slash (/) para indicar una etiqueta de cerrado. Es decir, escribir etiquetas en XML es lo mismo que en HTML. Más allá de estas similitudes visuales existe una inalcanzable diferencia en la función global del XML, la cual empieza con, cómo las etiquetas son usadas para marcar datos.

2.2.1.1 Convención de Nombres para Etiquetas

Los nombres de etiquetas o nombres de elementos pueden empezar con una letra permitida, la línea baja o los dos puntos. Después del primer carácter cualquier letra, dígito, línea baja, dos puntos, punto o guión es aceptado. Hay que recordar que XML usa unicode, por lo tanto cualquier letra significa mucho más que un estándar ASCII, "significa una letra en cualquier lenguaje".

2.2.1.2 Etiquetas de Apertura, de Cerrado y Vacías

Un par de etiquetas XML contiene texto, el cual es el dato que ellas describen. Estos datos deben estar contenidos dentro de una etiqueta abierta y una cerrada para ser aceptado por el parser, así:

<nombre>Alex</nombre>

A diferencia del HTML, en XML la etiqueta de cerrado es requerida. La etiqueta de apertura, la de cerrado y su valor forman un elemento. Puede haber ocasiones en las que se desee tener un elemento sin valor. En este caso se puede crear una etiqueta vacía. Una etiqueta vacía es una etiqueta de apertura y una de cerrado en una. Por ejemplo:

<nombre/>

Esta etiqueta vacía solo mantendrá un lugar para un elemento nombre sin que actualmente tenga valor. Esta forma de truncamiento no es necesitada en etiquetas HTML, porque el procesador HTML generalmente no interpreta el significado de elementos anidados. En HTML el siguiente código es aceptable:

El perro <B> salta sobre la <I> mesa <B><I>

HTML requiere una cantidad limitada de anidaciones correctas en varias áreas fundamentales: la estructura general del documento, tablas y elementos de tablas, y elementos de forma, pero la violación de estas reglas no termina en un colapso o página error como lo sería en XML.

2.2.2 ELEMENTOS

Los elementos constituyen la parte central, el corazón (kernel) de un documento XML. El Elemento Documento o Elemento Raíz constituye el padre de otros elementos y éstos a su vez pueden contener otros elementos.

2.2.2.1 Elementos Hijos

Todo elemento aparte del Elemento Documento es un elemento hijo. Los elementos hijos son de cuatro tipos: Si el elemento hijo solo contiene otros elementos hijos, se dice que es **element content.** Si contiene solo un valor es **character content**. Si contiene juntos, elementos y valores es **mixed content.** Y además existe un elemento **empty.**

2.2.2.2 Anidamiento

Es la manera como XML acomoda sus elementos dentro de un árbol jerárquico. Las reglas para este anidamiento son estrictamente obligatorias. Los browsers no permiten un inapropiado anidamiento en XML. La mayoría de datos que se puede describir y manipular posee algún patrón de anidamiento, por ejemplo:

```
<arbol>
        <rama>
                <nido>
                        <pajaro>
                                <pluma>...</pluma>
                        </pajaro>
                </nido>
        <rama>
<arbol>
```

Lo que esta jerarquía señala es que: la pluma pertenece al pájaro, no al nido ni a la rama. También que una particular pluma, no puede estar en más de un pájaro, un pájaro diferente, tendrá plumas de su propiedad.

2.2.2.3 Anidamiento Inadecuado

En HTML, el anidamiento es solo importante en determinadas áreas del documento, como en formas o tablas; incluso si no se cumple en estas áreas (anidamiento) el browser intentará presentar el documento, adivinando que es lo que puede significar. A continuación se detalla un error de anidamiento.

```
<arbol>
        <rama>
                <nido>
                        <pajaro>
                                <pluma>...</pluma>

                                <pluma>...
                        </nido>
                </pluma>
                </rama>
        </pajaro>
<arbol>
```

El primer problema es que la segunda pluma esta cerrada después de que se cierre el nido, entonces no se puede determinar si esta pluma pertenece a pájaro o a rama. Luego se puede ver que pájaro esta cerrado después de haber cerrado rama, por lo tanto no se sabe si el pájaro pertenece al nido o al árbol.

2.2.2.4 Comentarios

Los comentarios pueden aparecer en cualquier lugar del documento XML, a excepción del inicio, el cual es reservado para la Declaración XML. Los comentarios en XML, siguen el mismo formato que en HTML. Éstos están cerrados por los delimitadores <!--...-->. Como se conoce, los comentarios son útiles al momento de recordar lo que se está haciendo o para colocar otro tipo de información que sea útil en la lectura del documento. Sin embargo la gran característica de los comentarios es aislar partes del documento, esto ayuda cuando se está realizando pruebas de los programas en desarrollo. Por ejemplo:

```
<?xml version ="1.0" encoding="UTF-16" standalone="yes" ?>
<CONTACTO_INFO>
   <CONTACTO TIPO = "negocios">
      <NOMBRE>
         <NOMBRE>Alex</NOMBRE>
         <APELLIDO>Rosero</APELLIDO>
      </NOMBRE>
      <DIRECCION>
         <CALLE_INFO>Calle Corazon</CALLE_INFO>
         <CIUDAD>Ambato</CIUDAD>
         <PAIS>Ecuador</PAIS>
      </DIRECCION>
      <TELEFONO>852273</TELEFONO>
   </CONTACTO>
<!--comentario
   <CONTACTO TIPO = "personal">
      <NOMBRE>
         <NOMBRE>Fabian</NOMBRE>
         <APELLIDO>Altamirano</APELLIDO>
      </NOMBRE>
      <DIRECCION>
         <CALLE_INFO>Lugo 118 y Madrid</CALLE_INFO>
         <CIUDAD>Quito</CIUDAD>
         <PAIS>Ecuador</PAIS>
      </DIRECCION>
      <TELEFONO>02555399</TELEFONO>
   </CONTACTO>
...
-->
</CONTACTO_INFO>
```

Solo el primer elemento *<CONTACTO>* será usado. Es importante recordar que no se debe comentar fuera de la etiqueta de cerrado correspondiente al Elemento Raíz, porque se recibirá un error del analizador (parser).

2.2.3 ATRIBUTOS

Los atributos son aquellos que se usan para decir algo acerca de un elemento. Permiten añadir descripción a un elemento sin afectar su valor. En efecto se podría diseñar documentos XML para usar solo atributos sin valores explícitos. En el siguiente ejemplo la etiqueta *<CONTACTO>* tiene un atributo llamado *TIPO*, éste es usado para describir el tipo de contacto, que en este caso es "negocios", así:

```
<CONTACTO  TIPO="negocios" >
        <NOMBRE>...</NOMBRE>
```

Se puede también añadir un atributo a la etiqueta *<NOMBRE>*, para indicar como la persona debe ser llamada:

```
<NOMBRE prefijo="Dr.">...</NOMBRE>
```

El valor de un atributo debe siempre estar encerrado entre comillas. Cuando se incluye valores literales también se puede usar apóstrofes. Si se necesita usar el apóstrofe y las comillas en un valor literal, solo se debe usar uno diferente para delimitar el valor. Si se necesita que los 2 aparezcan en el valor, hay que tomar en cuenta los siguientes ejemplos:

"Cadena delimitada con comillas"

'Cadena delimitada con apóstrofes'

"No se puede usar com'illas en esta cadena"

' "No se puede usar comillas para delimitar esta cadena" '

' "La cadena Mac's tiene apostrofe y comillas" '

Se puede crear atributos para ayudar a describir las etiquetas, dándoles cualquier valor que parezca apropiado. Sin embargo hay 2 atributos especiales que son definidos por la recomendación XML versión 1.0: *xml:space* y *xml:lang*.

2.2.3.1 Atributos Especiales

Se puede usar los atributos especiales, para pasar información del documento al analizador XML (parser). Estos atributos usan la sintaxis XML Namespace.

Los Namespaces previenen problemas de ambigüedad y duplicación con nombres. El W3C los describe como: " Una colección de nombres, identificados por una referencia URI, la cual es usada en los documentos XML como tipos elementos y nombres atributos".

- **xml:space**

Este atributo, a pesar de ser bien documentado y formalmente especificado rompe la regla de propiedad para los atributos. Si este atributo es especificado en una etiqueta elemento se aplica a todos los nodos hijos perteneciendo a este elemento también.

```
<Nombre_Compania xml:space>
        Alfa
</Nombre_Compania>
```

Esto es similar a la etiqueta <*pre*> en HTML. Sin embargo no es una regla absoluta, su funcionalidad debe ser implementada por la aplicación XML o el parser. Lo que significa que esto no podría trabajar como se desearía.

- **xml:lang**

Este atributo permite especificar el lenguaje actual del elemento. También se aplica no solo al elemento, sino a sus hijos también.

```
<systemPrompt>
        <color xml:lang="en-GB"Chose a colour</color>
        < color xml:lang="en-US"Chose a color</color >
</systemPrompt>
```

El uso de este atributo no es obligatorio para las aplicaciones XML, y si se lo usa hay que tener precaución.

2.2.4 SECCIÓN CDATA

En los documentos XML, una sección CDATA indica al analizador (parser) que ignore todos los caracteres de marcas. La sección CDATA de un documento XML no es interpretada por el analizador (parser). Cuando se usa XSL stylesheets para transformar documentos XML en HTML, cualquier código debe estar en la sección CDATA. Para crear una sección CDATA, se utiliza la siguiente sintaxis:

```
<![CDATA[...los datos aquí...]]>
<SCRIPT>
        <![CDATA[
            function blowup()
            {
                alert("Uh  Oh")
            }
        ]]>
</SCRIPT>
```

Este ejemplo muestra como se debe incluir una función en un XSL stylesheet.

2.2.5 INSTRUCCIONES DE PROCESAMIENTO

Estas instrucciones son usadas para dirigir la aplicación XML o el analizador (parser). Todas las Instrucciones de Procesamiento tienen el formato <?instrucción recibida?>, como por ejemplo esta simple declaración:

<?xml versión="1.0"?>

El receptor es el analizador (parser) XML o aplicación, y la instrucción identifica a este documento como acatador de la versión 1.0 XML. Otra muy usada instrucción es la que está incluida en la referencia XSL stylesheet:

<?xml-stylesheet type="text/xsl" href="contactos.xsl"?>

Aquí se le dice al analizador (parser) XML stylesheet que la stylesheet es de tipo "text-xsl" y que puede ser encontrada en el archivo contactos.xsl

2.2.6 ENTIDADES

Hay dos tipos de entidades en XML: Referencias Carácter y Referencias Entidad.

Entidad	Descripción
&apos;	' - un apóstrofe
&amp;	&- un ampersand
&gt;	> - signo mayor que
&lt;	< - signo menor que
&quot;	" – comilla

Tabla # 2: Entidades predefinidas en XML

Es posible definir entidades propias del desarrollador, las cuales deben estar definidas en un DTD.

2.3 NAMESPACES

2.3.1 DEFINICIÓN

Un XML Namespace es una colección de nombres que pueden ser usados como elementos o nombres de atributos en un documento XML. Los Namespaces distinguen los nombres de elementos únicamente sobre la web para evitar conflictos entre elementos con el mismo nombre. El Namespace es identificado por algún Uniform Resource Identifier (URI), un Uniform Resource Locator (URL), o un Uniform Resource Number (URN). Los URIs son usados simplemente porque son globalmente únicos a través de Internet.

2.3.2.2 Declaración de un Namespace por Defecto

Un Namespace declarado sin un prefijo se convierte por defecto en el Namespace del documento. Todos los elementos y los atributos en el documento que no tienen un prefijo, pertenecerán al Namespace por defecto. El siguiente ejemplo declara que el elemento *<LIBRO>* y todos los elementos y atributos dentro de el *(<TITULO>, <PRECIO>*, moneda) son del Namespace "urn:BookLovers.org:LibroInfo", así:

```
<LIBRO xmlns="urn:BookLovers.org:LibroInfo">
    <TITULO>El Amor es dificil</TITULO>
    <PRECIO moneda="US Dollar">22.95</PRECIO>
</LIBRO>
```

2.3.3 ¿POR QUÉ USAR NAMESPACES?

El atractivo del XML consiste en la habilidad de inventar etiquetas que lleven información con significado. En el siguiente ejemplo, XML permite representar información acerca de un libro, de la siguiente manera:

```
<LIBRO>
    <TITULO>XML Guía para Desarrolladores</TITULO>
    <PRECIO moneda="US Dollar">44.95</PRECIO>
</LIBRO>
```

De igual forma, se puede representar información sobre el autor, así:

```
<AUTOR>
    <TITULO>Sr.</TITULO>
    <NOMBRE>Jose Luis Fierro</NOMBRE>
</AUTOR>
```

Aunque cualquier persona que lea el documento XML, puede distinguir entre las diferentes interpretaciones del elemento <*TITULO*>, un programa de computadora no tiene la capacidad de distinguirlos por separado. Sin información adicional, el programa no puede determinar que el primer elemento <*TITULO*> se refiere a la representación del título del libro, y que el segundo elemento se refiere a la representación del título del autor como "Sr." "Srta." "Sra." , Etc.

Los Namespaces solucionan este problema asociando un vocabulario (namespace) con un nombre de etiqueta. De esta forma, los elementos <*TITULO*> existentes en el documento pueden ser escritos de la siguiente forma:

<LibroInfo:TITULO xmlns:LibroInfo="libros-namespace-URI">XML Guía para Desarrolladores</LibroInfo:TITULO>

<AutInfo:TITULO xmlns:AutInfo="autor-namespace-URI">Sr.</AutInfo:TITULO>

El nombre que precede a los dos puntos, es llamado prefijo, el cual delinea a un XML namespace, identificado por un Universal Resource Identifier (URI). El namespace asegura la unicidad global cuando se fusionan fuentes XML, mientras el prefijo asociado(un nombre que sustituye el URI del namespace) debe ser único solo en el alcance del contexto del documento. Con este código, no existen conflictos entre etiquetas y atributos. Esto permite al documento anterior contener juntos: la información del libro y el autor, sin confusión. De esta manera, el elemento <*TITULO*> se refiere al libro o al autor. Si un programa quiere desplegar el nombre del libro en una interfaz de usuario, podría usar el modelo objeto para identificar el elemento <*TITULO*> del namespace "LibroInfo".

CAPÍTULO III

3 DTDS Y DOCUMENTOS ESQUEMAS.

3.1 DTD (DOCUMENT TYPE DEFINITION)

3.1.1 INTRODUCCIÓN

Cualquier diseñador de HTML está totalmente despreocupado de las interioridades del sistema que hacen posible la creación de los archivos HTML (o HTM) cuando utiliza algún editor de los que genera el código de forma automática. Es suficiente con componer el documento de forma similar a como se hace con un programa de tratamiento de textos, activar la opción de "Guardar como HTML" y ya está. A partir de entonces, cualquiera que abra el archivo desde un navegador, verá dicho documento en su pantalla. Esto que a primera instancia parece ser tan sencillo, exige que se estén manejando varias herramientas internas al sistema.

Cuando se genera el documento en un editor HTML, el código HTML que se crea está basado en un DTD (Document Type Definition, definición de tipo de documento) interno, que es una definición de las normas que regulan la formación de las etiquetas de un lenguaje de marcas determinado, en este caso el HTML. Por lo tanto, en el DTD que integra cada editor de HTML se especifican todas las etiquetas existentes, sus atributos, sus valores, etc, haciendo posible que se vayan integrando en el código del documento únicamente de acuerdo con dichas normas.

En XML no existen DTDs predefinidos del sistema, por lo que es labor del diseñador especificar su propio DTD para cada tipo de documento XML.

Un documento bien formado, no necesariamente es un documento válido. Establecer una propia estructura pre-definida para los documentos XML, significa que se tiene total seguridad que los datos van a ser desplegados correctamente en el browser, o encontrarán los requerimientos de la aplicación que va a trabajar con los datos, por ejemplo: el analizador (parser) o la aplicación que procesan los datos, pueden comparar el documento XML, con su estructura definida y verificar que todo esté conforme. Esta estructura pre-definida es conocida como el DTD, y un documento XML que está apegado a su DTD es un documento válido.

3.1.2 CONCEPTO

Un DTD es una definición exacta de la gramática de un documento, con la misión de que se genere el código adecuado sin errores.

El DTD, abarca una especial colección de declaraciones Markup, las cuales pueden tomar la forma de: un subconjunto interno(en el texto XML), una entidad externa(un archivo), o ambas. Estas declaraciones son caracteres especiales, las cuales definen la estructura de un documento XML, simplemente señalando y especificando que elementos son requeridos u opcionales y también el orden en que deben estar anidados.

3.1.3 DTD INTERNO

Un DTD interno, es aquel cuyas especificaciones se encuentran dentro del documento XML que lo va ha utilizar. Ejemplo:

3.1.5 DEFINICIÓN DE UN DTD

La Declaración del Tipo de Documento (DOCTYPE) se define en el prólogo del documento XML, así:

```
<?xml versión="1.0"?>
<¡DOCTYPE Presidentes SYSTEM "Pres.dtd">
```

Estas líneas declaran al documento XML y declaran también que es de la clase Presidentes. La palabra clave SYSTEM identifica que el DTD es para uso privado de este documento. Además el procesador analizará el archivo "Pres.dtd" para verificar las reglas de especificación con las cuales debe cumplir los datos XML.

El DOCTYPE también puede estar definido de la siguiente forma:

```
<¡DOCTYPE root-element PUBLIC "nombre" "URI of DTD">
```

Esta forma es usada cuando el DTD ha sido hecho público (esto no es muy usual). El XML intentará usar el DTD por el nombre. Si el intento falla el XML usará el URI.

Una declaración pública completa, podría verse así:

```
<¡DOCTYPE Autoparts PUBLIC "//AutoCo//DTD//EN"
                           "http://xmlserver.AutoCo.com/dtd/Autoparts.dtd">
```

Un DTD externo usualmente es usado en un escenario XML común, donde diversos sistemas necesitan compartir los mismos datos XML. Podría ser usado en aplicaciones de clase de producción, en donde se separan conceptos y los componentes ayudan en la documentación y en el mantenimiento del sistema. Un DTD interno es encontrado frecuentemente en pequeños sistemas y en la fase de prueba y desarrollo.

Si un DTD externo y un interno son usados al mismo tiempo, el DTD interno es procesado primero, y su información toma precedencia sobre el DTD externo.

3.1.6 ESTRUCTURA BÁSICA DE UN DTD

La estructura básica de un DTD, corresponde a una definición top-down de varios elementos que se necesita encontrar en el documento XML. Esto significa que el DTD es procesado desde el tope hacia el fondo, y que un elemento padre debe ser definido antes que cualquier elemento hijo. Cada elemento aparece en una Declaración Elemento, siguiendo el siguiente formato:

<!ELEMENT nombre_elemento regla>

A continuación se puede observar un documento XML acompañado de su DTD:

```
<?xml versión="1.0"?>
<!DOCTYPE ARTICULO SYSTEM "ejemplo.dtd">
<ARTICULO>
  <ENCABEZADO>
     <TITULO_PRIN>titulo principal va aquí</TITULO_PRIN>
     <SUBTITULO>subtitulo va aquí</ SUBTITULO >
  </ ENCABEZADO >
  <AUTOR_INFO>
        <NAME>Alex Rosero</NAME>
        <TITULO>Reportero</TITULO>
  </AUTOR_INFO>
  <FECHA>Febrero 19, 2001</FECHA>
  <CUERPO>
     contenido de la historia va aquí
  </CUERPO>
</ARTICULO>
```

Es fácil mirar como el DTD está siendo usado para definir el documento XML. Se requiere llamar al archivo "ejemplo.dtd", para que pueda ser encontrado usando la

referencia en el DOCTYPE. El archivo "ejemplo.dtd", es decir el DTD como tal, contiene las siguientes declaraciones:

```
<!ELEMENT ARTICULO(ENCABEZADO,AUTOR_INFO,FECHA,CUERPO)>
  <!ELEMENT ENCABEZADO (TITULO_PRIN,SUBTITULO)>
      <!ELEMENT TITULO_PRIN (#PCDATA)>
      <!ELEMENT SUBTITULO (#PCDATA)>
  <!ELEMENT AUTOR_INFO (NAME, TITULO)>
      <!ELEMENT NAME (#PCDATA)>
      <!ELEMENT TITULO (#PCDATA)>
  <!ELEMENT FECHA (#PCDATA)>
  <!ELEMENT CUERPO (#PCDATA)>
```

El Elemento Documento *<ARTICULO>*, contiene 4 elementos hijos: <ENCABEZADO>, *<AUTOR_INFO>*, *<FECHA>* y *<CUERPO>*, definidos con un separador coma y dentro de paréntesis, así:

```
<!ELEMENT ARTICULO(ENCABEZADO,AUTOR_INFO,FECHA,CUERPO)>
```

Algunos de estos elementos tienen hijos de su propiedad, tal como el elemento *<ENCABEZADO>*, así:

```
<!ELEMENT ENCABEZADO (TITULO_PRIN,SUBTITULO)>
```

Los elementos XML que contienen datos XML, tienen una ligera variación en su declaración

```
<!ELEMENT TITULO_PRINCIPAL (#PCDATA)>
```

El término en paréntesis PCDATA no se refiere a un elemento hijo, pero analiza el dato existente, el cual es procesado por el parser XML, en contraste al término CDATA, el cual no lo es.

3.1.7 DECLARACIÓN DE ELEMENTOS

La forma general de una declaración de elemento, es la siguiente:

<!ELEMENT nombre_elemento regla >

Todo elemento que aparece en un documento XML válido, necesitará estar primero definido en un DTD. Las reglas para formar un nombre de elemento apropiado son simples, así:

- Un nombre de elemento debe empezar con una letra o con un subrayado.
- No debe empezar con la cadena xml en mayúsculas o minúsculas o cualquiera de las combinaciones.
- Puede tener cualquier número de letras, números, subrayados, espacios o puntos.
- Los dos puntos están reservados para experimentación con Namespaces, los cuales no fueron incluidos en la especificación original. Consecuentemente, los dos puntos son caracteres legales, pero no deben ser usados en nombres de elementos.

3.1.7.1 Regla ANY

La primera y más básica declaración de un elemento es la todo-propósito regla ANY. Un elemento que está definido usando esta regla, puede contener lo que sea (o nada) permitido por el DTD, en cualquier orden. La regla ANY permitirá otras etiquetas o datos generales carácter (o ambos) dentro del elemento. La declaración de:

<!ELEMENT Recipiente ANY>

podría resultar en el siguiente XML:

```
<Recipiente>basura</Recipiente>
```

o:

```
<Recipiente>
        <Pequeño_Recipiente>
                basura
        </Pequeño_Recipiente>
</Recipiente>
```

3.1.7.2 Regla EMPTY

La declaración EMPTY especifica un elemento XML que no debe contener ningún dato, su forma general es la siguiente:

```
<!ELEMENT nombre_elemento EMPTY>
```

Al declarar un elemento vacío, parecería que este será de poca utilidad en el documento, sin embargo, los elementos que son declarados EMPTY, todavía tienen la facultad de contener información en forma de atributos. También existen elementos sin atributos que son útiles, por ejemplo en HTML, se usa con frecuencia la etiqueta *
* sin atributos. Un ejemplo de una etiqueta sin contenido, pero con atributos podría ser *<HR>*; se puede usar este elemento como tal, o modificarlo con un atributo *size*:

```
<HR size="3">
```

3.1.7.3 Declaración Mixta

Esta declaración contiene una lista de declaraciones encerradas entre paréntesis y separadas por el operador pipe (|). Éste provee un conjunto de reglas alternativas para el elemento, su sintaxis es la siguiente:

```
<!ELEMENT nombre_elemento (ElementoA | #PCDATA)>
```

Una declaración mixta puede ser usada cuando una de las opciones tiene elementos hijos, pero la otra opción no, por ejemplo:

```
<!ELEMENT Padre (Hijo | #PCDATA)>
    <!ELEMENT Hijo (HijoA, HijoB)>
        <!ELEMENT HijoA (#PCDATA)>
        <!ELEMENT HijoB (#PCDATA)>
```

En este ejemplo, el padre podría tener la información directamente o tener un elemento hijo, <*Hijo*>. El cual podría contener 2 elementos <*HijoA*> e <*HijoB*>.

3.1.7.4 Declaración Múltiple

La declaración múltiple de elementos permite el anidamiento de los mismos. El orden es importante, y se debe usar comas entre los elementos listados. Ya que el DTD posee una definición top-down, no es necesario declarar un elemento anterior para incluirlo en la lista de elementos, así:

```
<!ELEMENT nombre_elemento  (ElementoA, ElementoB)>
```

En este ejemplo, los dos ElementoA y ElementoB son requeridos.

3.1.7.5 Regla #PCDATA

La regla PCDATA es aquella que sirve para definir datos carácter, y es una de las reglas más comunes. Los caracteres analizados son datos carácter marcados, los que contienen las etiquetas markup. El parser interpretará éstas etiquetas. Usamos la palabra clave CDATA simplemente para identificar datos carácter que no serán analizados.

En el siguiente ejemplo, la parte del código: <prueba> mostrar esto </prueba> será tratada como solo otra secuencia de cadena, y no como XML a ser analizado, así:

```
<EjemploDoc>
   <Dato1> El Dato1 sera analizado </Dato1>
   <Dato2>
      <![CDATA[<prueba> esto no sera analizado </prueba>]]>
   </Dato2>
</EjemploDoc>
```

3.1.7.6 Agrupamiento, Ocurrencias y Símbolos Elemento

Cuando se crea las reglas para la declaración de un elemento, hay varios símbolos u operadores que son usados. Los símbolos de ocurrencia son usados en múltiples declaraciones para favorecer la definición de la estructura del XML, como se muestra en la Tabla # 3.

Símbolo	Descripción	Ejemplo
Paréntesis	Cierra una secuencia de conjunto de alternativas.	(ElementoA | #PCDATA)
Coma	Separa elementos en secuencia – orden crítico.	(ElementoA | ElementB)
Pipe	Separa elementos en un conjunto de alternativas.	(ElementoA | #PCDATA)
Marca de Pregunta	Debe aparecer nunca o una vez (0,1)	ElementoA?
Asterisco	Debe aparecer nunca o muchas veces (0,1,2,3,...)	ElementoA*
Más	Debe aparecer una o más veces (1,2,3,...)	ElementoA+
(Ninguno)	Debe aparecer solo una vez (1)	ElementoA

Tabla # 3: Símbolos de Ocurrencia usados en XML

Ejemplo de símbolos de ocurrencia:

<!ELEMENT EMAIL (To+, From, CC*, Subject?, Body?)>

Esto declara, que el elemento EMAIL tiene:

- Un elemento requerido To, y además se puede enviar el e-mail a más de una persona.
- Un elemento requerido From (uno solo)
- Un elemento opcional CC, que puede tener una o más direcciones.
- Una línea opcional Subject
- Una sección opcional Body

3.1.8 DECLARACIÓN DE ATRIBUTOS

Los atributos contienen información que no es parte del contenido XML encerrado entre etiquetas. Por ejemplo: *<P>...</P>* delimita un párrafo de texto. Sin embargo, el atributo tal como *ALIGN* en <P>ALIGN="CENTER"</P>, define como el párrafo está desplegado. La sintaxis para la definición de atributos en un DTD es la siguiente:

<!ATTLIST ElementoObjetivo AttrNombre AttrTipo defaults>

Mientras los atributos podrían aparecer en cualquier lugar del DTD, resulta una buena idea mantenerlos en la misma área del elemento que están apuntando. Esto debe hacerse para facilidad de lectura y mantenimiento. En la Tabla # 4, se puede observar los diferentes tipos de atributos que puede tener un DTD.

Tipo	Descripción
CDATA	Solo datos carácter pueden ser usados, los cuales serán ignorados por el analizador XML.
ENTITY	Valor debe referirse a una entidad externa declarada en el DTD.
ENTITIES	Igual que el anterior pero varios valores separados por espacio en blanco.
ID	Identificador de elemento único
IDREF	Valor de un único ID de tipo de atributo
IDREFS	Igual que el anterior pero varios valores separados por espacio en blanco.
NMTOKEN	Un nombre válido de señal XML
NMTOKENS	Igual que el anterior pero varios valores separados por espacio en blanco.
NOTATION	Valor debe referirse a una declaración de notación en el DTD.
Ennumerated	Valor del atributo debe corresponder a uno de los valores incluidos.

Tabla # 4: Tipos de Atributos de un DTD

Se puede colocar los valores por defecto de la declaración usando uno de los siguientes cuatro valores, expuestos en la Tabla # 5:

Tipo	Descripción
#REQUIRED	El valor debe ser especificado.
#IMPLIED	El valor del atributo es opcional.
#FIXED value	El atributo debe tener el valor suministrado – no puede ser cambiado por el usuario.
Default	El valor suministrado es por defecto.

Tabla # 5: Valores por defecto en una declaración de atributos

Si se analiza el ejemplo anterior del elemento EMAIL:

<!ELEMENT EMAIL (To+, From, CC*, Subject?, Body?)>

Se determina que algo falta, por lo tanto *BBC* es aumentado al DTD, pero necesita características adicionales. Requiere un atributo *"hidden"* colocado en *'TRUE'*, para que el contenido del elemento no este descubierto para el recipiente email. Adicionalmente, los usuarios no deben estar permitidos de cambiar el valor a *'FALSE'*, de esta forma se declara la lista BBC:

```
<!ELEMENT EMAIL (To+, From, CC*, BBC*, Subject?, Body?)>
...
<!ELEMENT BBC (#PCDATA)>
<!ATTLIST BBC Hidden CDATA #FIXED "TRUE">
...
```

A continuación se señalan algunos ejemplos de otros atributos:

Por Defecto:

```
<!ATTLIST  Addresscountry CDATA "US">
<!ATTLIST  BoilingPoint scale CDATA 'C'>
```

De Enumeración:

```
<!ATTLIST Sexo_empleado (masculino | femenino) #REQUIRED>
<!ATTLIST EstadoCivil (soltero | casado | divorciado ) #REQUIRED>
```

Hay que recordar algunos puntos cuando se trabaja con atributos:

- Si un atributo es definido más de una vez, solo la primera definición es usada.
- Puesto que un dato de atributo no es analizado, usamos el CDATA(dato carácter) y no el PCDATA.

- Los atributos no son ordenados, contrastando con los elementos XML.

3.1.9 DECLARACIÓN DE ENTIDADES

Las declaraciones de entidades en el DTD permiten crear entradas de substitución para texto adjunto. Ayudaría pensar que son contenedores para contenido – similares a las macros de contenido. Mientras hay varias entidades predefinidas en XML, la declaración ENTITY permite definir tantas entidades adicionales como se requiera. Esta es la forma general de esta definición:

<!ENTITY nombre_entidad definicion_entidad>

El uso de una entidad es delimitado por el (&) y el (;). Las cinco entidades predefinidas se muestran en la Tabla # 6:

Entidad	Carácter
&amp;	&
&lt;	<
&gt;	>
&quot;	"
&apos;	'

Tabla # 6: Entidades predefinidas para un DTD.

3.1.9.1 Entidades Generales o Internas

La entidad general es una que sustituye un nombre con una cadena predefinida de reemplazo de caracteres, ésta toma la siguiente forma:

<!ENTITY nombre caracteres_sustitucion>

Los siguientes son ejemplos de entradas de simples sustituciones:

<!ENTITY copyright "&#xA9">

Esto asigna el hexadecimal 0xA9 a la expresión 'copyright'. Esta entidad podría ser usada en el siguiente código XML:

```
<copyrightNotice>
        &copyright; 2000 por Alfa, Inc.
</copyrightNotice>
```

Las entidades de caracteres de reemplazo pueden incluir entidades predefinidas, pero es importante no crear referencias circulares mientras se las hace. Por ejemplo:

<!ENTITY itemA "Puedo incluir otra entidad como esta: &itemB; !">

Esto es válido si el itemB (abajo) contiene solo la cadena itemA.

<!ENTITY itemB "incluir itemA aqui, esta bien">

Pero en el siguiente ejemplo el itemA es una referencia a la entidad y por consiguiente es inválida:

<!ENTITY itemB "incluir &itemA; aqui, esta mal">

3.1.9.2 Entidades Externas

Las Entidades Externas son una poderosa herramienta en el mundo del XML. Es posible importar dinámicamente datos de un URI externo hacia el documento XML. Dos ejemplos comunes son: el traer datos que frecuentemente cambian (tal como los datos del clima) desde un sitio remoto o conseguir una foto o un archivo binario compartido por un usuario local.

Para el primer ejemplo, se podría tener el siguiente código dentro del DTD:

```
<!ENTITY climaA SYSTEM "http://servidor1/London/datos_clima.xml">
<!ENTITY climaB SYSTEM "http://servidor2/NewYork/datos_clima.xml">
```

Y así es como sería usado en el documento XML:

```
<Reporte_del_Clima>
<Title>El Reporte Actual del Clima</Title>
    &climaA;
    &climaB;
</Reporte_del_Clima>
```

Los datos que están en cada "datos_clima.xml" son procesados por el analizador (parser) XML. Si el documento XML contiene muchos datos dinámicos de diversas fuentes, podría ser posible llevar este concepto preferentemente a casos más extremos. Puesto que el dato está siendo recuperado de fuentes externas, hay que estar alerta del ancho de banda y otros tópicos de escalabilidad cuando se usan estas técnicas.

Algunos datos(imágenes binarias, por ejemplo) no deben ser procesados por el analizador (parser). Para evitar que esto suceda, se debe especificar la notación NDATA, así:

```
<!ENTITY imagen1 SYSTEM "MapadeLondon.gif" NDATA GIF>
<!ENTITY imagen2 SYSTEM "MapadeNewYork.gif" NDATA GIF>
```

Las entidades imagen1 e imagen2, serán usadas en el documento XML, en un estado de no-análisis, típicamente como un valor del atributo del elemento:

```
<Reporte_del_Clima>
        <Titulo>El Reporte Actual del Clima</Titulo>
                <map src="imagen1;"/>
                &climaA;
                <map src="imagen2;"/>
                &climaB;
</ Reporte_del_Clima >
```

3.1.9.3 Entidades Parámetros

Las Entidades Parámetros son similares a las Entidades Generales, ya que trabajan esencialmente en la misma forma. Sin embargo, su propósito es restringir solo al DTD. Se puede usar estas entidades, en caso de existir texto repetitivo en el DTD. Las entidades deben ser declaradas antes de ser usadas. Sintácticamente lucen así:

<!ENTITY % nombre "caracteres_sustitucion">

Hay que notar que un espacio se requiere en ambos lados del signo %, ejemplo:

<!ENTITY % ng "NDATA GIF">

Pero cuando usamos la entidad parámetro, el espacio no sigue al signo %:

<!ENTITY imagen1 SYSTEM "Imagen1.gif" %ng;>

También hay que recordar que las referencias circulares no se pueden resolver y son inválidas.

3.1.10 OTRAS PALABRAS CLAVES DTD

Además para los elementos, entidades, y atributos descritos anteriormente hay varias palabras claves DTD, que no están en esas categorías, así:

3.1.10.1 IGNORE e INCLUDE

Existen dos directivas que pueden ser usadas para cambiar bloqueos de DTD conteniendo *on* u *off*. Esto puede ser muy útil, especialmente cuando se esté desarrollando y probando la estructura DTD. La sintaxis básica para esta directiva es:

```
<![ IGNORE [
        DTD bloque no procesado
]]>
```

El complemento de la directiva IGNORE es la directiva INCLUDE, así:

```
<![ INCLUDE [
        DTD bloque procesado
]]>
```

El bloque DTD dentro de la directiva debe ser una declaración completa o múltiples declaraciones. Por ejemplo, considerando el siguiente código:

```
<!ENTITY % doTest "INCLUDE">
...

<! [ %doTest; [
        <!ELEMENT AUTOR_INFO (NOMBRE, TITULO)>
                <!ELEMENT AUTOR(#PCDATA)>
                <!ELEMENT TITULO(#PCDATA)>
]]>
...
```

Se puede ver que solo cambiando el valor de la entidad de parámetro *doTest* a IGNORE, se puede controlar la inclusión de bloqueos múltiples de DTD desde un solo lugar, así:

```
<!ENTITY  %  doTest "IGNORE" >
```

Usando esto en el desarrollo de cualquier aplicación, es mucho más conveniente y fácil que comentar líneas del DTD.

3.1.10.2 Comentarios

La forma de comentar líneas en un DTD es similar a como se lo hace en HTML:

```
<!--Esto es un comentario-->
```

Si se requiere comentar varias líneas de un DTD, resultaría tedioso realizar este trabajo línea por línea, por lo que es recomendable usar las directivas INCLUDE e IGNORE.

Ejemplo usando comentarios y las directivas:

```
<!ENTITY % doTest "INCLUDE">
...

<! [ %doTest; [
        <!ELEMENT AUTOR_INFO (NOMBRE, TITULO)>
                <!--apellidos, nombres>
                <!ELEMENT NOMBRE(#PCDATA)>
                <!—titulo otorgado al autor por la compañia >
                <!ELEMENT TITULO(#PCDATA)>
]]>
```

3.1.11 COMBINACIÓN DE DTD EXTERNO E INTERNO

Un DTD puede abarcar juntos, un subconjunto interno así como también un subconjunto externo. Un DTD común que está compartido, es el punto de partida para

las reglas que el documento XML debe seguir. Cada documento XML puede, opcionalmente modificar esas reglas como sea necesario.

Un uso común de combinaciones de subconjuntos DTD es para adaptar algunas entidades XML, así:

```
<?xml versión="1.0"?>
    <!DOCTYPE diario SYSTEM "generico.dtd" [
        <!ENTITY compania "ABC, Inc">
    ]>
    <diario>
        <nombre_diario> &compania; </nombre_diario>
        ...
    </diario>
```

El orden de procesamiento es el siguiente:

- La definición interna de la entidad "ABC, Inc".
- El contenido del archivo "generico.dtd".

Dado que la definición interna es procesada primero, toma precedencia, y el valor de &compania; en el documento XML es el nombre local.

3.1.12 DTD YA DEFINIDO

Varios equipos de investigación han desarrollado sus propios DTDs, destinados a cubrir necesidades específicas y que les permita describir la estructura de sus documentos, como se muestra en la Tabla # 7.

DTD	Descripción
Channel Definition Format (CDF)	Define Canales en donde el servidor envía periódicamente información a los usuarios.
Chemical Markup Language (CML)	Describe fórmulas y datos químicos.
Genealogical Data in XML (GedML)	Proporciona un formato de descripción para datos genealógicos.
Mathematical Markup Language (MathML)	Describe datos matemáticos.
Open Software Description (OSD)	Describe paquetes software que permiten instalar de modo remoto software y componentes de una INTRANET o Internet.
Syncrhonized Multimedia (SMIL)	Permite definir presentaciones para la web con objetos multimedia sincronizados.
Resource Description Framework(RDF)	Describe recursos de todo tipo para su mejor catalogación, búsqueda y referencia.

Tabla # 7: DTDs definidos por equipos de investigación en todo el mundo

3.1.13 LIMITACIONES DE UN DTD

A pesar de que los DTDs, pueden llegar a ser de gran utilidad para ciertas declaraciones destinadas a la validación de un documento XML, éstos tienen varias limitaciones, que no les permite ser una herramienta totalmente útil y aplicable, así:

- Debido a que son escritos en un lenguaje diferente al XML, requieren una tecnología de analizadores (parsers) diferente. Esto crea un problema para los desarrolladores de herramientas XML, los cuales necesitarían codificar dos analizadores, uno para el XML y otro para el DTD, esto reduce la posibilidad de tener un software ágil.
- Carecen de soporte para namespaces; debido a que el mundo de la computación está mucho más centrado en XML, y la inter-conectividad de la información XML es cada día más penetrante, la importancia del conocimiento de namespaces irá hacia delante y los DTDs serán menos usables.
- Poseen un limitado soporte para especificaciones correspondientes al diseño orientado a objetos como herencias y sub-clases.
- No tienen conceptos de tipo de datos (otros más que PCDATA y CDATA), los cuales son usados para ayudar a definir el contenido de elementos XML específicos.

3.2 DOCUMENTOS ESQUEMAS (SCHEMAS)

3.2.1 INTRODUCCIÓN

Al igual que los DTDs, la función de los Documentos Esquemas es validar el documento XML utilizando declaraciones específicas para cada parte del mismo. La gran diferencia con los DTDs es que los Documentos Esquemas están escritos en el mismo lenguaje que el documento XML, es decir en lenguaje XML, lo cual permite a los desarrolladores, tener mayor facilidad y amplitud al momento de escribir las declaraciones que servirán para validar los documentos XML, sin necesidad de aprender y utilizar ninguna clase de código que no esté relacionado con lenguaje XML.

3.2.2 DEFINICIÓN

Un "Documento Esquema XML" es algo similar a un DTD, es decir, define qué elementos puede contener un documento XML, cómo están organizados, y qué atributos y de qué tipo pueden tener sus elementos.

Como los DTDs, el Documento Esquema es usado para describir el contenido y la estructura de los datos del documento XML. De igual manera, el Documento Esquema es usado para la validación del documento XML.

3.2.3 COMPONENTES DE UN DOCUMENTO ESQUEMA

Cuando se utiliza Documentos Esquemas, siempre se usa dos modelos de documento: el documento de instancia (el cual es el XML bien formado), y el Documento Esquema como tal. El uso del término "instancia" proviene del mundo orientado a objetos. El Documento Esquema describe una clase de documentos XML, y el documento XML es una instancia de esa clase. A continuación se puede ver un documento XML seguido del Documento Esquema que lo valida:

```
<?xml version="1.0"?>
<ARTICULO>
    <ENCABEZADO seccion="economia">
        <TITULO_PRIN>titulo principal va aquí</ TITULO_PRIN>
        <SUBTITULO>subtitulo va aquí</ SUBTITULO >
    </ENCABEZADO >
    <AUTOR_INFO>
        <NOMBRE>Alex Rosero</NOMBRE>
        <TITULO>Reportero</TITULO>
    </AUTOR_INFO>
    <FECHA>Febrero 19, 2001</FECHA>
    <CUERPO>
      contenido de la historia va aquí
    </CUERPO>
    <DATOS >
        <REVISION>2001-02-18</REVISION>
        <PALABRAS>1532 </PALABRAS>
        <ID>D-54</ID>
    </DATOS >
</ARTICULO>
```

Si se analiza el documento XML anterior, se puede notar diferentes capas. Los elementos pueden contener otros elementos, y anidados pueden tener capas de profundidad. Si un elemento contiene sub elementos o atributos, ellos son tipos complejos (como *<AUTOR_INFO>*), o cuando un elemento solo contiene un número de cadenas, es un tipo simple (como *<FECHA>*). Un diseñador de Documentos Esquemas puede también definir un tipo simple usando la declaración simpletype. Los atributos no pueden contener sub elementos u otros atributos, por lo tanto ellos siempre tienen tipos simples.

El ejemplo anterior, no tiene todavía la conexión requerida a un Documento Esquema. A continuación se define el Documento Esquema, el cual describe el documento de ejemplo:

```
<Schema targetNamespace="http://www.MiCompania.com/noticias">
        xmlns:noticias="http://www.MiCompania.com/noticias"
        xmlns:xsd   = "http://www.w3.org/2001/XMLSchema">

<element name = "ARTICULO" type= "noticias:ARTICULOType" />
<element name = "FECHA"       type= "xsd:date" />
<element name = "CUERPO"     type= "xsd:string" />

<complexType name= "ARTICULOType">
  <element name= "ENCABEZADO" type= "noticias:ENCABEZADOType"/>
  <element name= "AUTOR"            type= "noticias:AUTORType"/>
  <element  ref= "FECHA"            minOccurs="0"/>
  <element  name= "CUERPO"          type= "xsd:string"/>
  <element  name= "DATOS"           type= "xsd:DATOSType"/>
       <attribute   name= "seccion"    type = "xsd:string"/>
</complexType>


<complexType name= "ENCABEZADOType">
       <element    name= "TITULO_PRIN"      type= "xsd:string"/>
       <element    name= "SUBTITULO"        type= "xsd:string"/>
</complexType>

<complexType name= "AUTORType">
       <element    name= "NOMBRE"       type= "xsd:string"/>
       <element    name= "TITULO"       type= "xsd:string"/>
</complexType>

<complexType name= "DATOSType">
       <element    name= "REVISION"       type= "xsd:date"/>
       <element    name= "PALABRAS"       type= "xsd:decimal"/>
       <element    name= "ID"             type= "noticias:IDType"/>
</complexType>

<simpleType  name= "IDType"  base= "xsd:string">
       <pattern  value= "[A-Z]-d(2)" />
</simpleType>

</Schema>
```

Existen varios detalles cuando se observa un Documento Esquema a primera vista, y que se deben tomar en cuenta, así:

- Un Documento Esquema, es un documento XML bien formado. Esto se puede comprobar de la misma manera que se comprueba un documento XML normal.
- Los elementos en el Documento Esquema que tienen el prefijo ".xsd" están identificados como pertenecientes al XML Schema namespace por la declaración:

  xmlns:xsd = "http://www.w3.org/2001/XMLSchema">.

  Esta declaración no indica la visita al sitio www.w3.org para corroborar el namespace, sino que le indica al "Parser" el uso de este namespace, el cual ya esta incluido dentro de la estructura local del "Parser".
- Los elementos que tienen el prefijo "noticias": pertenecen al vocabulario del autor:

  xmlns:noticias = "http://www.MiCompania.com/noticias".
- Los elementos en el Documento Esquema están declarados y luego definidos.

3.2.4 DEFINICIÓN DE TIPOS COMPLEJOS

El contenedor básico de todos los elementos en un Documento Esquema es el tipo complejo. Estos tipos pueden tener múltiples elementos y pueden tener atributos. Los tipos simples por otro lado, no deben tener elementos contenidos o atributos.

Un nuevo tipo complejo está construido usando el elemento <*complexType*>. La mayoría de los tipos complejos, tendrán varios elementos y declaraciones de atributos, aunque no existe nada que prohíba escribir un <*complexType*> con un solo elemento.

El siguiente ejemplo declara que en el documento XML, la instancia del elemento que está definida por *ARTICULOType* tendrá cuatro elementos tipo string. También tiene un

elemento opcional <FECHA> identificado por el atributo *minOccurs* con un valor de cero. Este elemento es solo una referencia, así:

```
<complexType name= "ARTICULOType">
   <element  name="ENCABEZADO"  type="noticias:ENCABEZADOType"/>
   <element  name= "AUTOR"          type= "noticias:AUTORType" />
   <element  ref= "FECHA"           minOccurs="0" />
   <element  name= "CUERPO"         type= "xsd:string" />
   <element  name= "DATOS"          type= "xsd:statsType" />
      <attribute     name= "seccion"          type = "xsd:string" />
</complexType>
```

Se deduce que cuando un tipo complejo contiene múltiples tipos complejos así como también atributos, la estructura puede convertirse muy elaborada. El tipo elemental que vemos en el *<complexType>* está construido dentro del Schema XML Definition y puede ser usado para construir otros tipos.

3.2.4.1 Definición de Tipos Existentes

Frecuentemente se requiere usar una definición de tipo existente, que ya se ha declarado. Para hacer esto, se usa el atributo *ref=*. La advertencia al usar este atributo, es que la referencia debe ser un elemento global. Los elementos globales son aquellos que están declarados en el tope del documento, como un hijo directo del elemento raíz del Documento Esquema. El elemento *<FECHA>*, fue declarado de esta manera:

```
<element    name= "FECHA"    type="date" />
```

Este elemento también ha utilizado el elemento *minOccurs* (mínima ocurrencia) con el valor de 0, el cual define un elemento opcional. El único valor aceptado para este

atributo es 0 o 1. Otro elemento que se puede utilizar es el *maxOccurs*, el cual puede tomar el valor de 1 o n.

Los valores por defecto para estos dos atributos son diferentes, dependiendo de donde sean usados. En un elemento el valor por defecto de *minOccurs* es 1 y no existe valor por defecto para *maxOccurs*. El resultado de no especificar estos atributos, es que el elemento luego debe ocurrir exactamente una vez. En el caso de los atributos, el valor por defecto de *minOccurs* es 0, y el valor por defecto de *maxOccurs* es 1.

3.2.5 DEFINICIÓN DE TIPOS SIMPLES

Los Tipos Simples son básicos grupos de caracteres del Documento Esquema. Incluso pueden ser tipos que están construidos en el Documento Esquema. En la Tabla # 8, se muestra una lista de tipos simples que son usados con mayor frecuencia por los desarrolladores.

Tipo Simple	Tipo	Descripción /Ejemplo
String	Primitivo	"Esta es una cadena"
Bolean	Primitivo	true,false. 1, 0
Float	Primitivo	Punto flotante 32-bit de precisión sencilla.
Double	Primitivo	Punto flotante 64-bit de doble precisión.
Decimal	Primitivo	-43.21, 0, 123.4, 1500.00
Integer	Derivado	-123456, -1, 0, 1, 123456
Date	Derivado	1983-06-03, (June3rd,1983)
Time	Derivado	15:25:57.000

Tabla # 8: Algunos Tipos Simples, usados en un Documento Esquema

3.2.6 FACETAS

Las Facetas se utilizan cuando se requiere crear un nuevo tipo simple. Se puede usarlas para modificar un tipo base. Algunas de estas Facetas fueron desarrolladas en primera instancia para trabajar con datos numéricos, mientras otras trabajan mayormente sobre representaciones de datos tipo cadena; *minExclusive*, *maxExclusive* y sus equivalentes son aplicables para todos los tipos. Las Facetas *lenght*, *minLenght*, y *maxlenght* son usadas sobre una cadena y los tipos derivados de XML 1.0, mientras que *presicion* y *scale* son solo para tipos numéricos.

Algunos ejemplos simples de estas Facetas son:

```
<!-- Referido a un sistema de hospital -->
<simplcTypc namc= "cdadPacicntcs" basc:"doublc">
        <minInclusive value = '0' />
        <maxInclusive value = '150' />
</simpleType>
```

```
<!-- Referido a un sistema de seguridad -->
<simplcTypc namc= "Usuario" basc:"string">
        <minLenght value = '4' />
        <maxLenght value = '32' />
</simpleType>
```

3.2.6.1 La Faceta Pattern

Otro aspecto importante de las facetas es el uso de expresiones regulares para dar máscaras a los datos. Estas máscaras pueden ser usadas en la creación de números telefónicos, zip, o códigos postales que son non-integers (mezcla de números y letras,

por ejemplo). Para esto, se utilizan expresiones regulares, las cuales son símbolos taquigráficos para describir patrones de texto.

Un simple ejemplo de una faceta pattern es:

```
<simpleType name="clienteID" base:"string">
        <pattern value="[A-Z]-d(2)" />
</simpleType>
```

3.2.6.2 La Faceta Enumeración

Otra de las Facetas más usadas es la de Enumeración. Aunque su uso es más frecuente sobre datos carácter, también puede resultar muy útil en elementos numéricos. Puede ser usada sobre todos los tipos simples a excepción del boolean. En el siguiente ejemplo solo se acepta tipos de sexo válidos:

```
<simpleType name="sexo"  base:"string">
        <cnnumcration valuc = 'masculino'>
        <ennumeration value = 'femenino'>
</simpleType>
```

En un ejemplo numérico, se podría usar:

```
<simpleType name="ataque" base:"decimal" >
        <cnnumcration valuc ='1' />  <!-- por tierra -->
        <ennumeration value ='2' />  <!-- por mar -->
 </simpleType>
```

En todos estos casos, si el dato instancia en el documento XML no es un valid-Schema, de acuerdo a las reglas que están definidas, se generará un error.

3.2.7 ATRIBUTOS

En XML frecuentemente usamos atributos para aclarar algunos aspectos del elemento. Una simple comparación de usar atributos versus elementos, es que los atributos son desordenados mientras los elementos son ordenados. Por ejemplo:

El parser XML verá:

```
<ElementoA Valor1="x" Valor2="y" />
```

como equivalente a:

```
<ElementoA Valor2="y" Valor1="x" />
```

Sin embargo, para el parser XML, esto:

```
<ElementoA>
    <Valor1> x </Valor1>
    <Valor2> y </Valor2>
</ElementoA>
```

es muy diferente a esto:

```
<ElementoA>
    <Valor2> y </Valor2>
    <Valor1> x </Valor1>
</ElementoA>
```

Otra consideración es que los atributos no pueden contener elementos o tener atributos de sí mismos. Si se analiza el siguiente fragmento de código:

```
<peso unidades="lbs">24.5</peso>
```

Esto será encontrado (sin una referencia al atributo) en el Documento Esquema asociado como:

```
<element  name="peso" type="decimal" />
```

Sin embargo, la línea anterior define un elemento simple, y los elementos simples no pueden tener atributos. Por lo tanto, especificando el atributo en el Documento Esquema, requiere más texto porque todos los elementos que tienen atributos deben estar definidos como tipos complejos, así:

```
<element nombre="peso">
        <complexType base="decimal" derivedBy="extension">
                <attribute name="unidades" type="string" />
        </complexType>
</element>
```

Examinando este Documento Esquema, se puede ver como el atributo es añadido al elemento.

¿Qué se debe hacer cuando un elemento solo tiene atributos y no tiene contenido textual?. Se debe usar el siguiente ejemplo XML:

```
<element nombre="peso">
    <complexType contend="empty">
        <attribute name="unidades" type="xsd:string" />
        <attribute name="cantidad" type="xsd:decimal" />
    </complexType>
</element>
```

Se necesita usar el atributo contenido (*contend*) del tipo complejo para lograr el objetivo.

3.2.8 ATRIBUTO CONTENIDO

El <*complexType*> tiene, en su definición Schema el atributo contenido (*contend*). Éste describe el modelo contenido deseado, del elemento descrito. El valor por defecto de *contend* es "elementOnly". Esto determina que el <*complexType*> solo puede contener elementos y atributos y no contenido textual. La lista completa de los posibles valores del modelo contenido se muestra en la Tabla # 9.

Tipo	Descripción
ElementOnly	Puede contener solo elementos
Empty	No debe tener contenido
Mixed	Puede contener ambos, texto y elementos
TextOnly	Solo puede contener texto

Tabla # 9: Posibles valores del Atributo Contenido

A continuación se tiene un ejemplo utilizando el atributo contend:

```
<element name='Data23'>
        <complexType contend='mixed'>
            <element name='Cantidad'  type='positive-integer' />
            <element name='Lugar'  type='string' />
        </complexType>

        <complexType contend='empty'>
            <attribute name='unidades' type='string' />
            <attribute name='cantidad' type='decimal' />
         </complexType>
</element>
```

3.2.9 GRUPOS

En un tipo complejo puede haber y de hecho lo hay, muchos elementos. Todos los elementos que están listados en el Documento Esquema tienen que existir en la instancia del documento (al menos que *minOccurs* sea igual a 0).

Cuando se forma un elemento *<complexType>*, es posible imaginar el escenario que podría cambiar los datos de los elementos requeridos. Un ejemplo de esto es el caso de la internacionalización. Por ejemplo, se da el caso de tener en un país, un código zip que está representado en un formato string, mientras que en otro país el código zip se representa en formato numérico. ¿Por qué no ofrecer una elección a la instancia del documento?. Este tipo de agrupamiento se denomina modelo de disyunción, Ejemplo:

```
<complexType name='Data54'>
    <group order="choice">
      <element name='codigozip' type='decimal' />
      <element name='codigopostal' type='string' />
    </group>
</complexType>
```

En la instancia del documento, podrá aparecer cualquiera: 'codigozip' o 'codigopostal'. Existe una segunda opción denominada modelo de conjunción, se refiere a que cada elemento debe aparecer en la instancia del documento, pero pueden aparecer en cualquier orden. Esta es una opción muy útil, cuando intentando usar un Documento Esquema común para dos sistemas que usan los mismos archivos de datos, pero debido a algunas restricciones de diseño de cada sistema, no se posee el mismo esquema, ejemplo:

```
<complexType name='Negocio'>
    <group order="all">
        <element name='simbolo' type='string' />
        <element name='fecha' type='date' />
        <element name='precio' type='decimal' />
        <element name='cantidad' type='decimal' />
    </group>
</complexType>
```

Hay algunas restricciones cuando se utiliza "all", las cuales se debe tener en cuenta. La primera, es que solo elementos individuales pueden ser incluidos en all, los grupos anidados no están permitidos. La segunda es que, cada item debe tener minOccurs=1 y maxOccurs=1 aunque esto puede cambiar en la especificación final.

3.2.10 ANOTACIONES

Las anotaciones pueden aparecer en cualquier lugar del Documento Esquema. Típicamente una nota *<documentation>*, podría ser puesta en el tope del Documento Esquema y contener información de derechos de autor o cualquier otro dato que sea útil al lector del documento.

La especificación no proporciona las reglas para el proceso de esta información, y deja al parser Schema que las determine. Ejemplo:

```
<annotation>
    <documentation>
        Pontificia Universidad Catolica del Ecuador – Sede Ambato 2001
        Todos los derechos reservados
    </documentation>
</annotation>
```

3.2.11 RESÚMEN DE DTD Y DOCUMENTOS ESQUEMAS

Aunque los objetivos del DTD y el Documento Esquema son los mismos, se puede señalar con certeza que la tecnología DTD no alcanza el nivel del Documento Esquema en mucho aspectos.

Los Documentos Esquemas son escritos en XML, mientras que los DTDs están en un lenguaje completamente diferente, lo que significa que el programador tendrá mucho más que aprender. Los Documentos Esquemas son mucho más que solo otra manera para describir un documento XML. Mirando hacia las carencias de los DTDs, es fácil entender porque los Documentos Esquemas están ganando popularidad.

Una de las mayores carencias de los DTDs es el tipo de datos; esto ha sido llenado totalmente en el Documento Esquema por lo que puede ser considerado un lenguaje de descripción type-centric. Si se aprende XML, resultará más fácil aprender a desarrollar Documentos Esquemas, ya que ambos utilizan el mismo lenguaje.

CAPÍTULO IV

4 EL MODELO DE OBJETOS DEL DOCUMENTO (DOM)

4.1 DEFINICIÓN

El Modelo de Objetos del Documento (DOM) es una Interfaz de Programación de Aplicaciones (API) para documentos HTML y XML. Define la estructura lógica de los documentos y el modo en que se acceden y manipulan los mismos. XML se utiliza cada vez más como un medio para representar muchas clases diferentes de información que puede ser almacenada en sistemas diversos, y mucha de esta información se vería, en términos tradicionales, más como datos que como documentos. Sin embargo, XML presenta estos datos como documentos, y se puede usar el Modelo de Objetos del Documento para manipular dichos datos.

Con el Modelo de Objetos del Documento los programadores pueden construir documentos, navegar por su estructura, añadir, modificar o eliminar elementos y contenido.

Siendo una especificación del W3C (World Wide Web Consortium), uno de los principales objetivos del Modelo de Objetos del Documento es proporcionar una interfaz estándar de programación que pueda utilizarse en una amplia variedad de entornos y aplicaciones. El Modelo de Objetos del Documento se ha diseñado para ser utilizado en cualquier lenguaje de programación.

El Modelo de Objetos del Documento define un conjunto estándar de comandos que los intérpretes exponen para facilitarle el acceso al contenido de los documentos HTML y XML desde sus programas. Un analizador (parser) de XML compatible con el Modelo de Objetos del Documento, toma los datos de un documento XML y los expone, mediante un conjunto de objetos que se pueden programar.

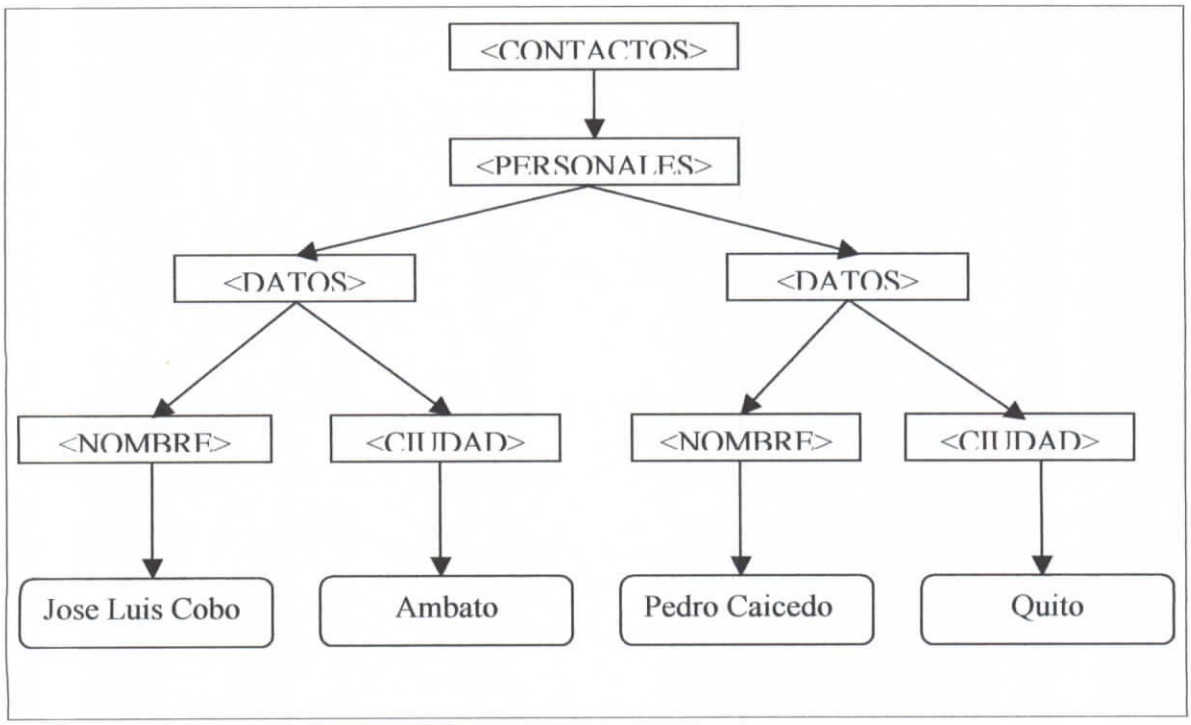
4.2 ORIGEN DEL MODELO DE OBJETOS DE DOCUMENTO

El Modelo de Objetos del Documento se originó como una especificación para permitir que los programas Java y los scripts de JavaScript fueran portables entre los navegadores web. El "HTML Dinámico" fue el ascendiente inmediato del Modelo de Objetos del Documento, y originalmente se pensaba en él principalmente en términos de navegadores. Sin embargo, cuando se formó el Grupo de Trabajo DOM (Document Object Model) en el W3C, también se unieron a él compañías de otros ámbitos, incluyendo los de la edición y archivo de documentos HTML y XML. Varias de estas compañías habían trabajado con SGML antes de que se hubiera desarrollado el XML. Algunas de estas compañías también habían desarrollado sus propios modelos de objetos para documentos a fin de proporcionar un API para los editores o los archivos de documentos SGML / XML, y estos modelos de objetos también han influido en el Modelo de Objetos del Documento.

4.3 NIVELES DEL MODELO DE OBJETOS DE DOCUMENTO

El W3C establece dos niveles de actuación, referidos al Modelo de Objetos del Documento, así:

El Modelo de Objetos del Documento representa este documento de la siguiente manera:



En el Modelo de Objetos del Documento, los documentos tienen una estructura lógica que es muy parecida a un árbol. Sin embargo, el Modelo de Objetos del Documento no especifica que los documentos deban ser implementados como un árbol, ni tampoco especifica cómo deben implementarse las relaciones entre objetos. El Modelo de Objetos del Documento es un modelo lógico que puede implementarse de cualquier manera que sea conveniente.

Una propiedad importante de los modelos de estructura del Modelo de Objetos del Documento es su isomorfismo estructural: si dos implementaciones cualesquiera del Modelo de Objetos de Documento se usan para crear una representación del mismo

documento, ambas crearán el mismo modelo de estructura, con exactamente los mismos objetos y relaciones.

Se denomina "Modelo de Objetos del Documento" porque es un "modelo de objetos" en el sentido tradicional del diseño orientado a objetos: los documentos se modelizan usando objetos, y el modelo comprende no solamente la estructura de un documento, sino también el comportamiento de un documento y de los objetos de los cuales se compone. Es decir, los nodos de un documento no representan una estructura de datos, sino que representan objetos, los cuales pueden tener funciones e identidad. Como modelo de objetos, el Modelo de Objetos del Documento identifica:

- Las interfaces y objetos usados para representar y manipular un documento.
- La semántica de estas interfaces y objetos, incluyendo comportamiento y atributos.
- Las relaciones y colaboraciones entre estas interfaces y objetos.

4.5 INTERCAMBIO DE DATOS CON PLATAFORMA NEUTRAL

XML es texto estructurado, y no existe un formado de datos más ampliamente soportado que el texto. Razón por la cual XML está ganando más popularidad. XML pasa por alto la incompatibilidad entre plataformas por lo que la especificación y el uso del Modelo de Objetos del Documento son apropiados por las siguientes razones:

- El intercambio de datos tiende ser relativamente compacto y bien definido.
- Las estructuras son bien definidas y tienden a repetirse, creando plantillas de documentos atractivas.

4.6 LO QUE EL MODELO DE OBJETOS DE DOCUMENTOS NO ES.

- El Modelo de Objetos del Documento no es una manera de ofrecer objetos persistentes para XML o HTML. En lugar de especificar cómo pueden representarse objetos en XML, el Modelo de Objetos del Documento especifica cómo se representan los documentos XML y HTML como objetos, de modo que puedan ser utilizados por programas orientados a objetos.

- El Modelo de Objetos del Documento no es un conjunto de estructuras de datos, es un modelo de objetos que especifica interfaces. Aunque los documentos XML contienen diagramas que muestran relaciones padre / hijo, éstas son relaciones lógicas definidas por las interfaces de programación, no representaciones de ninguna estructura interna de datos particular.

- El Modelo de Objetos del Documento no define "la semántica interna real" del XML o del HTML. La semántica de estos lenguajes está definida por las recomendaciones correspondientes del W3C. El Modelo de Objetos del Documento es un modelo de programación diseñado para respetar estas semánticas. El Modelo de Objetos del Documento no tiene ninguna consecuencia en el modo en que se escriben los documentos XML y HTML; cualquier documento que pueda escribirse con estos lenguajes puede ser representado en el Modelo de Objetos del Documento.

4.7 RECUPERACIÓN DE LA INFORMACIÓN DE UN DOCUMENTO

Para extraer la información que contiene un documento XML, se podría escribir código para analizar el contenido del dicho documento, pues no deja de ser un archivo de texto,

tal y como se lo podría hacer con HTML. Sin embargo, esta solución no es muy aconsejable y desaprovecharía una de las ventajas de XML: "el ser una forma estructurada de representar datos".

La mejor forma de recuperar información de archivos XML es utilizar un analizador (parser) de XML, que sea compatible con el Modelo de Objeto de Documento de XML. El Modelo de Objetos del Documento define un conjunto estándar de comandos que los analizadores devuelven para facilitar el acceso al contenido de los documentos XML desde sus programas. Un analizador de XML compatible con el Modelo de Objetos del Documento toma los datos de un documento XML y los expone mediante un conjunto de objetos que se pueden programar.

La Especificación del Modelo de Objeto de Documento (DOM) del W3C define actualmente lo que debería mostrar un Modelo de Objetos del Documento como propiedades, métodos y eventos.

4.8 EL ANALIZADOR (PARSER) MSXML

El componente MSXML es un analizador (parser) propuesto y desarrollado por Microsoft, el cual es compatible con el Modelo de Objetos de Documento (DOM), y brinda a los desarrolladores la posibilidad de utilizar la librería MSXML.dll, con el propósito de manejar diferentes interfaces relacionadas con la estructura misma del documento XML, a través de propiedades y métodos.

Además de permitir validar un documento XML que posee un DTD o un Documento Esquema (Schema), el analizador MSXML expone el documento XML en forma de un árbol de nodos. El árbol es iniciado con el más alto elemento en el documento (elemento raíz), cualquier otra construcción en el documento XML es representada como un nodo dentro del árbol.

El MSXML divide el tratamiento de nodos en 5 principales interfaces, así:

4.8.1 INTERFAZ DE DOCUMENTOS

Esta interfaz establece propiedades y métodos para el objeto documento en general, es la primera que actúa cuando un documento es cargado. A continuación se detallan algunas de las más importantes propiedades (Tabla # 10) y métodos (Tabla # 11) que se pueden utilizar con esta interfaz:

Propiedad	Significado
async	Indica si el documento debe ser cargado sincrónicamente o asincrónicamente.
childNodes	Una colección de objetos: Los inmediatos elementos hijos del documento: típicamente son: la declaración XML o el nodo raíz.
definition	La definición del elemento raíz en el DTD o Schema asociado (retorna la actual declaración XML para un Schema asociado).
documentElement	Retorna el elemento raíz del documento.
doctype	Retorna el nodo tipo de documento declarando el DTD.
parsed	Retorna true si todos los elementos hijos han sido analizados y false si el análisis gramatical ha quedado incompleto.
parseError	Un objeto de detección de errores, que retorna cualquier error detectado en la última operación de carga del documento.
readyState	Indica el estado actual del documento.
text	Retorna el texto contenido en todo el documento

Tabla # 10: Algunas propiedades de la Interfaz de Documentos

Métodos	Significado
appendChild (nuevo_hijo)	Añade un nuevo nodo hijo al final de la colección childNodes.
createElement (nombre)	Crea un nuevo elemento XML, cuyo nombre de nodo es el parámetro nombre.
createAttribute (nombre)	Crea un nuevo atributo con el nombre dado.
hasChildNodes	Retorna true si el nodo tiene hijos, false si no los tiene.
removeChild (nombre_hijo)	Remueve el nodo hijo señalado, si se encuentra en la colección childNodes
load (URL)	Carga y analiza la referencia de documento dada en el parámetro URL.

Tabla # 11: Algunos métodos de la Interfaz de Documentos

4.8.2 INTERFAZ DE NODOS

Esta interfaz abarca todos los varios tipos de nodos que un árbol del Modelo de Objetos del Documento puede soportar, tales como: elementos, atributos, secciones PCDATA, instrucciones de proceso, y otros. A continuación se detalla propiedades (Tabla # 12) y métodos (Tabla # 13) de esta interfaz.

Propiedad	Significado
firstchild	Retorna el primer hijo (el más a la izquierda) de los nodos que tienen hijos, caso contrario retorna NULL.
lastchild	Retorna el último hijo (el más a la derecha) de los nodos que tienen hijos, caso contrario retorna NULL.
nodeName	Retorna una cadena, para elementos, atributos y entidades, dicha cadena es el nombre calificado de un nodo.
nodeType	Retorna el tipo enumerado del nodo.
nodeTypedValue	Retorna el valor del tipo enumerado del nodo.
dataType	Para atributos, elementos y referencias de entidades retorna la representación en cadena (string) del tipo de datos especificado en el Schema.
nodevalue	Para atributos, comentarios, secciones CDATA y nodos texto, retorna el contenido textual del nodo.
parentNode	Retorna el nodo padre de los tipos que tienen padres.

Tabla # 12: Algunas propiedades de la Interfaz de Nodos

Método	Significado
cloneNode	Crea y retorna una copia del nodo.
hasChildNodes	Retorna true si el nodo tiene hijos, y false cuando el nodo no los tiene o es un tipo que no puede tener hijos.
insertBefore (nuevo, ref)	Inserta un nodo nuevo a la izquierda del nodo de referencia.
removeChild (hijo)	Remueve y retorna el nodo hijo especificado.
replaceChild (nuevo, viejo)	Reemplaza el nodo viejo con el nuevo y retorna el viejo.

Tabla # 13: Algunos métodos de la Interfaz de Nodos

4.8.3 INTERFAZ DE LISTAS DE NODOS

Esta Interfaz es usada para cualquier colección de nodos en un documento XML. En la práctica, las listas de nodos están separadas dentro de los atributos XML (en la colección de atributos) y todo lo demás en la colección de nodos hijos. Esta interfaz únicamente posee una sola propiedad (Tabla # 14) y tres métodos (Tabla # 15), así:

Propiedad	Significado
Lenght	Número de nodos en la colección nodeList.

Tabla # 14: Propiedad de la Interfaz de Listas de Nodos

Método	Significado
Item	Permite un acceso randómico a los nodos en la colección.
NextNode	Retorna el siguiente nodo en la colección, cuando se está iterando a través de la lista de nodos.
Reset	Restaura la iteración, cuando se usa el método nextNode.

Tabla # 15: Métodos de la Interfaz de Listas de Nodos

4.8.4 INTERFAZ DE MAPAS DE NODOS

Esta interfaz es similar a la interfaz Lista de Nodos, pero la diferencia se encuentra en el hecho de que los nodos son direccionados por el nombre, a pesar de su indexación ordinal. Otra importante diferencia, es que esta interfaz es usada únicamente por atributos, ya que los nombres de atributos serán únicos en cualquier instancia particular de un elemento, por lo tanto, el acceso por nombre es un método natural para direccionar atributos. Las propiedades (Tabla # 16) y métodos (Tabla # 17) de esta interfaz son las siguientes:

Propiedad	Significado
lenght	Número de nodos en el mapa de nodos.

Tabla # 16: Propiedad de la Interfaz de Mapas de Nodos

Método	Significado
getNamedItem (item)	Retorna el atributo nombrado.
nextNode	Retorna el siguiente nodo en la colección, o NULL si el final de la colección ha sido alcanzado.
reset	Restaura la iteración, cuando se está recorriendo la colección usando nextNode.
removeNamedItem (item)	Remueve y retorna el atributo nombrado si es encontrado, de otra forma este método retorna NULL.

Tabla # 17: Métodos de la Interfaz de Mapas de Nodos

4.8.5 INTERFAZ DE MANIPULACIÓN DE ERRORES

Esta Interfaz maneja los errores generados durante el análisis de un documento XML. Es una parte crítica dentro de la tarea de cargar un documento. Las propiedades de esta interfaz (Tabla # 18), son las siguientes:

Propiedad	Significado
errorCode	Código de error en decimal, del último error detectado. El valor del código de error es 0 cuando no existen errores.
filepos	Posición Absoluta de fila, donde el analizador detectó el error.
Line	Número de línea que contiene el error.
linepos	Posición del carácter dentro de la línea donde el error fue detectado.
reason	Descripción textual de la razón del error.
srctext	Todo el texto de la línea, que posee el error.
url	URL del documento que contiene el error.

Tabla # 18: Propiedades de la Interfaz de Manipulación de Errores

4.9 FUNCIONAMIENTO DEL MODELO DE OBJETOS DE DOCUMENTO.

Ya que el Modelo de Objetos del Documento soporta cualquier tipo de plataformas y lenguajes de programación, los desarrolladores de aplicaciones XML, tienen la capacidad de utilizar cualquier herramienta de software, la que más se acomode sus necesidades o con la que más estén familiarizados. De esta manera se puede encontrar un sinnúmero de ejemplos de utilización del Modelo de Objetos del Documento, desarrollados en una variedad de lenguajes como por ejemplo, Java, Visual Basic, C++, etc, así como también en ambientes ASP(Active Server Pages), utilizando JavaSripts o Visual Basic Scripts.

La utilización de un analizador (parser) compatible con el Modelo de Objetos de Documento (DOM) como el MSXML, abarca una extensa variedad de posibilidades de trabajo con el documento XML, gracias al sinnúmero de propiedades y métodos establecidos en cada interfaz, como se señaló anteriormente; no obstante, el trabajo más importante, dentro de un documento XML desde el Modelo de Objetos del Documento, se resumen en las siguientes tareas:

4.9.1 CREACIÓN DEL ANALIZADOR (PARSER)

La creación del analizador, se refiere a iniciar una instancia de la librería MSXML dentro de la aplicación que se esté desarrollando. Por ejemplo:

Si se está trabajando con Java script o Visual Basic script en ambientes ASP (Active Server Pages), se tiene:

```
//En Java script
var doc = new ActiveXObject("microsoft.xmldom");


'En Visual Basic script
set doc =  Server.CreateObject("microsoft.xmldom")
```

o si se está trabajando dentro de un lenguaje de programación como Visual Basic, se utiliza:

```
set doc =  CreateObject("microsoft.xmldom")
```

Si existe algún problema al momento de crear la instancia del analizador, el método createObject retornará Nulo.

4.9.2 CARGA Y VALIDACIÓN DEL DOCUMENTO

Para cargar un documento lo primero que se tiene que hacer, principalmente en ambientes ASP, es forzar al documento que se cargue de forma síncrona, para esto, la propiedad async de la instancia del documento, debe ser puesta en falso, y luego se procede a cargar el documento, valiéndose del método load, así: (código en JavaScript)

```
doc.async = false;
doc.load (Server.MapPath(archivo.xml));
```

Cuando load ha realizado la acción de carga del archivo por completo, también ha analizado el mismo. Es recomendable antes de realizar cualquier operación con el documento, verificar si el estado de la propiedad *readyState* se encuentra con el valor 4 y que el valor de la propiedad *errorCode* sea 0. Esto significa que el documento se ha cargado y que se han interpretado los datos del mismo, y además que no se ha detectado ningún error, así: (código en JavaScript)

```
If (doc.readystate = = 4  &&  doc.parseError.errorCode) = = 0)
{
  Response.Write ("El estado del documento es correcto");
}
else
  Response.Write("Se ha detectado un error");
```

Si un error ha sido localizado, se puede utilizar la propiedad doc.parseError.reason, con el propósito de saber cuál fue el motivo del fallo de validación del documento. La propiedad *readyState* devuelve uno de los cinco valores posibles enumerados en la Tabla # 19.

Estado	Valor
Uninitialized (sin inicializar): el método de carga no se ha iniciado.	0
Loading (cargando): mientras se ejecuta el método de carga.	1
Loaded (cargado): cuando ha terminado el método de carga.	2
Interactive (interactivo): hay suficiente DOM disponible para un examen de sólo lectura y los datos sólo se han interpretado parcialmente.	3
Completed (finalizado): ya se han cargado e interpretado los datos, que están disponibles para operaciones de lectura y escritura.	4

Tabla # 19: Posibles valores de la propiedad ReadyState

- **Solución de errores**

Un documento puede no cargarse por muchos motivos. Una causa corriente es que el nombre de documento pasado al método Load no sea válido. Otro motivo posible es que el propio documento no esté bien formado o que no sea válido.

De forma predeterminada, el intérprete MSXML valida el documento comparándolo con un DTD o con un Documento Esquema (Schema) si así se ha especificado en el documento. Un documento incorrecto puede hacer que el programa presente errores por muchos motivos. Por lo que es aconsejable proporcionar información acerca del error que ha ocurrido al momento de cargar el documento.

Independientemente del tipo de error, se puede recurrir al objeto *ParseError* (error de interpretación) para solicitar información sobre el error al intérprete. Se puede establecer una referencia a la interfaz *IXMLDOMParseError* del propio documento para trabajar con las propiedades del objeto *ParseError*.

El siguiente código de Visual Basic, muestra un cuadro de mensajes y toda la información disponible sobre el error procedente del objeto *ParseError*, utilizando las propiedades de la interfaz de tratamiento de errores:

```
Dim xDoc As MSXML.DOMDocument
Set xDoc = New MSXML.DOMDocument
If xDoc.Load("C:\Mis documentos\ejemplo.xml") Then
 ' El documento se ha cargado correctamente.
Else
 ' No se ha cargado el documento.
 ' Se debe identificar la causa, y detallar el error
 Dim Error_info As String
 Dim xPE As MSXML.IXMLDOMParseError
  ' Se Obtiene el objeto ParseError
 Set xPE = xDoc.parseError
 With xPE
  strErrText = "El Documento XML no se ha podido cargar" & _
     "debido al siguiente error" & vbCrLf & _
     "Error #: " & .errorCode & ": " & xPE.reason & _
     "Linea #: " & .Line & vbCrLf & _
     "Posicion de Linea: " & .linepos & vbCrLf & _
     "Posicion en Archivo: " & .filepos & vbCrLf & _
     "Texto Fuente: " & .srcText & vbCrLf & _
     "Documento URL: " & .url
 End With
 MsgBox Error_Info, vbExclamation
End If
Set xPE = Nothing
```

4.9.3 LECTURA A TRAVÉS DEL DOCUMENTO CARGADO.

El recorrer el documento XML cargado en memoria, es una de las más importantes tareas que se pueden realizar dentro de una aplicación. Para esto, se utiliza una rutina recursiva, que lista todos los nodos del documento, utilizando la propiedad *childNodes*.

A continuación se puede observar el código en Visual Basic, que inicializa y carga el documento y además la rutina recursiva que lista todos los nodos del documento:

```
' Inicialización y carga del documento
Public Sub Cargar_Documento()
  Dim xDoc As MSXML.DOMDocument
  Set xDoc = New MSXML.DOMDocument
  If xDoc.Load("C:\Mis documentos\ejemplo.xml") Then
    ' El documento se ha cargado correctamente.
    DesplegarNodos   xDoc.childNodes, 0
  Else
   ' No se ha cargado el documento.
   ' Ha ocurrido algún error
  End If
End Sub

' Esta es la rutina para recorrer el documento
Public Sub DesplegarNodos(ByRef Nodos As MSXML.IXMLDOMNodeList, _
  ByVal sangria As Integer)
  Dim xNodo As MSXML.IXMLDOMNode
  sangria = sangria + 2
  For Each xNodo In Nodos
    If xNodo.nodeType = NODE_TEXT Then
      Debug.Print Space$(sangria) & xNodo.parentNode.nodeName & _
      ":" & xNodo.nodeValue
    End If
    If xNodo.hasChildNodes Then
      DesplegarNodo xNodo.childNodes, sangria
   End If
  Next xNodo
End Sub
```

La rutina *Cargar_Documento*, abre un documento XML. A continuación, *Cargar_Documento* llama a otra rutina, *DesplegarNodo*, que es la que recorre el documento en realidad. *CargarDocumento* pasa una referencia a la propiedad *ChildNodes* del documento XML abierto, en la forma de un parámetro y un valor entero que especifica dónde comenzar el nivel con sangría. El código emplea el parámetro *sangria* para dar formato a la presentación del texto en la *ventana inmediato* de la estructura del documento en Visual Basic.

La función *DesplegarNodo*, recorre el documento buscando sólo nodos del tipo *NODE_TEXT*. Cuando el código encuentra un nodo del tipo *NODE_TEXT*, recupera el texto del nodo mediante la propiedad *NodeValue* (valor del nodo). Además, la propiedad *ParentNode* del nodo actual sirve para obtener una referencia retrospectiva a un nodo del tipo *NODE_ELEMENT*. Los nodos del tipo *NODE_ELEMENT* muestran una propiedad *NodeName* (nombre del nodo). Se muestra el contenido de *NodeName* y *NodeValue*.

Si un nodo tiene nodos secundarios, lo cual se determina revisando la propiedad *HasChildNodes* (tiene nodos hijos), la rutina *DesplegarNodo* se llama a sí misma recursivamente hasta llegar al final del documento.

La rutina *DisplayNode* escribe la información en la *ventana inmediato* de Visual Basic mediante Debug.Print (depurar e imprimir).

4.9.4 CREACIÓN Y MANIPULACIÓN DE DOCUMENTOS Y SU CONTENIDO.

El Modelo de Objetos de Documento (DOM), además de permitir al desarrollador realizar las tareas citadas anteriormente, brinda la posibilidad de realizar otras funciones relacionadas con el tratamiento de documentos XML, tales como: crear documentos, elementos y atributos, copiar anidamientos de un árbol del documento, y añadir instrucciones de procesamiento. A continuación se detalla el desarrollo de estas tareas, a través de un documento ASP (Active Server Pages), en el cual se utiliza Java Script, y tomando como referencia práctica, un Menú de Restaurante, así:

Nuevos Documentos.

Una vez creada una instancia de un objeto XML, se puede declarar un elemento raíz, el cual indica que se está empezando a crear un documento XML, así:

```
// crea la instancia del objeto XML
  var doc = new ActiveXObject("Microsoft.XMLDOM");

// Crea el elemento documento (elemento raíz) y lo conecta al objeto documento
 raiz = doc.createElement("Menu");
 doc.documentElement = raiz;
```

De esta forma, se empieza a crear un documento XML, el cual tiene como elemento raíz a *<Menu>*, que es el que abarcará a todos los demás elementos hijos que se creen posteriormente.

Nuevos Elementos.

Una vez creado el elemento raíz *<Menu>*, se pueden crear elementos hijos de este; para esto, se utiliza la propiedad *appendChild* del objeto "raíz", y la propiedad *createElement* del objeto "documento", así:

```
// Crea dos elementos hijos de Menu
raiz.appendChild(doc.createElement("Aperitivos"));
raiz.appendChild(doc.createElement("Postres"));

// Crea elementos hijos de Aperitivos
nodoAperitivos = doc.selectSingleNode ("/Menu/Aperitivos")
if (nodoAperitivos ! = null)
{
  itemNode = doc.createElement("Item")
  itemNode.appendChild(doc.createElement("Nombre"));
  itemNode.appendChild(doc.createElement("Precio"));
  itemNode.appendChild(doc.createElement("Descripcion"));

  // A continuación se establece los valores de cada nodo hijo
  itemNode.childnodes(0).text = "Hongos fritos con ensalada"
  itemNode.childnodes(1).text = "6.00"
  itemNode.childnodes(2).text = "Hongos franceses, frescos y con una jugosa ensalada"

  nodoAperitivos.appendChild(itemNode)
}

// Crea elementos hijos de Postres
nodoPostres = doc.selectSingleNode ("/Menu/Postres")
if (nodoPostres ! = null)
{
   itemNode = doc.createElement("Item")
   itemNode.appendChild(doc.createElement("Nombre"));
   itemNode.appendChild(doc.createElement("Precio"));
   itemNode.appendChild(doc.createElement("Descripcion"));

   // A continuación se establece los valores de cada nodo hijo
  itemNode.childnodes(0).text = "Pastel de Manzana"
  itemNode.childnodes(1).text = "7.00"
  itemNode.childnodes(2).text = "Delicioso pastel de manzanas chilenas"

  nodoAperitivos.appendChild(itemNode)
}
```

Copias de Anidamientos.

Un anidamiento comprende un elemento y sus descendientes, y si se realiza una clonación de dicho anidamiento, éste se repetirá a través del documento, cuantas veces se quiera. Para esto, se utiliza el método *cloneNode*, así:

En el ejemplo anterior, para crear los hijos del nodo *<Postre>*, se puede tener el siguiente código:

```
// Crea elementos hijos de Postres
nodoPostres = doc.selectSingleNode ("/Menu/Postres")
if (nodoPostres ! = null)
{
  // Copia los elementos hijos de su nodo hermano "Aperitivos"
  cloneNode = itemNode.cloneNode(true);

  // Establece los valores de sus nodos hijos
  cloneNode.childnodes(0).text = "Pastel de Manzana"
  cloneNode.childnodes(1).text = "7.00"
  cloneNode.childnodes(2).text = "Delicioso pastel de manzanas chilenas"

  nodoAperitivos.appendChild(itemNode)
}
```

Nuevos Atributos.

Se puede añadir atributos a un elemento, usando el siguiente código:

```
// Crea la instancia del objeto XML
var doc = new ActiveXObject("Microsoft.XMLDOM");

// Crea el elemento documento (raiz) y lo conecta al objeto documento
raiz = doc.createElement("Menu");
doc.documentElement = raiz;

// Añade 2 atributos del elemento raíz Menu: fecha_inicio y fecha_expiración

atributoNodo = doc.createAttribute("fecha_inicio");
atributoNodo.nodeValue = "2000-04-01";
raiz.attributes.setNamedItem(atributoNodo);

atributoNodo = doc.createAttribute("fecha_expiracion");
atributoNodo.nodeValue = "2000-06-30";
raiz.attributes.setNamedItem(atributoNodo);
```

En el código anterior, se está creando dos atributos del elemento raíz *<Menu>*, el atributo *fecha_inicio*, con el valor "2000-04-01", y el atributo *fecha_expiracion*, con el valor "2000-06-30".

Añadiendo Instrucciones de Procesamiento.

Todo documento XML necesita tener la instrucción de procesamiento, que se refiere a la declaración XML, la cual debe estar colocada al principio del documento. Se puede insertar dicha instrucción en un documento, con el siguiente código:

```
instruccion  = doc.createProcessingInstruction ("xml",  "version='1.0' ");
doc.insertBefore (instruccion, doc.childNodes(0));
```

Resultados

Con lo hecho anteriormente, se ha creado el siguiente documento XML:

```
<?xml version="1.0"?>
<Menu fecha_inicio="2000-04-01" fecha_expiracion="2000-06-30">
   <Aperitivos>
    <Item>
     <Nombre>Hongos fritos con ensalda</Nombre>
     <Precio>6.00</Precio>
     <Descripcion>Hongos franceses, frescos y con una jugosa ensalada</Descripcion>
    </Item>
   </Aperitivos>
   <Postres>
    <Item>
      <Nombre>Pastel de Manzanas</Name>
      <Precio>7.00</Price>
      <Descripcion>Delicioso pastel de manzanas chilenas</Description>
    </Item>
  </Postres>
</Menu>
```

4.9.5 RETORNO DEL DOCUMENTO A UN USUARIO CLIENTE

Una vez creado el documento, como se ha hecho anteriormente, resulta muy útil ponerlo a disposición de un usuario cliente, desde el servidor de páginas Web, con el propósito de poder ver el contenido del documento XML (en este caso el Menú del Restaurante) en el browser. Esta tarea se puede realizar utilizando la propiedad del documento XML con un objeto de respuesta ASP, así:

```
Response.Write(menuDoc.xml)
```

Esta línea hace que el analizador (parser) ponga el documento dentro de una cadena y la ejecute con el método *Write*. De esta manera, la información encerrada dentro del documento XML, será desplegada en el browser del usuario cliente. Cabe señalar que esta línea, debe ser colocada al final de todo el proceso visto anteriormente.

CAPÍTULO V

5 FORMATO Y TECNOLOGÍAS AFINES A XML

5.1 HOJAS DE ESTILO EN CASCADA (CSS - CASCADE STYLESHEET)

5.1.1 INTRODUCCIÓN

Los datos (información) que envuelve un documento XML, no pueden ser vistos con un estilo específico, directamente en cualquier browser o en otro medio de visualización de datos. Esta información necesita tener un estilo o una forma de presentación (formato), la cual va ha ser desplegada usuario final.

Al abrir un documento con extensión ".xml", desde el browser (IE 5.0, por ejemplo), este únicamente da una visión del documento en sí, señalando los diferentes nodos existentes en él. En definitiva, se necesita la ayuda de una tecnología relacionada con XML, que permita establecer estilos para los mismos, ya sea en forma general (para todo el documento), o en forma específica (algunos nodos del documento). Las Hojas de Estilo en Cascada (CSS), permiten llevar a cabo esta tarea, corroborado los conceptos de separación de: datos y presentación, en los documentos XML.

Los conceptos de utilización de Hojas de Estilo en Cascada, también son utilizados, aunque en menor grado por documentos HTML, con el propósito de darles mayor consistencia y manejo en cuanto se refiere al aspecto de la presentación. Sin embargo la utilización de estilos en HTML está limitada, ya que únicamente se puede trabajar y dar formato a las etiquetas predefinidas del HTML, en cambio en XML trabajan para todas las etiquetas que han sido definidas en el documento.

5.1.2 DEFINICIÓN

Las Hojas de Estilo en Cascada (CSS), son una recomendación del W3C que permiten controlar la presentación de los datos de documentos XML y HTML. Es una tecnología que puede ser usada en varias plataformas, pero su uso mas general es en Internet.

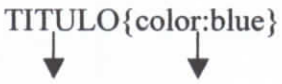
Esta herramienta permite establecer diferentes tipos de estilos para un solo documento XML, de esta manera se tiene múltiples presentaciones para un único documento, lo cual resulta muy útil al momento de configurar presentaciones para diferentes tipos de usuarios o necesidades.

Una Hoja de Estilos en Cascada se expresa en forma general, mediante reglas en un fichero de texto. Cada regla contiene el nombre del elemento al que se aplica el estilo definido.

5.1.3 FUNDAMENTOS Y SINTAXIS DE LAS HOJAS DE ESTILO EN CASCADA

5.1.3.1 Selectores

Una regla de estilos, dentro de un documento ".css" (extensión de archivos de hojas de estilo en cascada), consta de dos partes: el selector y la cadena de estilos. El selector corresponde al nombre de un elemento del documento fuente XML y la cadena de estilos es la especificación del o de los estilos que se quiere dar al selector (etiqueta), esta cadena está encerrada entre llaves, por ejemplo:



Selector Cadena de estilos

En este simple ejemplo, se establece que el contenido del elemento *<TITULO>*, será desplegado en color azul.

La cadena de estilos, consta de un conjunto de pares de elementos separados por el carácter **";"** , formados por el nombre de una propiedad y un valor, separados por el carácter **":"**, ejemplo:

TITULO{display:block; font-family:serif; color:black;}

Aquí se determina que el contenido del elemento *<TITULO>*, será desplegado en un solo bloque, con el tipo de letra serif y que además será de color negro.

Otro uso muy útil, es la combinación de múltiples selectores, los cuales requieren el mismo estilo, así:

Si se tiene el siguiente fragmento de un documento XML:

```
<CALLE>Cevallos # 154</CALLE>
<CIUDAD>Ambato</CIUDAD>
<PAIS>Ecuador</PAIS>
<TFNO>852274</TFNO>
```

se puede agrupar de la siguiente manera, con el propósito de que todos los selectores señalados tengan el mismo estilo, así:

CALLE, CIUDAD, PAIS, TFNO (color:blue; font-family:serif)

Los selectores también pueden especificar múltiples elementos, con una determinada clase (CLASS) o un identificador (ID), esto se da cuando en el documento XML fuente, se tiene grupos de elementos encerrados con etiquetas del mismo nombre, como por ejemplo:

```
<BIBLIOTECA>
 <LIBRO>
  <TITULO>Don Quijote de la Mancha</TITULO>
  <CAMPO>Literatura</CAMPO>
  <PAGINAS>203</PAGINAS>
 </LIBRO>

 <LIBRO>
  <TITULO>Hamlet</TITULO>
  <CAMPO>Literatura</CAMPO>
  <PAGINAS>400</PAGINAS>
 </LIBRO>

 <LIBRO>
  <TITULO>Física Cuántica</TITULO>
  <CAMPO>Física</CAMPO>
  <PAGINAS>500</PAGINAS>
  </LIBRO>
</BIBLIOTECA>
```

Entonces, si se determina el mismo estilo para el elemento *<LIBRO>*, este estilo se aplicará a todos los elementos *<LIBRO>* encontrados en el documento XML, por lo tanto, se puede establecer clases para uno, varios o todos los elementos *<LIBRO>*, así:

```
<BIBLIOTECA>
 <LIBRO CLASS="LITERATURA">
   <TITULO>Don Quijote de la Mancha</TITULO>
   <CAMPO>Literatura</CAMPO>
   <PAGINAS>203</PAGINAS>
 </LIBRO>

 <LIBRO CLASS="LITERATURA">
  <TITULO>Hamlet</TITULO>
  <CAMPO>Literatura</CAMPO>
  <PAGINAS>400</PAGINAS>
 </LIBRO>

 <LIBRO CLASS="FISICA">
  <TITULO>Física Cuántica</TITULO>
  <CAMPO>Física</CAMPO>
  <PAGINAS>500</PAGINAS>
 </LIBRO>
</BIBLIOTECA>
```

De esta manera se puede establecer en la hoja de estilos, diseños diferentes para las diferentes clases del elemento *<LIBRO>*, así:

```
LIBRO.LITERATURA {color:blue}
LIBRO.FISICA {color:black}
```

Se puede dar el caso también de que se necesite, definir un estilo único a un elemento en especial, de una lista de elementos que poseen el mismo nombre, así:

```
<BIBLIOTECA>
 <LIBRO ID="ESPECIAL">
  <TITULO>Don Quijote de la Mancha</TITULO>
  <CAMPO>Literatura</CAMPO>
  <PAGINAS>203</PAGINAS>
 </LIBRO>

 <LIBRO>
  <TITULO>Hamlet</TITULO>
  <CAMPO>Literatura</CAMPO>
  <PAGINAS>400</PAGINAS>
 </LIBRO>

 <LIBRO ID="ESPECIAL">
  <TITULO>Física Cuántica</TITULO>
  <CAMPO>Física</CAMPO>
  <PAGINAS>500</PAGINAS>
 </LIBRO>
</BIBLIOTECA>
```

Por lo tanto, la hoja de estilos tendrá una variación en su definición, así:

```
LIBRO{color:black}
LIBRO#ESPECIAL {color:blue;text-align:center}
```

5.1.3.2 Selectores Especiales

También existe otro tipo de selectores, los cuales se basan en un patrón, que es un elemento buscado en el documento fuente XML y para el cual se aplican las propiedades señaladas. A continuación, en la Tabla # 20 se resume una lista de algunos de estos selectores, que pueden ser muy útiles al momento de dar estilo a determinados elementos de un documento XML.

Selector	Significado	Ejemplo
E F	Selector del elemento del tipo F descendiente de un elemento del tipo E.	LIBRO TITULO {propiedades}
E > F	Selectiona cualquier elemento F hijo del elemento E.	LIBRO > TITULO {propiedades}
E[atributo]	Selecciona todos los elementos E con el nombre de *atributo* especificado.	LIBRO[precio] {propiedades}
E[atributo="valor"]	Selecciona todos los elementos E con el *atributo* especificado e inicializado a *valor*.	LIBRO[precio="20"] {propiedades}
E + F	Selecciona todos los elementos F cuyo inmediato hermano es un elemento E.	TITULO + PAGINAS {propiedades}

Tabla # 20: Lista de Selectores Especiales, usados en CSS

5.1.3.3 Unidades utilizadas en las Hojas de Estilo en Cascada

Las Hojas de Estilo en Cascada brindan la posibilidad de trabajar con dos tipos de unidades: las relativas y las absolutas, así:

a) Unidades Relativas

A este tipo de unidades pertenece el **em,** un **em** es igual al tamaño por defecto del tipo de letra. La utilidad del em, radica en que esta unidad, trabaja con proporciones y no con tamaños; razón por la cual el tamaño de los datos en el browser, va a depender de la fuente establecida por defecto en esa máquina, por lo que el usuario tendrá la posibilidad de ajustar el tamaño de lo que ve, como él quiera.

Otras unidades relativas, usadas para trabajar con el tamaño de tipos de letras, dentro de una hoja de estilos CSS, son los valores de tamaño absoluto. Estos son:

xx-small, x-small, medium, large, x-large, xx-large.

b) Unidades Absolutas

Las unidades absolutas permitidas en CSS, son:

mm (milímetro), cm (centímetro), in (pulgada), pt (punto, 72 puntos = 1 pulgada), pc (pica, 1 pica = 12 puntos).

5.1.3.4 Algunas propiedades utilizadas en las Hojas de Estilo en Cascada

- **Propiedad *display***

Esta propiedad, determina que tipo de contenedor tendrán los datos de todo el documento o de ciertos elementos del mismo. Esta propiedad tiene tres posibles valores, como se puede ver en la Tabla # 21:

Propiedad	Descripción
block	Los elementos empiezan y terminan en una nueva línea.
inline	Los elementos no empiezan y no terminan en una nueva línea. Éstos son desplegados en la misma línea. Este es el valor por defecto.
none	Los elementos están ocultos. No hay espacio reservado para los elementos en la página.

Tabla # 21: Valores de la propiedad *display*

- **Propiedades *padding, margin y border***

Los elementos que utilizan la propiedad display con valor de block, son rodeados por una caja limitada por bordes. Alrededor de la caja están tres cintas separadoras, las cuales tienen una colección de propiedades que pueden configurarse usando CSS. Estas tres capas son *padding, margin y border*, las cuales tienen similares colecciones de propiedades, así:

Para **padding**:

padding-top, padding-right, padding-bottom, padding-left y padding.

Las cuatro primeras propiedades establecen el valor dado, en la dirección señalada, en cambio la quinta, configura el valor para todas las direcciones, ejemplos:

* LIBRO {display:block;**padding-left:2em**}
* LIBRO {display:block;**padding:3em**}

Para **margin**:

margin-top, margin–right, margin–bottom, margin–left y margin.

Ejemplos:

* LIBRO {display:block;padding-left:2em;**margin-left:2em**}
* LIBRO {display:block;padding:3em;**margin:4em**}

La propiedad **border**, tiene las mismas alternativas, sin embargo posee además 3 parámetros que son:

border: width style color

por ejemplo:

LIBRO {display:block;padding:3em;margin:4em**;border: 2px solid black**}

- **Propiedades Relacionadas con Fuentes**

Existen cinco propiedades utilizadas para el tratamiento de características referentes a fuentes (tipos de letra), que son:

font-family: El valor que puede tomar esta propiedad, es el nombre del tipo específico de fuente que se va a utilizar, el cual puede ser: *Serif*, *sans-serif*, *Monospace*, *Cursive* y *Fantasy.*

font-style: Esta propiedad determina, el estilo que tendrá la fuente. Los valores disponibles son *normal*, *italic* y *oblique*.

font-variant: Esta propiedad puede tomar dos valores: *normal* y *small-caps,* controla la apariencia con estas dos alternativas.

font-weight: Determina el grosor de la fuente. Los valores que puede tomar son *normal*, *bold*, *bolder*, *lighter*, y valores en el rango de *100... 900*.

font-size: Establece el tamaño de la fuente, el cual puede ser especificado como un tamaño normal (14pt por ejemplo) o mediante una palabra clave de unidades relativas o absolutas.

Ejemplo general de utilización de propiedades para fuentes:

```
TITULO {display:block;
        font-size:2em;
        font-family:cursive;
        font-style: italic;
        font-variant:normal;
        font-weight:bold }
```

Cabe señalar que el orden de las propiedades establecidas en el estilo, no importa.

- **Propiedades relacionadas con el Texto**

Este conjunto de propiedades trabajan la apariencia del texto. En este grupo hay 8 propiedades:

word-spacing: Permite expandir o contraer el espacio entre palabras. Puede tomar cualquier valor de longitud y el valor normal es 1em.

letter-spacing: Permite expandir o contraer el espacio entre letras. Puede tomar cualquier valor de longitud y el valor normal es 0.3em.

text-decoration: Puede tomar uno de estos cinco valores: *none*, *underline*, *overline*, *line-through* y *blink*.

vertical-align: Especifica la posición relativa de un elemento respecto a la línea de escritura. Los valores que puede tomar son *baseline*, *sub*, *super*, *top*, *text-top*, *middle*, *bottom* y *text-bottom*.

text-transform: Permite indicar que tipo de letra utilizar. Los valores permitidos son *capitalize*, *uppercase*, *lowercase* y *none*.

text-align: Indica la alineación del texto *left*, *right*, *center* y *justify*. Se aplica a elementos de bloque.

text-indent: Indica la indentación de la primera línea de los elementos de bloque. Los posibles valores son cualquier medida de longitud.

line-height: Indica la distancia entre una línea de escritura y la siguiente. Puede tomar cualquier valor absoluto de longitud o un porcentaje que indica la altura con respecto al alto de la letra.

Ejemplo utilizando algunas propiedades para manejar el texto:

```
TITULO {display:block;
        text-align:center;
        text-transform:uppercase;
        letter-spacing:1em}
```

5.1.3.5 Herencia

Esta característica de las Hojas de Estilo en Cascada, es la habilidad de tener un conjunto de estilo en una parte del documento, el cual es heredado por una parte inferior del mismo, ejemplo:

Si se tiene el siguiente fragmento XML:

```
<BIBLIOTECA>
  <LIBRO>
    <TITULO>Don Quijote de la Mancha</TITULO>
    <CAMPO>Literatura</CAMPO>
    <PAGINAS>203</PAGINAS>
  </LIBRO>
</BIBLIOTECA>
```

Y si se desea que todos los elementos se desplieguen con el mismo tipo de letra, la opción más idónea dentro del CSS, es la siguiente:

```
BIBLIOTECA {font-family:sans-serif}
```

De esta manera, todos los elementos hijos incluidos dentro del elemento raíz (o no raíz) *<BIBLIOTECA>*, heredarán las características de estilo que éste posee.

5.1.3.6 Cascada

Es la habilidad para tener más de un estilo influyente en la presentación de un documento o texto. A un mismo documento es posible adjuntar más de un fichero de estilos, y esto puede provocar que un mismo elemento posea varias reglas de estilo, posiblemente contradictorias entre sí, por lo que se determina un orden de aplicación de estas reglas, denominadas estilos en cascada, ejemplo:

Si se tiene el siguiente documento XML:

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

<?xml-stylesheet type="text/css" href="estilo_biblio1.css"?>
<?xml-stylesheet type="text/css" href="estilo_biblio2.css"?>

<BIBLIOTECA xmlns:HTML="http://www.w3.org/TR/REC-html40">
        <HTML:STYLE>
          CAMPO {font-style:italic;}
        </HTML:STYLE>

        <LIBRO>
         <TITULO>Don Quijote de la Mancha</TITULO>
         <CAMPO>Literatura</CAMPO>
         <PAGINAS>203</PAGINAS>
        </LIBRO>
</BIBLIOTECA>
```

En este documento XML, existen dos declaraciones que especifican hojas de estilo para el mismo: "estilo_bilio1.css" y "estilo_biblio2.css", las cuales se ven así:

* **estilo_biblio1.css**

```
TITULO {font-familiy:sans-serif; font-size:2em}
CAMPO {font-familiy:arial; font-size:1em;font-style:bold}
PAGINAS {font-familiy:arial; font-size:1em}
```

* **estilo_biblio2.css**

```
TITULO {font-size:3em}
```

Además se ha incluido una referencia namespace HTML, con el propósito de poder incrustar un estilo para el elemento <*CAMPO*> dentro del mismo documento XML, así:

```
<BIBLIOTECA xmlns:HTML="http://www.w3.org/TR/REC-html40">
```

5.3.6 PREDICADOS

En los ejemplos anteriormente señalados, únicamente se ha utilizado un eje y un nodo prueba en la estructura del XPath. Además de éstos, se puede añadir lo que se denomina predicado. Básicamente, los predicados son usados para restringir el conjunto de nodos seleccionados por un eje a aquellos que cumplen cierta condición. Es decir, éstos sirven de filtro de nodos, los cuales retornan valores booleanos, usando varios operadores en conjunción con funciones XPath. Los predicados van encerrados en corchetes.

Los operadores booleanos son: =, !=, <, >, <=, y >=. Los operadores lógicos son **and** y **or**. Los operadores matemáticos son: +, -, *, div y mod.

A continuación se detalla algunos ejemplos de predicados:

- child::capitulo**[4]**

Esta expresión selecciona únicamente el cuarto elemento <*capitulo*> de los hijos del nodo contexto.

- descendant::***[@tipo='no']**

Esta instrucción selecciona todos los atributos tipo de cualquier o de todos los elementos del nodo contexto, que tengan el valor igual a 'no'.

- //***[@num]**

Aquí se hace referencia a todos los elementos que tengan el nombre de atributo igual a *num* (// es igual a *decendant*).

- /Lista_Libros/Libro**[@numero]**

En este ejemplo, se está utilizando una expresión XPath anidada y a la vez específica, ya que se está determinando que se haga referencia a todos los elementos <*Libro*> hijos de <*Lista_Libros*> que tengan un atributo con el nombre de *numero*. En este caso, cada expresión XPath es separada por el slash (/).

5.3.7 FUNCIONES XPATH

Además de presentar las características que se señaló anteriormente, XPath posee la habilidad de utilizar algunas funciones que pueden servir de utilidad al momento de realizar alguna operación con diferentes nodos del elemento XML.

Ejemplos:

- descendant::***[sum(persona/@edad)]**

Esta instrucción XPath, determina el uso de la función **sum,** la cual retorna la suma de los valores del atributo edad de todos los elementos <*persona*>.

- child::articulo[parrafo**[position( ) = 4]**]

Aquí se utiliza la propiedad position ( ) para seleccionar el elemento hijo <*articulo*> del nodo contexto que tiene un cuarto elemento <*parrafo*>.

- child::persona**[start-with(@apellido, 'S')]**

En esta instrucción, se usa la función **start-with** con el propósito de retornar todos los elementos <*persona*> hijos del nodo contexto, que tienen el valor del atributo *apellido* iniciando con la letra 'S'.

5.4 XLINK (LENGUAJE DE ENLACE)

5.4.1 DEFINICIÓN

XLink es un lenguaje definido en términos de marcaje (lenguaje de marcas), que permite introducir enlaces en un documento XML, de modo que se pueda enlazar archivos entre sí.

Este lenguaje, al ser una extensión del XML, utiliza su misma sintaxis y estructura con el propósito de crear y describir enlaces unidireccionales (como los de HTML) o también enlaces con características más sofisticadas.

Un enlace XLink, es una relación explícita entre recursos o partes de esos recursos. El término recursos se refiere a cualquier unidad de información o servicio que sea accesible a través de una dirección, como por ejemplo: archivos, imágenes, documentos, programas, etc.

5.4.2 LIMITACIONES DE LOS ENLACES HTML

En HTML, la etiqueta <A>, es usada para describir un enlace a otra página Web, por ejemplo:

<A HREF="http://www.hotmail.com">Correo Hotmail</A>

Este enlace, señala que el texto Correo Hotmail, es usado para conectar al sitio www.hotmail.com. No obstante, en los enlaces HTML, se pueden encontrar algunas deficiencias, así:

- Los enlaces deben ser incrustados en el documento fuente actual. Esta clase de enlaces, carece de soporte para imágenes o para documentos de Office, por ejemplo. Resultaría mucho mejor poder definir enlaces externos hacia el archivo fuente, permitiendo de esta manera manejar los enlaces desde una central, como un archivo o una base de datos.

- Un enlace HTML, actúa en una sola dirección, esto quiere decir, que si se desea tener una conexión entre un archivo "X" y otro "Y" y viceversa, se debe declarar el enlace en los dos documentos. Lo más práctico sería el definir un solo enlace, que permita una conexión bidireccional, o sea que se declare la conexión en un solo documento.

- En HTML un enlace es activado por un click, e inmediatamente la página a la que se hace referencia en el mismo es desplegada en la ventana actual del browser o en una nueva ventana. Lo ideal sería establecer si el enlace debe ser activado automáticamente o esperar a la acción de un click, además si se debe abrir el enlace en la misma ventana o en otra.

Los enlaces en XML, a través de XLink, establecen mejoras a todo lo señalado anteriormente, ya que permiten:

- Aplicar enlaces entre más de dos recursos.

- Asociar meta datos con un enlace.

- Establecer enlaces, que se encuentran separados de los documentos fuentes.

5.4.3 DECLARACIÓN DEL NAMESPACE XLINK

El uso de atributos y elementos referidos a XLink, requiere de la declaración previa de su Namespace, así:

```
<miElemento  xmlns:xlink=http://www.w3.org/1999/xlink>
...
</miElemento>
```

En esta declaración, se establece el uso del prefijo *xlink* referido al namespace *xlink*, dentro del elemento *<miElemento>*.

5.4.4 ALGUNOS ATRIBUTOS XLINK

La recomendación XLink, provee atributos para su uso, a continuación se detallan los más importantes:

- **Atributo type**

Este atributo especifica el tipo de enlace, que puede ser simple o extendido, por ejemplo:

* Enlace simple:

<ENLACE xlink:**type**="simple">.....</ENLACE>

* Enlace Extendido

<ENLACE xlink:**type**="extended">.....</ENLACE>

- **Atributo href**

Este atributo especifica el URI(Universal Resource Identificator) del recurso al cual se está haciendo el enlace, por ejemplo:

<ENLACE xlink:type="simple" **href**="http://www.pucesa.edu">PUCESA</ENLACE>

- **Atributos de comportamiento**

Los atributos de comportamiento son dos: ***actuate*** y ***show.*** Estos especifican el comportamiento de un enlace cuando se navega sobre el mismo, ya sea en enlaces simples o en enlaces múltiples.

El atributo ***show***, puede tener los siguientes valores:

new - Una aplicación debe desplegar el recurso final (archivo o parte de él) en una nueva ventana.

replace - El recurso final debe ser desplegado en la misma ventana del recurso fuente.

embed - El recurso final debe ser desplegado en el lugar del recurso fuente

El atributo ***actuate***, puede tener los siguientes valores:

onLoad - El recurso es recuperado inmediatamente

onRequest - El recurso es recuperado cuando el usuario de un procesador XLink lo pide.

5.4.5 TIPOS DE ENLACES

Existen dos tipos de enlaces, el simple y el extendido:

5.4.5.1 Enlace Simple

Un enlace simple, es aquel que asocia exactamente dos recursos, uno local y uno remoto. Este tipo de enlace provee la misma funcionalidad de la etiqueta HTML <A>, por lo tanto brinda un enlace de una sola vía, envolviendo en este, un recurso fuente y un recurso de destino, ejemplo:

```
<hypervinculo
        xlink:type="simple"
        xlink:href="http://www.yahoo.com">
    Correo Yahoo al instante
</hypervinculo>
```

5.4.5.2 Enlace Extendido

Un enlace extendido, es aquel que asocia un arbitrario número de recursos. Estos pueden ser cualquier combinación: remotos o locales. Típicamente los elementos de enlace extendido están almacenados de forma separada a los recursos que ellos asocian. De esta manera, los enlaces extendidos son importantes para situaciones donde los recursos relacionados son de solo lectura o donde cuesta mucho modificar y actualizar los enlaces. La figura # 7, ilustra el enlace extendido desde un archivo externo, hacia diferentes recursos:

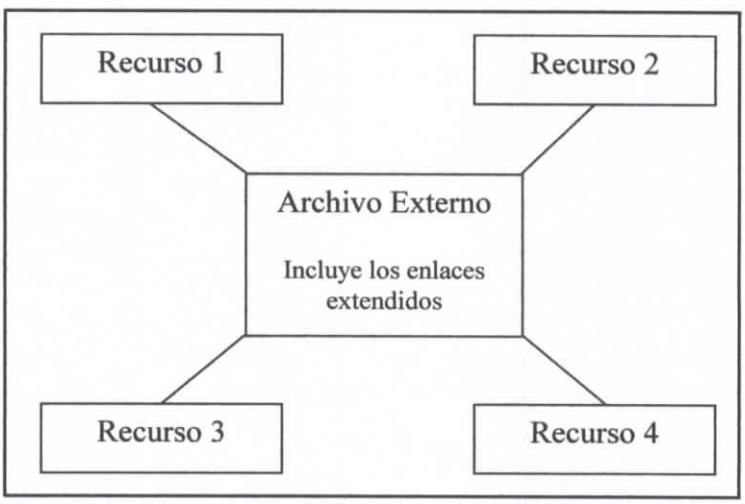


Figura # 7: Enlace extendido desde un archivo externo hacia diferentes recursos.

Los enlaces extendidos también contienen una combinación de los siguientes valores en para el atributo *type*:

- Xlink:type="**locator**"

Este valor del atributo type, indica que están tomando parte del enlace extendido, recursos remotos, ejemplo:

```
<ENLACES_DE_CORREO xlink:type="extended">
   <recurso1 xlink:type="locator"
             xlink:href=http://www.hotmail.com/>
   <recurso2 xlink:type="locator"
             xlink:href=http://www.yahoo.com/>
</ENLACES_DE_CORREO>
```

- xlink:type=**"arc"**

Este valor, especifica junto con otras reglas, la dirección de enlace de los recursos señalados en el enlace extendido.

El valor del atributo *xlink:**from*** define el recurso desde el cual se iniciará el enlace y el valor del atributo *xlink:**to*** indica el recurso al cual debe llegar, ejemplo:

```
<ENLACES xlink:type="extended">

   <recurso1 xlink:type="locator"
             xlink:href="http://www.mipagina.com/datos_personales.xml"
             xlink:label="personal"/>
   <recurso2 xlink:type="locator"
             xlink:href="http://www.mipagina.com/experiencia_laboral.xml"
             xlink:label="exp">

   <flujo_enlace  xlink:type="arc"
                  xlink:from="personal"
                  xlink:to="exp"/>

   <flujo_enlace  xlink:type="arc"
                  xlink:from="exp"
                  xlink:to="personal"/>

</ENLACES>
```

En este ejemplo, se define dos arcos, los cuales permiten ir en ambas direcciones, entre los datos personales y la experiencia laboral, utilizando el atributo **label**, cuyo valor ("exp", por ejemplo) da una referencia del enlace en cuestión.

La Recomendación definitiva XLink es reciente, ya que apenas salió publicada en Junio 27 del 2001, razón por la cual todavía están en desarrollo las herramientas que permitan la ejecución de enlaces, a través de un browser. No obstante, se presenta como un potente estándar de aplicación de diferente tipo de vínculos, que en poco tiempo, ayudará a que los documentos XML sean más útiles, potentes y flexibles en el mundo Web.

5.5 XPOINTER (LENGUAJE DE LOCALIZACIÓN)

5.5.1 DEFINICIÓN

XPointer, es un lenguaje que permite apuntar (localizar) a fragmentos específicos de un documento XML.

Ya que XLink permite enlaces a recursos enteros, este puede ser usado junto con XPointer y XPath, con el propósito de establecer enlaces a puntos específicos de dichos recursos.

XPointer soporta direccionamientos dentro de la estructura interna de los documentos XML. Permite examinar la estructura jerárquica y elegir partes internas de la misma,

basado en varias propiedades, tales como, tipos de elementos, valores de atributos, contenido textual, y posición relativa.

XPointer permite:

- Direccionar puntos y rangos, así como también nodos enteros.
- Localizar información a través de correspondencia de cadenas.
- Usar expresiones de direccionamiento en referencias URI (Universal Resource Identidifier) como identificadores de fragmentos.

5.5.2 APUNTADORES EN HTML

En HTML se apunta hacia cualquier parte del documento, a través de la etiqueta <A>, la cual requiere dos situaciones:

1. Definir el objetivo (area del documento html), así:

```
<HTML>
  <BODY>
    <A NAME="libro1">Introducción al XML</A>
    <A NAME="libro2">Estructura del XML</A>
  </BODY>
</HTML>
```

2. Crear el enlace en el documento fuente, así:

```
<HTML>
 <BODY>
    <A HREF="libros.htm#libro1">Capítulo No. 1</BR></A>
    <A HREF="libros.htm#libro2">Capítulo No. 2</BR></A>
 </BODY>
</HTML>
```

Este método utilizado en HTML, presenta algunos problemas. Primeramente se tiene que crear el objetivo para todo documento que se desee enlazar, lo cual resulta en más trabajo del necesario e incluso se tiene más de un problema, en el caso de que no se tenga acceso de escritura al documento. En segundo lugar, todo el documento debe ser cargado par acceder a una parte del mismo.

5.5.3 UTILIZACIÓN DE XPOINTER

XPointer extiende las utilidades de XPath, brindando sintaxis para declarar información de direcciones, en un enlace a un documento XML. Es decir, que XPointer puede utilizar cualquiera de las características XPath para efectos de localización. Además XPointer añade otras características (extensiones de XPath) denominadas puntos y rangos.

A continuación se detalla un ejemplo del uso de XPointer, utilizando un enlace URI a un documento XML, así:

*** Documento XML**

```
<Lista_Libros>
        <Libro id="001">
            <titulo>Don Quijote de la Mancha</titulo>
            <editorial>Planeta</editorial>
        </Libro>

        <Libro id="002">
            <titulo>HAMLET</titulo>
            <editorial>Planeta</editorial>
        </Libro>
                .....
                .....
</Lista_Libros>
```

*** Enlace**

<enlace xlink:type="locator"
xlink:href="http://www.libros.com/lista.xml**#xpointer(//Libro[.@id='001'])**"/>

En este enlace, se conjuga el uso de XLink, XPointer y XPath, ya que primeramente se realiza un enlace extendido de tipo "locator" con una referencia (href) URI, posteriormente se añade la sintaxis XPointer en la cual se utiliza una expresión XPath.

XPointer también permite el uso de alternativas de localización, así:

<enlace xlink:type="locator"
xlink:href="http://www.libros.com/lista.xml**#xpointer(//Libro[.@id='001'])**
xpointer(//Libro[.@id='002'])"/>

Este código señala que en caso de que no se localice el elemento *<Libro>* con valor del atributo ***id*** igual a **'001'**, se evaluará la siguiente búsqueda que es la de **xpointer(//Libro[.@id='002']).**

Como se señaló anteriormente, XPointer añade a la especificación XPath, los términos de localización referidos a puntos y rangos.

Una localización por puntos, puede ser un nodo así como también una localización definida por su contenido carácter, como por ejemplo, la primera letra del texto contenido en el elemento <titulo>, así:

#xpointer/Lista_Libros/Libro/titulo/**text()/point()[position()=1]**

Una localización por rangos, está definida por un punto de partida y un punto de finalización del rango, así:

#xpointer(<elemento_de_localización> to < elemento_de_localización >)

Por ejemplo:

Si se tiene el siguiente documento XML:

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

    <Libro id="001">
        <titulo>Literatura Universal</titulo>
        <editorial>Planeta</editorial>
        <autor>Jose Luis Arteaga</autor>
        <autor>Marco Villacis</autor>

    </Libro>

    <Libro id="002">
        <titulo>Castellano Ilustrado</titulo>
        <editorial>Planeta</editorial>
        <autor>Alex Wolf</autor>
        <autor>Gustavo Adolfo Becker</autor>
    </Libro>

</Lista_Libros>
```

Se puede realizar la siguiente localización por rango:

#xpointer(//Libro **to** autor[2])

En este ejemplo, primeramente se localiza los elementos *<Libro>*, y posteriormente se localiza los elementos *<autor>* con la segunda secuencia, por lo tanto, aplicando la referencia obtenida se obtendrá el siguiente resultado:

```
<Libro id="001">
    <autor>Marco Villacis</autor>

<Libro id="002">
    <autor>Gustavo Adolfo Becker</autor>
```

Al ser todavía un Work Draft (especificación en desarrollo), las aplicaciones XPointer están en condiciones de prueba y su completo desarrollo será un hecho en muy poco tiempo. No obstante, se perfila como una herramienta muy útil, junto con XLink, ya que estas dos brindan posibilidades que actualmente el HTML no puede abarcar dentro de su funcionalidad.

CAPÍTULO VI

6 HERRAMIENTAS RELACIONADAS CON XML

6.1 INTRODUCCIÓN

Debido a que el XML tiene relación con algunos estándares relativamente similares e interactúa con otros, como el HTML, es necesario realizar un breve estudio de éstos, para tomar en cuenta las virtudes, limitaciones, opciones, diferencias y similitudes existentes entre ellos.

6.2 HTML (HYPERTEXT MARKUP LANGUAGE)

6.2.1 DEFINICIÓN

HTML (HyperText Markup Language) es un lenguaje de marcas muy sencillo que permite describir hipertexto, es decir, texto presentado de forma estructurada y agradable, con enlaces (hyperlinks) que conducen a otros documentos o fuentes de información relacionadas y con inserciones multimedia (gráficos, sonido...). La descripción se basa en especificar en el texto la estructura lógica del contenido (títulos, párrafos de texto normal, enumeraciones, definiciones, etc) así como los diferentes efectos que se quieren dar (especificar los lugares del documento donde se debe poner cursiva, negrita, subrayado o un gráfico determinado) y dejar que luego la presentación final de dicho hipertexto se lleve a cabo, a través de programas especializados (como Internet Explorer, Netscape, Mosaic, etc..) llamados navegadores o browsers.

Como se ha señaló anteriormente, el HTML también se deriva del SGML y su creación y desarrollo se dirigieron a la unificación del contenido y la presentación en un mismo documento, además de buscar una manera fácil de desarrollar documentos para la Web.

6.2.2 ESTRUCTURA BÁSICA DE UN DOCUMENTO HTML.

Un documento HTML se compone de una estructura formada por directivas escritas en formato texto que están incluidas en un archivo, el cual puede ser hecho en cualquier programa editor de textos. Cualquier directiva utilizada en HTML estará encerrada entre dos etiquetas **"<"** y **">"** que distinguen a estas del texto normal; Siempre cada directiva tiene una directiva de inicio *<directiva_de_inicio>* y otra de finalización *</directiva_de_finalización>*, sin que esto sea una regla inquebrantable.

Un típico documento HTML está formado así:

Documento H.T.M.L.

Encabezamiento

Cuerpo

Fin del Documento H.T.M.L.

Ejemplo:

```
<HTML>
 <HEAD>
  Apariencia General del Documento H.T.M.L.
 </HEAD>

 <BODY>
   Texto del documento, menciones a gráficos, enlaces, etc.
 </BODY>
</HTML>
```

Un documento HTML comienza con la etiqueta **<HTML>**, y termina con **</HTML>**. Dentro del documento (entre las etiquetas de principio y fin de html), hay dos grupos bien diferenciados: el *encabezamiento*, delimitado por **<HEAD>** y **</HEAD>**, que sirve para definir diversos valores válidos en todo el documento; y el *cuerpo*, delimitado por **<BODY>** y **</BODY>**, donde reside la información del documento.

6.2.2.1 Encabezamiento <HEAD>

La utilidad más importante del encabezamiento es la directiva **<TITLE>**, que permite especificar el título de un documento HTML, el cual se despliega en la barra de títulos de la ventana del navegador que muestra la página. Por ejemplo, en el encabezamiento de una página Web cualquiera, se puede escribir la siguiente línea:

<TITLE>Página Web de la P.U.C.E.S.A**</TITLE>**

Dentro del encabezado se puede incluir otras directivas adicionales como la directiva **<META>,** la cual indica al navegador de Internet las palabras clave y contenido de una

página Web. Muchos de los buscadores de páginas Web de Internet (Yahoo, Lycos, Altavista, etc.) utilizan el contenido de esta directiva para incluir la página en sus bases de datos. La directiva **<META>** lleva generalmente dos parámetros, **name** y **content**.

Ejemplo :

<META name = "PUCESA" **content** = "Introducción, Enlaces, Correo">

Indica al navegador el nombre de la página y sus contenidos principales.

6.2.2.2 Cuerpo <BODY>

El cuerpo de un documento HTML contiene el texto que, con la presentación y las directivas que se decidan, se presentará ante el usuario. Dentro del cuerpo son aplicables un sinnúmero de directivas y efectos, los cuales establecen la manera en la que el contenido se va a presentar al usuario

El elemento **<Body>** tiene una serie de atributos que permiten cambiar la apariencia global del documento en general, como por ejemplo:

<Body background="imagen.jpg">

El atributo **background** indica el nombre de un archivo gráfico que servirá como "fondo" de la página.

<Body bgcolor="#0000FF">

El atributo **bgcolor** indica el color de fondo del documento. En este ejemplo, se especifica que el documento tendrá al negro como color de fondo, a través del código hexadecimal de color **"#0000FF".**

Dentro del cuerpo del documento **<Body>**, puede existir un sinnúmero de elementos que pertenecen exclusivamente al HTML, los cuales permiten establecer características propias al contenido que va ha ser desplegado por el browser. En la tabla # 27 se detallan algunos de los elementos que se encuentran en la última versión (4.0) formulada por el W3C:

Elemento	Descripción
A	Enlace de hipertexto o destino de un link
B	Texto en Negritas
BIG	Texto más grande
BR	Salto de línea
BUTTON	Botón
DIV	Contenedor Genérico a nivel de bloque
FORM	Formulario Interactivo
H1	Título de nivel 1
H2	Título de nivel 2
HR	Línea Horizontal
I	Texto en Itálicas
IMG	Imagen
INPUT	Entrada de Formularios
OPTGROUP	Grupo de Opciones
OPTION	Menú de Opciones
P	Párrafo
SCRIPT	Script ejecutado en la máquina cliente
SELECT	Selector de Opciones
STRONG	Énfasis fuerte
TABLE	Tabla

Tabla # 27: Descripción de algunos elementos HTML 4.0

Dentro de estas directivas, es necesario resaltar algunas de ellas, por la funcionalidad tan particular que brindan al desarrollador de páginas web, así:

Hipervínculos.

Una de las características principales de una página Web es que se puede incluir en ella Hipervínculos. Un Hipervínculo es un elemento de la página que hace que el navegador acceda a otro recurso, otra página Web, un archivo, etc.

Para incluir un Hipervínculo se utiliza la directiva **<A></A>**. El texto o imagen que se encuentre dentro de los límites de esta directiva será sensible, esto quiere decir que si pulsamos con el ratón sobre él, se realzará la función de hipervínculo indicada por la directiva **<A></A>**. Si el Hipervínculo esta indicado por un texto, este aparecerá subrayado y en distinto color, si se trata de una imagen, esta aparecerá con un borde rodeándola. Esta directiva tiene el parámetro **href** que indica el lugar a donde nos llevará el Hipervínculo si lo pulsamos.

Por ejemplo, si tenemos

<A href = "http://www.hotmail.com/"> Hotmail – Correo Gratuito**</A>**

En el navegador se verá como :

Hotmail – Correo Gratuito

Si se da un clic en este hipervínculo, automáticamente se irá hacia la página de Hotmail, que se encuentra detallada con la opción href.

Lo mismo se puede hacer con un gráfico:

<A href = "http://www.hotmail.com/" > **<IMG src** = "hotmail.gif"**></A>**

Pulsando sobre la imagen se accedería a la pagina situada en *http://www.hotmail.com/.*

Un Hipervínculo también puede llevar al usuario a una zona específica de la página. Para ello se debe marcar en la página las diferentes secciones en las que se divide. Esto se realiza a través del atributo **name**, ejemplo:

```
<A name = "seccion1"></A>
```

Esta instrucción marca el inicio de una sección dentro de la página. La sección se llamará ***seccion1***. Para hacer un enlace a esta sección dentro de la página lo haríamos de la siguiente forma:

```
<A href = "#seccion1">Primera Parte</A>
```

O también :

```
<A href = "http://www.jet.es/mipagina.htm#seccion1">Primera Parte</A>
```

Tablas

Las tablas permiten representar cualquier elemento de una página (texto, listas, imágenes, etc.) en diferentes filas y columnas separadas entre si. La tabla se define mediante la directiva **<TABLE></TABLE>.**

Para definir las celdas que componen la tabla se utilizan las directivas **<TD>** y **<TH>**. **<TD>** indica una celda normal, y **<TH>** indica una celda de "cabecera", es decir, el contenido será resaltado en negrita y en un tamaño ligeramente superior al normal. Los parámetros opcionales de ambas directivas son:

Para indicar que acaba una fila de celdas se utiliza la directiva **<TR>**. A continuación se detalla un ejemplo de una tabla que contiene solo texto. Como se indicó anteriormente el contenido de las celdas puede ser cualquier elemento de HTML, un texto, una imagen, un hipervínculo, etc.

```
<html>
<body>
 <TABLE border = 4 cellspacing = 4 cellpadding = 4 width =80%>
 <TH align = center>Buscadores
 <TH align = center colspan = 2>Descripción
 <TR>
 <TD align = LEFT>Altavista
 <TD align = LEFT>Un tanto lento, pero con gran interactividad
 </TR>
 <TR>
 <TD align = LEFT>Google
 <TD align = LEFT>Resultados rápidos y efectivos
 </TR>
</body>
</html>
```

Lo cual resulta en la siguiente tabla:

Buscadores	Descripción
Altavista	Un tanto lento, pero con gran interactividad
Google	Resultados rápidos y efectivos

Las directivas **<TD>** y **<TH>** deberían contar con sus respectivas etiquetas de cerrado, sin embargo los navegadores asumen que un elemento de la tabla, queda automáticamente "cerrado" cuando se "abre" el siguiente.

Formularios

Un formulario dentro de una página web, permite solicitar información al visitante y procesarla. En un formulario se puede solicitar diferentes datos (campos), cada uno de los cuales se representa a través de una caja de texto, una lista de opciones, una caja de verificación o un grupo de opciones, dependiendo de las necesidades del sitio.

La declaración del formulario se pone entre las directivas **<FORM></FORM>**. En el interior de la declaración se indican los elementos de entrada. La directiva **<FORM>** tiene los parámetros **action** y **method**.

action = "programa"

Indica el programa que va a "tratar" a las variables que se envíen con el formulario.

method = POST / GET

Indica el método según el que se transferirán las variables.

Para enviar la información como parte del encabezado HTTP (para poder extraerla utilizando la colección Forms del objeto Request del servidor), se establece el valor "POST". Para enviar la información como una cadena de búsqueda al archivo de proceso del formulario (para poder extraerla utilizando la colección QueryString del objeto Request), se establece el valor "GET".

Para la introducción de datos se utiliza la directiva **<INPUT>**. Esta directiva tiene el parámetro **type** que indica el tipo de elemento en el cual se va a introducir los datos y **name** que indica el nombre que se le dará al campo. Cada tipo de elemento tiene sus propios atributos, a continuación se detallan todos los tipos de elementos **<INPUT>,** que pueden estar dentro de un formulario **<FORM>** HTML:

<INPUT type= **text** name = nombre_campo>

Indica que el campo a introducir será un texto.

<INPUT type = **password** name = nombre_campo>

Indica que el campo será una clave. Mostrará asteriscos (*) en lugar de las letras escritas.

<INPUT type = **checkbox** name = nombre_campo>

El campo se elegirá marcando una casilla.

<INPUT type = **radio** name = nombre_campo>

El campo se elegirá marcando una casilla. Solo permite marcar una sola de las casillas.

<INPUT type = **hidden** name = nombre_campo>

El usuario no puede modificar su valor, ya que el campo no es visible se manda siempre con el valor indicado por el parámetro:

value = "valor"

<INPUT type = **submit**>

Representa un botón de envío de formulario. Al pulsar este botón la información de todos los campos se envía al programa que procesará los datos.

<INPUT type = **reset**>

Representa un botón que borra el contenido de todos los campos.

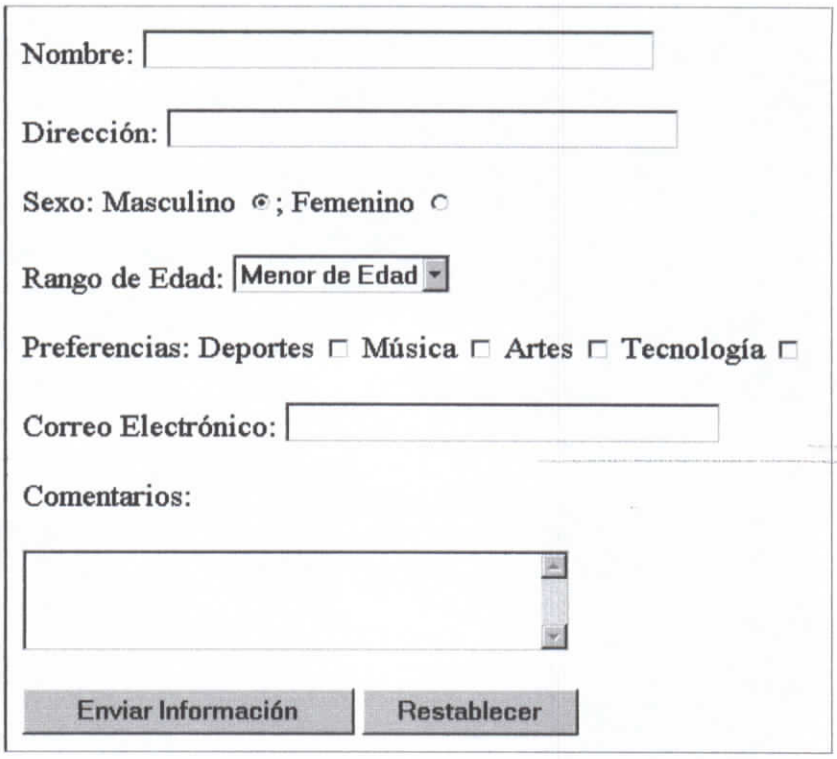
A parte del elemento **<INPUT>** con todas sus variaciones, se puede introducir dentro de un formulario, campos de selección, determinados a través del elemento **<SELECT> </SELECT>.** Este tipo de campos despliegan una lista de opciones, entre las que el usuario debe escoger una opción. Las diferentes opciones de la lista se indican con el elemento **<OPTION>**. Esta directiva puede incluir el parámetro **selected** para indicar cual es la opción por defecto. En caso de que no se especifique, se tomara por defecto la primera opción de la lista.

También se puede incluir áreas de texto, las cuales son un campo de texto de múltiples líneas. Normalmente se utiliza para que se incluyan comentarios. La directiva usada es **<TEXTAREA> </TEXTAREA>.**

A continuación se presenta un ejemplo de un formulario HTML:

```
<form method="POST" action="archivo_de_proceso.asp">

Nombre:
 <input type="text" name="Texto1" size="38">
Dirección:
 <input type="text" name="Texto2" size="38">

Sexo:
  Masculino
 <input type="radio" value="V1" checked name="Radio1">;
 Femenino
 <input type="radio" name="Radio2" value="V2"></p>

 Rango de Edad:
 <select size="1" name="Lista1">
  <option>Menor de Edad</option>
  <option>Entre 18 y 25</option>
  <option>Entre 26 y 40</option>
  <option>Más de 40</option>
 </select></p>

 Preferencias:
  Deportes <input type="checkbox" name="Chequeo1" value="ON">
  Música <input type="checkbox" name="Chequeo2" value="ON">
  Artes <input type="checkbox" name="Chequeo3" value="ON">
  Tecnología <input type="checkbox" name="Chequeo4" value="ON">

Correo Electrónico:
  <input type="text" name="Texto3" size="32">

Comentarios:
 <textarea rows="3" name="S1" cols="32"></textarea>
 <input type="submit" value="Enviar Información" name="Boton1">
 <input type="reset" value="Restablecer" name="Boton2">
</form>
```

Este código HTML, presentará el siguiente formulario en el browser:



Si se llena este formulario y se pulsa el botón **Enviar Información**, se enviará la información al archivo de proceso señalado en el atributo **action** del formulario. Si se pulsa el botón **Reestablecer,** se borrará toda la información ingresada en los campos del formulario.

6.3 XHTML

6.3.1 DEFINICIÓN

La especificación XHTML 1.0 (recomendación del 26 de Enero del 2000), es una reformulación del HTML como aplicación XML. Esta normativa ayuda a eliminar los errores gramaticales (usuales en HTML), unificando la descripción del código y

facilitando la portabilidad de los documentos. Se puede señalar, que el XHTML es una **llamada al orden a HTML**.

Las principales razones expuestas por el W3C para aconsejar el uso del XHTML son dos:

- Ya que XHTML en forma general resulta una aplicación XML, ha sido diseñado para ser ampliable (de ahí el añadido de la palabra *Extensible*). Esto significa que se pueden añadir nuevas etiquetas o elementos sin alterar la DTD (Definición de Tipo de Documento) en la que está basado el análisis del documento.
- XHTML ha sido diseñado pensando en la portabilidad. Aunque hoy en día la unión de la potencia de los ordenadores y de los navegadores es suficiente para asumir las posibles diferencias y errores gramaticales del código HTML, se espera que para los próximos años se produzca un aumento considerable de los aparatos que sean capaces de tratar información en código HTML, siempre que esté realmente unificado y se ajuste a normas estrictas para no dar problemas que exijan soluciones complejas.

6.3.2 MEJORAS.

Los desarrolladores Web, que creen trabajos basados en XHTML o migren sus antiguas aplicaciones HTML hacia XHTML, apreciarán las siguientes ventajas:

- Los documentos XHTML son conformes a XML. Como tales, son fácilmente visualizados, editados y validados con herramientas XML estándar.

- Los documentos XHTML pueden escribirse para que funcionen igual o mejor que lo hacían antes, tanto en los visualizadores (browsers) conformes a HTML 4.0, como en los nuevos visualizadores conformes a XHTML 1.0.

- Los documentos XHTML pueden usar aplicaciones (ejemplo: scripts y applets) que se basen ya sea en el Modelo de Objeto de Documento (DOM) de HTML o XML.

- En XML es relativamente fácil añadir nuevos elementos, así como también atributos adicionales a dichos elementos. La familia XHTML está concebida para acomodar estas extensiones a través de módulos XHTML y técnicas para desarrollar nuevos módulos conformes a XHTML.

6.3.3 DIFERENCIAS CON HTML

Las principales diferencias entre el XHTML y el HTML son:

- En XHTML, todas las etiquetas y nombres de atributos deben estar en minúsculas, ejemplo:

En HTML	En XHTML
<BODY BgCOLOR="#FFFFFF">	<body bgcolor="#ffffff">

En este punto existe una excepción, ya que los valores de los atributos definidos por los usuarios pueden estar tanto en minúsculas como en mayúsculas, así que, en el ejemplo anterior, también se podría escribir: <body bgcolor="#FFFFFF">.

- Todos los valores de atributos deben estar encerrados entre comillas o apóstrofes, ejemplo:

En HTML	En XHTML
<TABLE BORDER=2>	<table border="2"> o <table border= '2'>

- Todos los elementos "no vacios" deben ir entre la etiqueta de principio y la etiqueta de final, ejemplo:

En HTML	En XHTML
Texto 1 <P> Texto 2 </P>	<P>Texto 1 </P> <P>Texto 2 </P>

- Los elementos vacíos obligatoriamente deben llevar terminación, ejemplo:

En HTML	En XHTML
  <HR>	</br> </hr>

- Todos los elementos deben estar anidados ordenadamente.

Ya que en HTML, no es una regla esencial, el que todos los elementos estén correctamente anidados, al momento de crear un documento HTML, existen un sinnúmero de ambigüedades y desorganización en el mismo. En XHTML, como en

- El elemento raíz del documento, siempre debe ser *<html>*

No puede existir nada antes de <html> (a excepción de la declaración DTD), y nada después de </html>. Además de esto, esta etiqueta debe estar seguida de la especificación del namespace XHTML, asi:

<html xmlns="http://www/w3/org/TR/xhtml1">

- Los elementos *<script>* y *<style>*

Si un documento XHTML requiere el uso de cualquiera de estos dos elementos, es indispensable incluir una sección CDATA que abarque el contenido de estas etiquetas, así:

En HTML	En XHTML
<SCRIPT LANGUAGE="JavaScript"> <!-- document.write (<P>Texto de Prueba</P>); //--> </SCRIPT>	<script language="JavaScript"> <!-- <![CDATA[document.write (<P>Texto de Prueba</P>);]]> //--> </script>

6.4 SGML (STANDARD GENERALIZED MARKUP LANGUAGE)

6.4.1 DEFINICIÓN

El SGML es un estándar para el manejo de información, adoptado por la ISO en el año de 1986, el cual se deriva del GML (Generalized Markup Language) desarrollado por IBM. El SGML fue diseñado para permitir el intercambio de información entre distintas

plataformas y a partir de él se han derivado el HTML y el XML, es decir, que el SGML es un metalenguaje, a partir del cual se han definido lenguajes de marcas, como los señalados anteriormente.

La función del etiquetado es aportar información que por lo general refleja la estructura jerárquica de un documento, de forma que ayude al lector humano o al ordenador a procesar su contenido. El lenguaje de etiquetado SGML permite distinguir entre el contenido o datos y el etiquetado.

6.4.2 VENTAJAS DEL SGML

- Hace énfasis en las anotaciones estructurales en lugar de las procedurales. Es decir, el estándar de SGML no incluye información sobre como deben ser interpretadas las anotaciones. Su principal objetivo es describir las unidades lógicas de un documento (entidades), más no como deben ser tipografiadas.

- Inicia el concepto de tipos de documentos. Informalmente, se puede hablar de tipos de documento como: memorandums, cartas, reportes, artículos, libros, etc. SGML formaliza esta idea al crear la noción de Definición de Tipo de Documento (Document Type Definition --DTD). Una DTD describe una clase o familia de documentos con ciertas características comunes.

- Contenido independientemente del sistema. SGML utiliza caracteres normalmente encontrados en casi todos los sistemas y plataformas, además define mecanismos para emplear caracteres especiales o poco comunes.

- Permite crear lenguajes basados en etiquetas descriptivas, como el HTML Y XML.

6.4.3 CARACTERÍSTICAS DE UN DOCUMENTO SGML

Cuando se concibe un documento SGML se debe tener en cuenta que:

- El material que constituye un documento se puede distribuir en diferentes archivos, tantos como sean necesarios.
- Un archivo puede contener la portada, otro la introducción, otro un gráfico, otro un organigrama, otro la bibliografía, etc.
- En SGML, cada uno de estos objetos recibe el nombre de **entidad**.
- Las entidades pueden tener cualquier tamaño, haber sido creadas por cualquier programa de software o estar guardadas en cualquier ordenador.
- Las entidades pueden estar compartidas por distintos documentos.
- Un documento estará definido en función de la **estructura de las entidades** que lo conforman.
- Las entidades se organizan en una *estructura lógica,* de manera jerarquizada, lo que se denomina **estructura de los elementos** del documento.

6.4.4 ESTRUCTURA DE UN DOCUMENTO SGML

Al igual que en XML (su hijo), un documento SGML posee la misma estructura, en cuanto a su elaboración, constando de los siguientes puntos:

6.4.4.1 Prólogo SGML.

En un documento SGML, el prólogo se refiere a la declaración del tipo de documento (DTD) que se va ha utilizar, ejemplo:

<!DOCTYPE SGML Public "c:\Mis Documentos\Documento.dtd" >

Como ya se conoce, un DTD (Document Type Definition) sirve para especificar:

- El convenio de las abreviaturas que aparecen en las etiquetas.
- Elementos (nombrados con etiquetas) y su contenido.

Lo cual trae consigo ventajas como:

- Especificación formal e ineludible, de los convenios usados con las etiquetas.
- Una aplicación puede asegurarse que un documento SGML cumple con lo que se ha determinado en el DTD.

6.4.4.2 Los Elementos.

Un elemento tiene el propósito de dar un nombre identificativo a los componentes de la estructura de un documento. Un elemento puede contener otros elementos, pero éste no puede involucrarse en el campo de otros elementos. Si se desea definir un DTD compuesto de elementos específicos, éste luciría así:

<!ELEMENT Direccion - - (Ciudad, Calle, Telefono)>

En este caso, **Direccion** es el identificador general del elemento, los signos - - indican que las etiquetas de inicio y de cerrado son necesarias, y **(Ciudad, Calle, Telefono)** son los nombres de los elementos contenidos en el elemento Direccion.

6.4.4.3 Los Atributos.

Los atributos le dan características específicas a cualquier elemento de un documento SGML, un atributo puede pertenecer a varios elementos, pero su valor debe señalar información única de ese elemento. En un DTD, se puede especificar, que un elemento cualquiera posee un atributo, de la siguiente manera:

<!ATTLIST Direccion Pos>

Esto señala, que el elemento **Direccion** posee un atributo denominado **Pos,** el cual podrá contener cualquier valor que identifique al elemento al cual pertenece.

Como se puede observar, en su forma más básica, un documento SGML tiene la misma estructura de un documento XML, esto quiere decir que un documento SGML es igual a un documento XML. Sin embargo, la razón por la que el WW3 (World Wide Web Consortium) optó por desarrollar una versión simplificada y compacta del SGML, o sea el XML, es que el primero posee una recomendación (documento) demasiada larga (aproximadamente 400 páginas) con muchas reglas sintácticas difíciles de captar y con otras que pocas veces se las podría usar. De esta forma, con el desarrollo del XML, la creación de aplicaciones orientadas al análisis, depuración, despliegue, etc... de un documento XML será mucho más fácil de entender y de implementar.

CAPÍTULO VII

7 SOFTWARE XML

7.1 INTRODUCCIÓN

Debido a que XML es un estándar universal, dirigido al tratamiento de información estructurada (con significado), su utilización no está atada a ninguna plataforma (sistema operativo) ni a ningún software específico. Al contrario, al ser un estándar promulgado por el W3C, cualquier empresa o persona dedicadas al desarrollo de software, está en capacidad de crear sus propios programas que involucren cualquier aspecto de la tecnología XML. Claro está, que cualquier aplicación que sea desarrollada en este contexto, deberá cumplir a cabalidad con todas las especificaciones (reglas) que ha determinado el W3C para cada especificación relacionada con el XML.

Desde que la especificación XML fue promulgada, decenas de empresas desarrolladoras de software dirigieron sus objetivos a la elaboración de herramientas que faciliten el entendimiento y uso de todos los aspectos referentes al XML, hasta que en la actualidad existe un sinnúmero de programas que abarcan todos los campos de trabajo referentes al XML.

En definitiva, un documento XML o cualquier documento relacionado con la tecnología XML, puede ser elaborado desde el más simple editor de texto, hasta el más complejo software específico de edición de documentos XML.

A continuación se detalla los diferentes tipos de software XML existentes en el mercado, con enlaces a los sitios en donde se encuentran las herramientas más útiles y populares:

7.2 EDITORES XML

Este tipo de aplicaciones permiten escribir documentos XML, tomando muy en cuenta la correcta estructura que deben tener los mismos (bien formados), también permiten la edición de DTDs (Tipos de Documentos), documentos esquema (Schemas) y algunos ofrecen la capacidad de validación de un documento XML sobre un DTD o un Schema. Entre muchos de los editores que se pueden encontrar, se pueden señalar los siguientes:

7.2.1 XML SPY

Este software comercial trabaja sobre plataformas Windows 95/98/NT/2000, y posee un ambiente integrado de desarrollo para XML, el cual incluye edición y validación de documentos XML, edición y validación de Schemas, y además también soporta edición y transformación XSL. Posee una interfaz amigable, con opciones gráficas en lo referente al árbol de datos de los documentos y está dirigido a un nivel intermedio de desarrollo de documentos XML. Se puede visitar su página en: http://www.xmlspy.com/.

7.2.2 XMETAL

Esta aplicación, igualmente comercial, está orientada a plataformas Windows. Brinda al desarrollador, un ambiente muy familiar al momento de la edición de documentos

XML, posee además un conjunto de herramientas muy útiles, como vistas múltiples de documentos e inspecciones de datos del documento. Brinda la posibilidad de edición y validación de DTDs y tiene un soporte básico para SGML. Su enlace es: http://www.xmetal.com/

7.2.3 XML STYLER

Este programa comercial está específicamente orientado a la creación y modificación de documentos XSL. Brinda una agradable interfaz gráfica que facilita el tratamiento de este tipo de documentos, eliminando lo laborioso que puede ser el desarrollo de un documento XSL. Trabaja para plataformas Windows y también para Solaris. Su contacto es: http://www.arbortext.com/xmlstyler/

7.2.4 XML NOTEPAD

Esta herramienta pertenece a Microsoft y permite la rápida elaboración y edición de documentos XML, DTDs, y XSL. Ofrece una interfaz gráfica simple y permite visualizar el árbol de la estructura del documento XML. Se lo puede conseguir gratuitamente en: http://msdn.microsoft.com/xml/notepad/intro.asp

7.2.5 XPUBLISH

Este software está destinado a la plataforma Macintosh, el cual provee soporte de edición de documentos XML y CSS, además permite al usuario generar HTML a partir de documentos XML que utilizan CSS, con el propósito de poder ser visualizados en cualquier browser. Este producto comercial está en:

http://interaction.in-progress.com/xpublish/index

7.3 PARSERS XML

Este tipo de software permite el análisis sintáctico de cualquier documento XML. En este grupo se puede encontrar dos tipos de analizadores (parsers): los no validadores y los validadores.

- **Parsers no validadores:** Son aquellos que no verifican un documento XML en contra de algún DTD o un Schema, esto quiere decir que únicamente verifican que el documento esté bien formado, o sea que cumpla las reglas sintácticas establecidas en la recomendación del W3C.

- **Parsers validadores:** Son aquellos que a parte de verificar que el documento esté bien formado, chequean que cumpla con las reglas de un DTD o un Schema elaborados para dicho documento.

7.3.1 LISTA DE PARSERS NO VALIDADORES :

A continuación se detalla algunos de los más importantes Parsers Validadores, que se pueden encontrar:

7.3.1.1 XP

Este parser es desarrollado en Java y detecta cualquier mal formación de documentos XML, además puede analizar sintácticamente entidades externas como son los DTDs. Trabaja bajo aplicaciones Java. Su enlace es: http://www.jclark.com/xml/xp/index.html

secretaría y además analizar el rendimiento académico del curso y de cada alumno.

3. **Consulta, Actualización, Modificación de notas para la Secretaría de la Escuela,** el cual permite a la secretaria o a cualquier persona encargada, consultar, actualizar o modificar las notas de cualquier alumno de la Escuela de Sistemas, que esté matriculado en el semestre (periodo) en curso. Esta tarea posibilitará una alternativa paralela de manipulación de notas.

Es necesario señalar que, al no existir actualmente en la Escuela de Sistemas una fuente de datos apropiada para realizar la implementación real del proyecto (aplicación), se utilizará una fuente de datos (base de datos) no real, la cual, en cierta forma ayudará a dar una idea de la forma en la cual los datos reales estarán almacenados.

8.2 ANÁLISIS DEL SOFTWARE

8.2.1 FLUJO DE INFORMACIÓN

Los principales datos (información) que la aplicación necesita, corresponden a los registros de notas que los Señores Profesores entregan en secretaría, dicha entrega se realiza cuatro veces al mes, repartida de la siguiente manera: los tres primeros registros de notas corresponden a las calificaciones de los tres bimestres, y el último registro se refiere a la nota del examen final.

Cuando el Sr. Profesor ha registrado en papel, las notas de cada uno de sus alumnos, después de tener el aporte final de cada bimestre, éstas son entregadas a la Secretaria, con el fin de que sean ingresadas a la base de datos, posteriormente se imprimen las notas de cada bimestre y se las publica en la cartelera de la Universidad.

Este flujo de la información, plasmado en la aplicación, se muestra en los siguientes gráficos:

Nivel 0

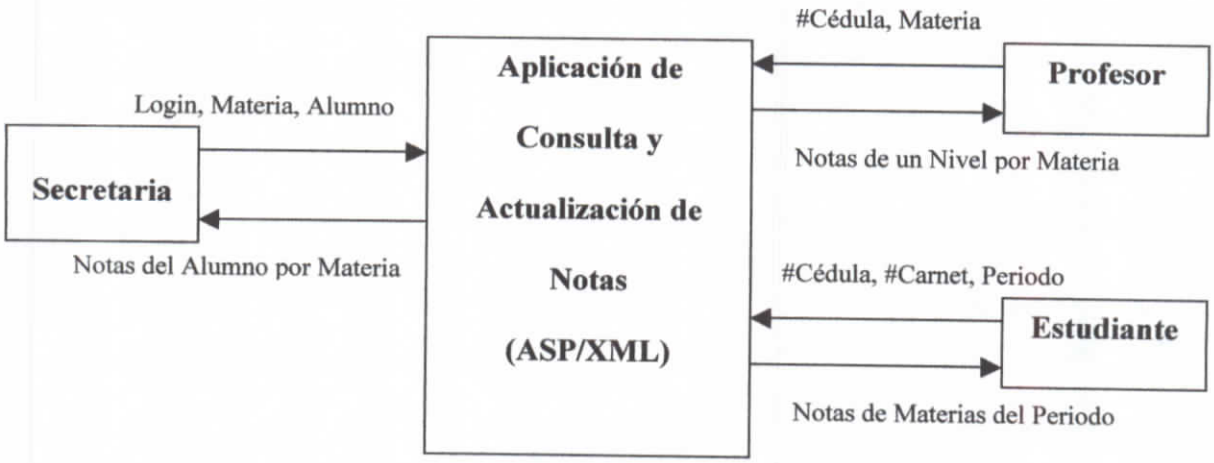


Figura # 8. Flujo de Datos, Nivel 0

Nivel 1

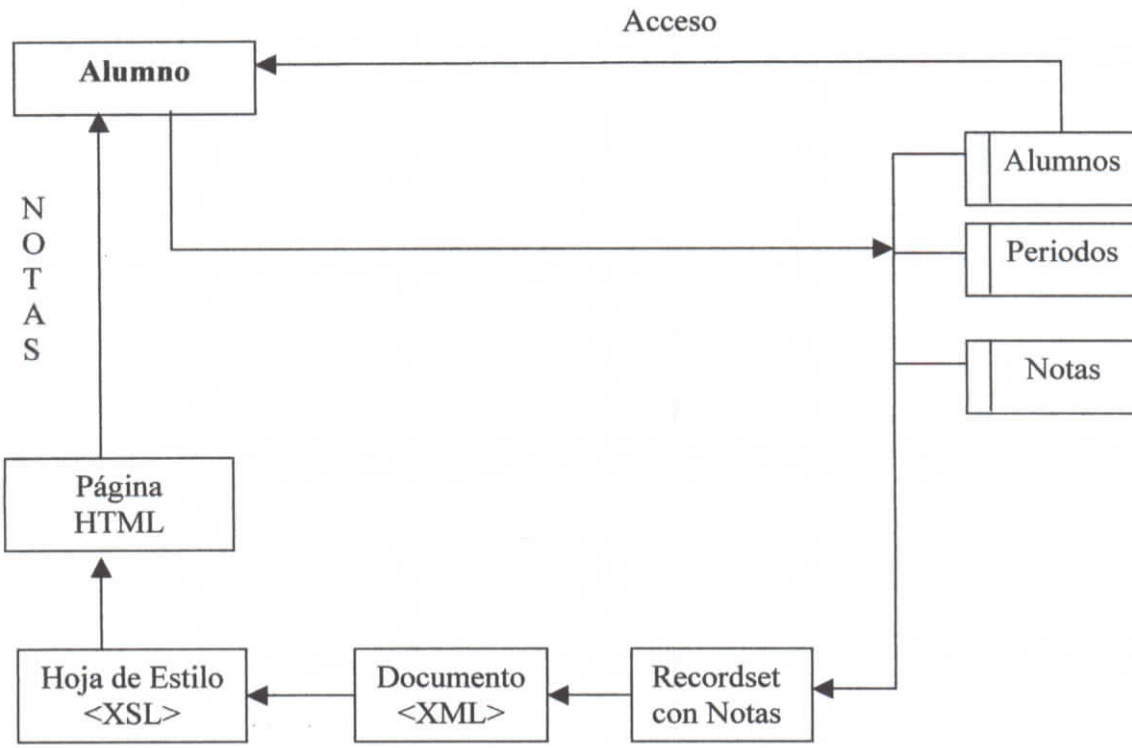


Figura # 9. Flujo de Datos, Módulo Alumnos, Nivel 1

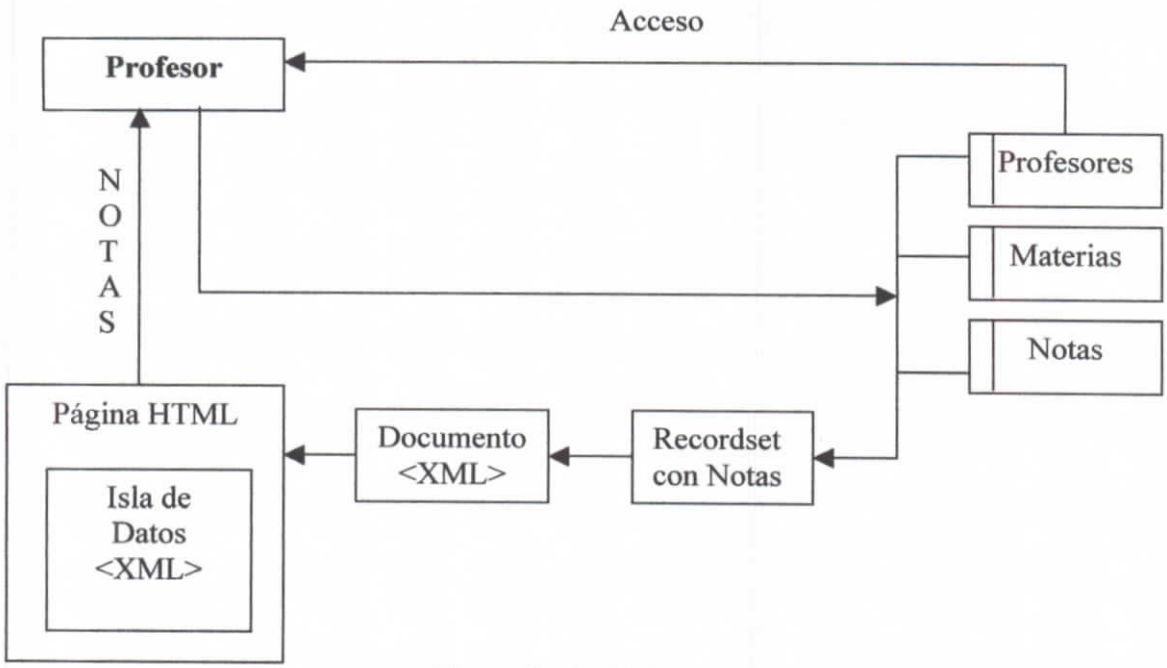


Figura # 10. Flujo de Datos, Módulo Profesores, Nivel 1

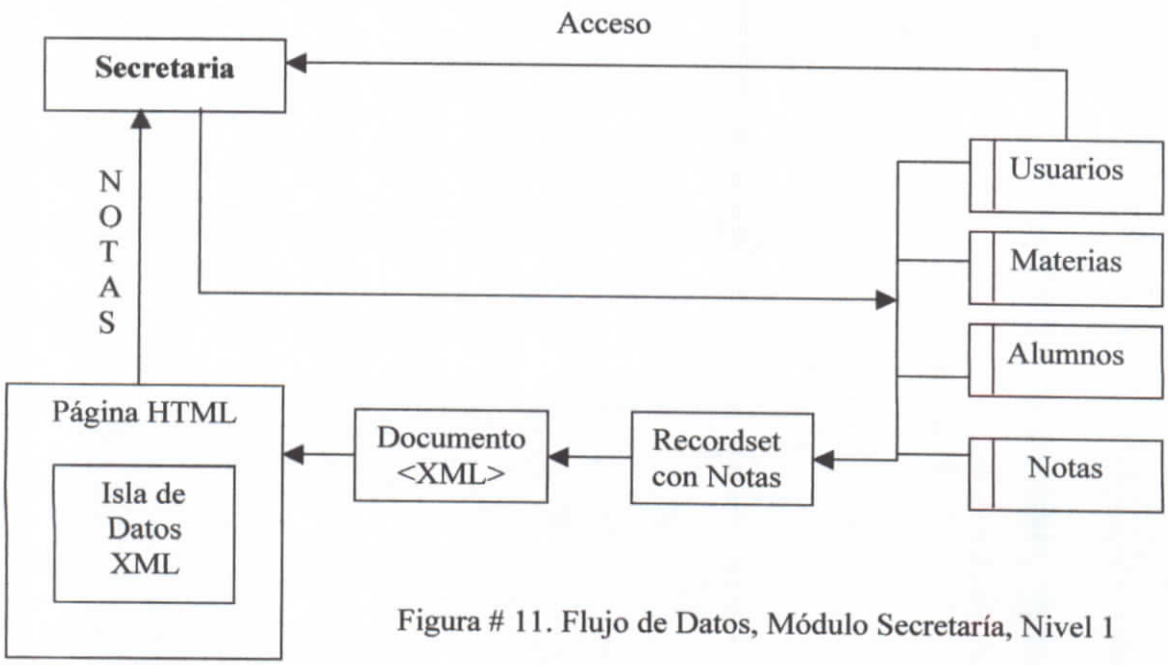


Figura # 11. Flujo de Datos, Módulo Secretaría, Nivel 1

8.3 DISEÑO DEL SISTEMA

8.3.1 SELECCIÓN Y DESCRIPCIÓN DE HERRAMIENTAS

En una aplicación Web, en la cual se desee utilizar XML, es necesario conjugar el uso de esta herramienta junto con otras, con el propósito de lograr los resultados esperados.

Para el desarrollo de la aplicación de "Consulta y Actualización de Notas", se utilizaron las siguientes herramientas:

- Visual Interdev versión 6.0
- Microsoft Access 2000
- Internet Information Server
- Tecnología ASP (Active Server Pages)
- HTML versión 4.0
- XML versión 1.0
- ADO (ActiveX Data Object)

8.3.1.1 Visual Interdev versión 6.0

Esta herramienta está dirigida a desarrolladores de aplicaciones Web, la cual brinda la posibilidad de realizar tareas como las de: diseño, construcción, prueba, depuración, desarrollo y manejo de las mismas.

Visual Interdev es usado para crear páginas Web, usando HTML y script, tomando ventaja de los últimos avances de la tecnología relacionada con los web browsers.

Además provee un robusto ambiente de desarrollo con controles en tiempo de diseño y una caja de herramientas para poder realizar un rápido desarrollo, prueba y depuración de cualquier aplicación orientada al Web. La Figura # 9 muestra una ilustración del ambiente de desarrollo de Interdev, señalando sus principales características y herramientas:

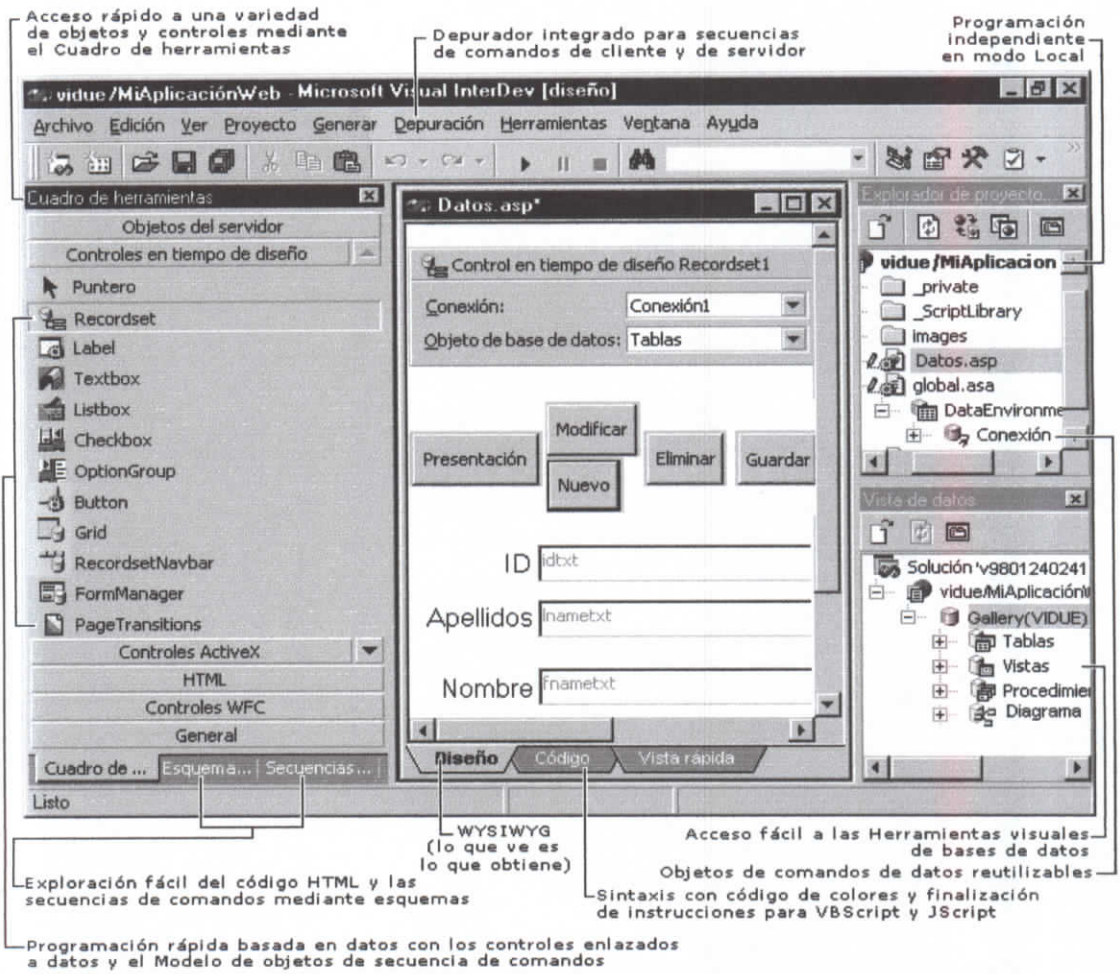


Figura # 12: Ambiente de Desarrollo de Visual Interdev

8.3.1.2 Microsoft Access 2000

Este software, perteneciente a la casa Microsoft, es una aplicación que permite la creación y manipulación de base de datos relacionales. Además brinda la posibilidad de trabajar con formularios, consultas, informes, macros y añade la facilidad de crear código interno a través de un ambiente de programación Visual Basic para aplicaciones.

En realidad es una herramienta relativamente fácil de usar, y en el caso del desarrollo de la aplicación en curso, servirá de sitio de almacenamiento de los datos requeridos para la misma.

8.3.1.3 Internet Information Services

Internet Information Services es el servidor Web utilizado para el desarrollo y la ejecución de la aplicación. Este servidor Web viene integrado con Windows 2000.

Se puede usar esta herramienta, con el propósito de levantar sitios Web o sitios FTP sobre una Intranet Corporativa, crear amplios sitios Web para Internet, o desarrollar programas basados en componentes.

Las principales características de este Servidor Web son:

- **Seguridad**

Internet Information Services, ofrece una robusta autenticidad de usuarios a través de servidores proxy y firewalls, además brinda comunicaciones seguras, por medio de Secure Sockets Layer (SSL), permitiendo un confiable intercambio de información

entre clientes y servidores, también utiliza el Server-Gated Cryptography (SGC), el cual es una extensión del IIS y permite encriptación de datos de 128 bits.

- **Administración**

Esta característica ofrece opciones como las de:

- Reinicio del Servidor, sin necesidad de iniciar otra vez la computadora.
- Posibilidad de realizar copia de seguridad de los datos de configuración
- Información de los sitios Web que están utilizando recursos del CPU en el servidor.
- Limitación del porcentaje de tiempo que el CPU gasta en estados fuera de procesos en lo concerniente a aplicaciones ASP, ISAPI y CGI.
- Envío de mensajes a los clientes, cuando un error ha sido detectado en sus sitios Web.
- Configuración de permisos de lectura, escritura, ejecución, script a nivel de directorios o de archivos.

- **Programabilidad**

IIS permite la creación de contenido dinámico dentro de un sitio Web, a través del uso de script y componentes del lado servidor. Active Server Pages (ASP), provee una fácil alternativa para incrustar secuencias de comandos o componentes de servidor, realizadas en cualquier lenguaje script, dentro de páginas HTML.

- **Estándares de Internet**

IIS obedece al estándar HTTP 1.0, incluyendo características como: la habilidad de personalizar mensajes de error HTTP y soporte para el uso de encabezados HTTP. Se puede usar los servicios SMTP y NNTP, con el propósito de configurar correo de Intranet y servicio de noticias que trabajan en conjunción con IIS. Además IIS, permite resumir archivos que se están bajando a través de FTP. En adición a esto, IIS provee la más rápida transmisión de páginas entre el servidor Web y clientes habilitados con compresión http, permitiendo la compresión y cacheo de archivos estáticos.

8.3.1.4 Tecnología Active Server Pages (ASP)

ASP es un ambiente de desarrollo que utiliza script de lado servidor, permitiendo la creación de interactivas y poderosas aplicaciones Web. Cuando el servidor recibe un pedido desde un archivo .asp, este procesa el script de servidor contenido en el archivo, y posteriormente construye la página web que es enviada al browser. Además, una página ASP puede contener HTML, incluyendo script del lado cliente. También se puede realizar llamados a componentes COM (Component Object Model), que ejecutan variedad de tareas, tales como: conexiones a base de datos o procesos lógicos específicos.

En resumidas cuentas, un script ASP, permite realizar las siguientes tareas:

- Leer Información desde una petición HTTP
- Personalizar una respuesta HTTP
- Guardar información acerca de un usuario.

- Extraer las capacidades del browser del usuario

Entre las principales ventajas que se pueden destacar de esta tecnología, se puede señalar las siguientes:

- Se puede elegir el lenguaje de programación de entre estos tres: JavaScript, Visual Basic Script y PerlScript. Incluso se pueden tener scripts en distintos lenguajes dentro de la misma página ASP. El lenguaje escogido es compatible con cualquier navegador, ya que se trata de código que se ejecuta en el servidor. Al navegador sólo le llega código HTML.

- Se pueden utilizar componentes del lado de servidor. Esto permite programar al estilo Visual Basic creando objetos y utilizando sus métodos y propiedades.

- Se puede acceder a bases de datos con objetos "recordset" de un modo muy parecido a como se hace en Visual Basic.

- Tiene persistencia de variables en memoria (entre distintas visualizaciones de páginas Web) que se pueden asociar a cada sesión de usuario o a la aplicación en su conjunto. Esto resuelve de forma elegante uno de los mayores problemas de la programación en la Web: El servidor Web no tiene memoria entre la visualización de una página Web y la siguiente.

La principal diferencia entre páginas HTML y ASP es el sitio donde el script es ejecutado. DHTML, o script cliente, es ejecutado en el cliente (browser), después de

que la página es enviada desde el servidor. ASP o script servidor, es ejecutado en el servidor antes que la página sea enviada al browser. El servidor Web procesa el script y genera las páginas HTML que son retornadas al Web browser. A continuación se presenta ejemplos de script servidor y script cliente, realizando la misma tarea, de presentar al cliente la hora y fecha actuales:

Script Cliente (página html)

```
<HTML>
<BODY>
<H3>Bienvenidos</H3>
La hora es :
<SCRIPT LANGUAGE=VBScript>
Document.Write time()
</SCRIPT>.<BR>
La fecha es:
<SCRIPT LANGUAGE=VBScript>
Document.Write date()
</SCRIPT>.
</BODY>
</HTML>
```

Script Servidor (página .asp)

```
<HTML>
<BODY>
<H3>Bienvenidos...</H3>
La hora es : <%=Time()%><BR>
La fecha es : <%=Date()%>.
</BODY>
</HTML>
```

Como se puede observar, una secuencia de comandos dentro de una página ASP, está delimitada por **<%** *secuencia de comandos* **%>**, en este caso se utilizará el lenguaje de servidor que esté configurado por defecto, no obstante, se puede también especificar el inicio de una secuencia de comandos, determinando que lenguaje se va ha utilizar, usando la directiva @, así:

```
<%@Language=VBScript
  secuencia de comandos
  ...
%>
```

o también, se puede declarar el inicio de una secuencia de comandos, a través de la etiqueta *<SCRIPT>...</SCRIPT>* , claro está, determinando que el script será ejecutado en el servidor así:

```
<SCRIPT LANGUAGE=VBSCRIPT  RUNAT=SERVER>
  secuencia de comandos
  ...
</SCRIPT>
```

Objetos de Servidor disponibles en ASP

En una página ASP, el script servidor tiene la facultad de utilizar algunos objetos; la tabla # 28 describe estos objetos:

Objeto	Descripción
Request	Recupera los valores que el browser pasa al servidor durante un pedido HTTP.
Response	Controla que información es enviada al browser en el mensaje de respuesta HTTP.
Session	Usado para manejar y guardar información acerca de una sesión de usuario en particular.
Application	Usado para manejar y guardar información acerca de la aplicación Web.
Server	Provee acceso a recursos que residen en el servidor.
ObjectContext	Usado para accionar o abortar una transacción manejada por Microsoft Transaction Server (MTS) para páginas ASP que se ejecutan en una transacción.

Tabla # 28. Objetos disponibles en ASP

En forma general, con todos estos objetos se puede:

- Recuperar información pasada desde el browser al servidor usando el objeto **Request.**
- Enviar salidas (resultados) al browser usando el objeto **Response.**
- Guardar información para un usuario específico usando el objeto **Session**.
- Compartir información entre todos los usuarios de las aplicaciones, usando el objeto **Application**

- Trabajar con las propiedades y métodos de componentes en el servidor, usando el objeto **Server**.

8.3.1.5 ActiveX Data Objects (ADO)

Esta tecnología desarrollada por Microsoft, permite escribir aplicaciones que acceden a información desde fuentes de datos OLE DB, incluyendo fuentes ODBC. Cuando se usa ADO dentro de una página ASP, todo el acceso a datos se realiza en el servidor.

ADO es completamente independiente del lenguaje de programación para accesar a datos, de tal manera que se puede escribir código en una variedad de lenguajes tales como: Visual Basic, Visual C + +, VBScript, etc..., con el propósito de obtener los datos requeridos.

ADO representa una colección de objetos COM, que permiten el acceso a una fuente de datos a través de su proveedor OLE DB. A continuación se describen los objetos existentes en ADO versión 2.5.

Objeto Connection

Sirve para realizar una conexión a una base de datos específica. Sus propiedades y métodos más importantes son:

ConnectionString

Sirve para especificar una cadena de texto, con la información necesaria para establecer una conexión con la fuente de datos. Aunque hay hasta 7 argumentos distintos que se

pueden suministrar en esta cadena, los básicos son el DSN que identifica al archivo de base de datos y el login y password si existen. Los argumentos se separan con punto y coma.

Ejemplo: "DSN=midsn; UID=milogin; PWD=micontraseña"

Open

Abre la conexión con la base de datos.

Close

Cierra la conexión con la base de datos.

Ejemplo de una conexión a una base de datos:

```
<%
 Set conexion = Server.CreateObject("ADODB.Connection")
 con.ConnectionString = "DSN=midsn"
 con.Open
 ' .....
 ' .....
 miconexion.Close
%>
```

Objeto Recordset

Este objeto permite el acceso directo a un conjunto de registros alojados en las tablas de la base de datos.

Un objeto Recordset representa una tabla, que puede ser una tabla completa de la base de datos o bien una obtenida mediante un filtrado o sentencia SQL. En cualquier caso, el objeto representa a la tabla con todos sus registros, aunque sólo uno de ellos es el activo. El registro activo es en el cual se puede leer o modificar los valores de los campos. A continuación se detallan las propiedades y métodos más relevantes:

ActiveConnection

Indica al Recordset la base de datos que en ese momento se encuentra activa (abierta).

Open

Indica al objeto Recordset el conjunto de registros que abarcará, a través de una sentencia SQL.

Ejemplo:

```
<%
 Set con =
Server.CreateObject("ADODB.Connection")
 con.ConnectionString = "DSN=midsn"
 con.Open
 Set rs = Server.CreateObject("ADODB.Recordset")
 rs.ActiveConnection = miconexion
sql = "select * from tbl_clientes"
rs.Open sql
 ..... ' Operaciones con los datos
 rs.Close
 con.Close
%>
```

EOF (End of file)

Esta propiedad indica si el recordset tiene un estado de fin de archivo, en donde tiene un valor de TRUE, caso contrario tendrá un valor de FALSE.

BOF (Begin of file)

Esta propiedad indica si el recordset tiene un estado de inicio de archivo, en donde tiene un valor de TRUE, caso contrario tendrá un valor de FALSE.

MoveFirst

Mueve el cursor al primer registro de la tabla

MoveLast

Mueve el cursor al último registo de la tabla

MoveNext

Mueve el cursor al siguiente registro

MovePrevious

Mueve el cursor al registro anterior

Fields

A través de esta propiedad, se puede tener acceso a los datos de un registro activo, por ejemplo: después de haber abierto un recordset a través de una sentencia sql, se puede acceder a los datos de un campo en específico, así:

```
<%
 Set con = Server.CreateObject("ADODB.Connection")
 con.ConnectionString = "DSN=midsn"
 con.Open
 Set rs = Server.CreateObject("ADODB.Recordset")
 rs.ActiveConnection = miconexion
sql = "select nombre_cliente from tbl_clientes  where id_cliente='001'"
Rs.open sql, conexión_activa
session("nombre_cliente") = Rs.fields("nombre_cliente")
rs.Close
con.Close
%>
```

Objeto Command

Este objeto es la representación de un comando que se ejecuta sobre la base de datos. Este comando puede ser:

- Llamada a un procedimiento guardado en la base de datos.
- El texto de una sentencia SQL.
- El nombre de una tabla.

Las propiedades y métodos más importantes son:

ActiveConnection

Es una referencia al objeto connection que enlaza con la base de datos. Es imprescindible asignar esta propiedad antes de invocar el método command.execute para ejecutar el comando.

8.3.2 DISEÑO DE LA BASE DE DATOS

Como se señaló anteriormente, actualmente en la Universidad no existe disponible una fuente de datos apropiada que sirva para obtener datos (notas) reales, con los cuales se pueda trabajar e implementar directamente la aplicación. Debido a esta situación, se ha diseñado una base de datos cuyas tablas están compuestas solo por campos necesarios, de esta forma se puede acercar al diseño real de la base de datos que en un futuro será instalada en la Universidad.

8.3.2.1 Descripción de la Base de datos

La base de datos se denomina **NOTAS** y la descripción de sus tablas se detalla a continuación:

Base de Datos: Notas **Tabla:** tbl_alumnos

Descripción: Tabla en la cual se almacena la información de cada estudiante de la Escuela de Sistemas.

Campo	Tipo	Longitud	Clave
id_alumno	Texto	10	Principal
cedula_alumno	Texto	10	
nombres_alumno	Texto	40	
apellidos_alumno	Texto	40	

Base de Datos: Notas **Tabla:** tbl_profesores

Descripción: Tabla en la cual se almacena la información de cada profesor de la Escuela de Sistemas.

Campo	Tipo	Longitud	Clave
id_profesor	Texto	10	Principal
nombre_profesor	Texto	40	

Base de Datos: Notas **Tabla:** tbl_materias

Descripción: Tabla en la cual se almacena la información de las materias que se dictan en la Escuela de Sistemas.

Campo	Tipo	Longitud	Clave
id_materia	Texto	10	Principal
id_profe_materia	Texto	10	Foránea
nombre_materia	Texto	40	
nivel_materia	Texto	2	

Base de Datos: Notas **Tabla:** tbl_periodos

Descripción: Tabla en la cual se almacena la información de cada periodo en curso.

Campo	Tipo	Longitud	Clave
id_periodo	Texto	10	Principal
desc_periodo	Texto	40	

Base de Datos: Notas **Tabla:** tbl_paralelos

Descripción: Tabla en la cual se almacena la información de los paralelos existentes para cada materia.

Campo	Tipo	Longitud	Clave
id_paralelo	Texto	10	Principal
id_materia_paralelo	Texto	10	Foránea
desc_paralelo	Texto	40	

Base de Datos: Notas **Tabla:** tbl_alum_periodos

Descripción: Tabla en la cual se almacena la información de los periodos en los que cada alumno ha sido matriculado

Campo	Tipo	Longitud	Clave
id_periodo_alum_p	Texto	10	Foránea
id_alumno_alum_p	Texto	10	Foránea

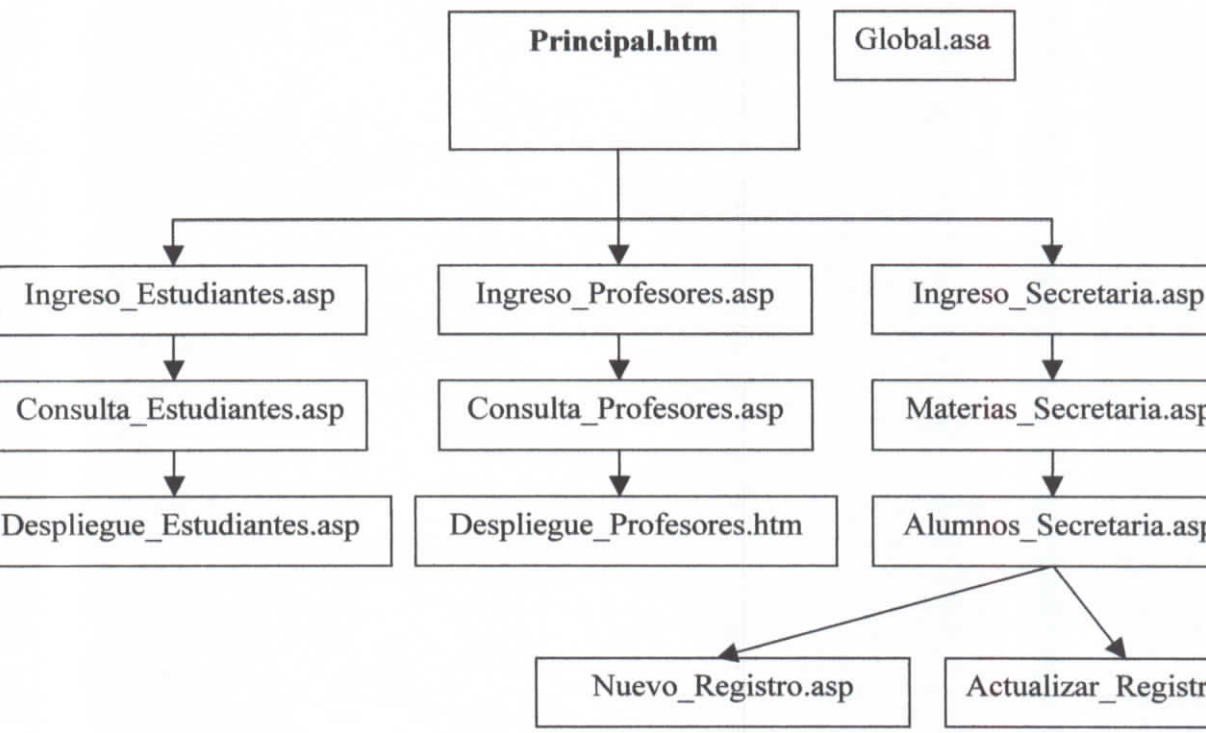
Base de Datos: Notas **Tabla:** tbl_notas

Descripción: Tabla en la cual se almacena la información de cada registro de notas.

Campo	Tipo	Longitud	Clave
id_alumno_notas	Texto	10	Foránea
id_periodo_notas	Texto	10	Foránea
id_paralelo_notas	Texto	10	Foránea
parcial1_notas	Decimal		
parcial2_notas	Decimal		
parcial3_notas	Decimal		
parcial4_notas	Decimal		

8.4 ESTRUCTURA DE LA APLICACIÓN

A continuación se muestra la interfaz, que el usuario final tendrá dentro de la aplicación, y la manera cómo se enlazan las páginas del sitio Web.



8.5 XML EN LA APLICACIÓN

El rol que cumple el XML dentro de la aplicación, está destinado al despliegue de los datos (notas) en el browser a partir de un "recordset", obtenido de una consulta específica y posteriormente devuelto al cliente en forma de XML con estilo. Para realizar esta tarea, en la aplicación se han empleado dos métodos:

8.5.1 OBTENCIÓN DE XML A PARTIR DE UNA PÁGINA ASP, UTILIZANDO ADO (ACCESS DATA OBJECT) Y APLICANDO XSLT

Dentro de la aplicación existen páginas ASP que devuelven al cliente un documento XML como tal, partiendo de una consulta SQL y posteriormente obteniendo un "recordset" a través de ADO. Ejemplo:

```
<% option explicit %>
<!-- METADATA TYPE="typelib"
   FILE="C:\Program Files\Common Files\System\ado\msado15.dll" -->
<%
 Response.ContentType = "text/xml"
%>

 <?xml version="1.0"?>
 <?xml-stylesheet  type="text/xsl"  href="estilo.xsl"?>

<%
 dim sql
 dim rs
 sql = "select parcial1_notas,parcial2_notas,parcial3_notas,final_notas from tbl_notas " & _
"where id_alumno_notas = '" & Request.QueryString("codigo_alumno") & "' "  & _
"and id_materia_notas = '" & session("id_materia") & "'"  & _
"and id_paralelo_notas = '" & session("id_paralelo") & "'"  & _
"and id_periodo_notas = '" & session("id_periodo_actual")  & "'"

 set rs = server.CreateObject("adodb.recordset")
 rs.Open session("sql"),application("con")
 rs.Save response,adPersistXML
%>
```

Dentro de esta página ASP, se puede destacar los siguientes puntos:

- En primer lugar, para que se devuelva al cliente datos XML, se debe configurar la propiedad **ContentType** del objeto **Response** en "**text/xml**", esto significa que los datos devueltos al cliente, no será código HTML, sino será XML.

- Si se desea retornar un documento XML al cliente, obviamente se requiere del prólogo del mismo (indespensable). Si retornamos el XML como tal al browser, la información simplemente se presentaría como un documento XML, con el prólogo y todas sus etiquetas, lo cual no sería de mucha ayuda para el usuario final. Por esta razón, se requiere de una hoja de estilos de transformación, la cual le da estilo a la información obtenida (XML), a través de etiquetas HTML. Para esto se debe hacer referencia (dentro del prólogo) a un archivo **.xsl,** así:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="estilo.xsl"?>
```

La hoja de estilos **"estilo.xsl",** utilizada en la aplicación, tiene la siguiente estructura:

```
<HTML>
      <HEAD>
        <TITLE>P.U.C.E.S.A  -- Consulta de Notas -- Resultados</TITLE>
        <H3 align="center"> RESULTADOS DE LA CONSULTA </H3>
      <STYLE>
       .Table {background:black}
       .TableHead {align:center; font:bold; color:white; background:WHITE}
       .TableColumnHead {font:bold 'Serif' bold; color:white; background:#4682b4}
       .TableRow {font:x-small 'Arial'; color:black; background:#CCCCCC}
      </STYLE>
     </HEAD>
     <BODY background="images/BACK2.JPG">
      <DIV xmlns:xsl="http://www.w3.org/TR/WD-xsl" >
      <center>
        <TABLE CLASS="Table" ID="RecordTable">
            <THEAD>
              <TR CLASS="TableHead">
                  <TD CLASS="TableColumnHead">Materia </TD>
                  <TD CLASS="TableColumnHead">Nivel </TD>
                  <TD CLASS="TableColumnHead">Parcial I </TD>
                  <TD CLASS="TableColumnHead">Parcial II </TD>
                  <TD CLASS="TableColumnHead">Parcial III </TD>
                  <TD CLASS="TableColumnHead">Final </TD>
                  <TD CLASS="TableColumnHead">Suma Total</TD>
                  <TD CLASS="TableColumnHead">Profesor </TD>
              </TR>
            </THEAD>

            <xsl:for-each select="xml/rs:data/z:row">
              <TR CLASS="TableRow">
               <TD><xsl:value-of select="@nombre_materia"/></TD>
               <TD><xsl:value-of select="@nivel_materia"/></TD>
               <TD><xsl:value-of select="@parcial1_notas"/></TD>
               <TD><xsl:value-of select="@parcial2_notas"/></TD>
               <TD><xsl:value-of select="@parcial3_notas"/></TD>
               <TD><xsl:value-of select="@final_notas"/></TD>
               <TD><xsl:value-of select="@Suma_Notas"/></TD>
               <TD><xsl:value-of select="@nombre_profesor"/></TD>
               </TR>
            </xsl:for-each>
            </TABLE>
        </center>
        </DIV>
      </BODY>
</HTML>
```

- Una vez que en la página ASP, se haya definido la referencia a la hoja de estilos, lo que resta es devolver el documento XML, el cual nace del "recordset" obtenido a través de la consulta SQL y abierta mediante ADO. Para retornar el documento XML, se utiliza el método **Save** del objeto recordset (**rs**), el cual recibe como parámetros **response** (retornar al cliente) y **adPersistXML**(método para retener los datos XML, incluso después de haber cerrado o removido la fuente de datos original)

```
rs.Open session("sql"),application("con")
rs.Save response,adPersistXML
```

- Cabe señalar que los datos de un registro obtenido de una consulta, se reflejan en el documento XML devuelto, en forma de atributos, por cada campo consultado. Estos atributos se encuentran dentro de la etiqueta <z:row>, es decir, por cada registro obtenido, se encontrará una etiqueta <z:row>. Además es necesario señalar que dentro del documento XML resultante, todas las etiquetas <z:row>, se encuentran delimitadas por la etiqueta <rs:data>, ejemplo:

```
<rs:data>
  <z:row parcial1='8.5' parcial2='7' parcial3='6' final='15' />
  <z:row parcial1='8.5' parcial2='7' parcial3='6' final='15' />
  <z:row parcial1='8.5' parcial2='7' parcial3='6' final='15' />
</rs:data>
```

8.5.2 UTILIZACIÓN DE UNA ISLA DE DATOS

Dentro de la aplicación, también se da uso de la técnica denominada **Isla de Datos,** la cual es una manera de vincular información de un documento XML a elementos HTML, a través de la utilización de la etiqueta **<XML>** dentro de código HTML

El elemento **<XML>** marca el inicio de una Isla de Datos. Este posee dos atributos: **ID** el cual brinda un nombre que sirve para hacer referencia a la Isla de Datos y **SRC** el cual especifica el documento XML del cual se van a vincular los datos, así:

<XML ID='XMLID' **SRC=**'documento.xml'**>**

Ejemplo de la aplicación:

```
<%@ Language=VBScript %>
<% option explicit %>
<HTML>
<HEAD>
<SCRIPT LANGUAGE=vbscript>
<!--
Sub window_onload()
  Set objXML = document.all("dsoData")
  Set objRecs = objXML.recordset
  Set objRecs = objRecs("rs:data").value
  Set objRecs = objRecs("z:row").value
  For Each objField in objRecs.Fields
   strFieldName = objField.Name
   If strFieldName <> "$Text" then
     if strFieldName = "parcial1_notas" Then
       document.all("text1").value= objField.value
     elseif strFieldName = "parcial2_notas" Then
       document.all("text2").value= objField.value
      elseif strFieldName = "parcial3_notas" Then
       document.all("text3").value= objField.value
         elseif strFieldName = "final_notas" Then
           document.all("text4").value= objField.value
     end if
   end if
  Next
End Sub
-->
</SCRIPT>
</HEAD>
<BODY background=images/BACK2.JPG>
<XML ID="dsoData" SRC="Ado_persist_secre.asp"></XML>
<%>
 ' Script de Servidor
<%>

</BODY>
</HTML>
```

En el ejemplo anterior, se puede considerar los siguientes puntos:

- Como se puede observar, dentro de la etiqueta <BODY>, se encuentra especificada la etiqueta **<XML>,** en donde se establece un identificador **ID="dsoData",** y también se establece el origen del documento fuente (XML), del cual se van a vincular los datos: **SRC="Ado_persist_secre.asp",** en este caso no se especifica directamente el documento XML como tal, sino que, se hace referencia a la página ASP que da origen al documento (como en el punto 8.5.1)

- Una vez que se ha especificado el origen de los datos (XML), a través de la etiqueta **<XML>,** se utiliza un procedimiento de Script cliente, ejecutado con el evento **Window_OnLoad(),** en donde se tiene acceso a las etiquetas y datos del documento XML, referenciado a través de la Isla de Datos y en donde además se recorre el conjunto de registros XML, con el propósito de ir asignando los diferentes valores de cada uno de los campos obtenidos con la consulta, a elementos HTML, que en este caso son cajas de texto.

CONCLUSIONES

- El trabajo de disertación, realizado para la obtención del título de Ingeniero en Sistemas, representa una fuente de investigación académica y de desarrollo profesional.

- La elaboración de este tipo de trabajos, refuerza los conocimientos adquiridos en el transcurso del periodo universitario, complementando así la formación del futuro Ingeniero en Sistemas.

- A través del trabajo realizado, se ha podido descubrir las ventajas y beneficios que presta el lenguaje extensible de marcas XML y las tecnologías asociadas con éste.

- El XML representa un estándar de manejo de información, el cual le da significado a la misma.

- El XML no está atado a ninguna plataforma en particular, de esta manera, es totalmente utilizable en cualquier entorno, en el cual se trate con información.

- El XML posibilita el intercambio de información entre plataformas heterogéneas, determinando así, la universalidad de los datos (información).

- El desarrollo de la aplicación de "Consulta y Actualización de Notas" para la Escuela de Sistemas, es un paso importante hacia la automatización de los procesos que se realizan en la misma.

- La investigación de nuevas tecnologías, contribuye a encontrar soluciones alternativas dirigidas a problemas de la vida real.

- La utilización del XML, junto con otras tecnologías como el HTML y ASP, facilitan el desarrollo de aplicaciones personalizadas, dirigidas a necesidades específicas.

- El XML es un lenguaje universal.

- En un futuro no muy lejano el XML será una herramienta universalmente utilizada, gracias a su poder de descripción de la información.

RECOMENDACIONES

- La Escuela de Sistemas debe promover los trabajos de investigación, referentes a nuevas herramientas y tecnologías que se presentan en el mercado, con la finalidad de utilizarlas en aplicaciones específicas acordes con el medio y además brindar fuentes de consulta alternativas para los alumnos.

- Muchos tópicos informáticos de gran importancia (como el XML) escapan del pénsum de estudios normal que se dicta en la Universidad, no obstante, las autoridades deben incluir periodos adicionales, con el propósito de que los Sres. Profesores de la Escuela dicten, al menos en forma general, clases referentes a dichas tecnologías.

- Al ser considerado el XML un lenguaje de estructuración de información de carácter universal, su estudio e investigación resulta imprescindible, ya que se presenta como una tecnología que brinda un sinnúmero de utilidades y aplicaciones en varios campos referentes al trato de información.

- El trabajo desarrollado, debe ser utilizado como una fuente de consulta inicial en lo referente al XML, el cual servirá tanto a profesores como alumnos a tener nociones claras y objetivas acerca del funcionamiento y utilización de esta tecnología de vanguardia.

- Debido a que la aplicación actual es un prototipo que trabaja con una base de datos no real, es recomendable que en un futuro se la pueda implementar, trabajando con datos reales.

BIBLIOGRAFÍA

TEXTOS

Professional XML, Varios Autores, Wrox Press, 2000

Professional ASP XML, Varios Autores, Wrox Press, 2001

REFERENCIAS WEB

- Página oficial de XML,

  http://www.xml.com/

- Página Web del World Wide Web Consortium

  http://www.w3.org

- Introducción al XML en Castellano,

  http://ibium.com/alf/xml/index.asp

- XML en Microsoft,

  http://www.microsoft.com/latam/msdn/recursos/temas/xml/

- Documentación sobre XML,

  http://html.programacion.net/doc/xml.htm

- Café con Leche, XML News and Resources

  http://www.ibiblio.org/xml/

- Página Oficial de SGML

  http://www.sgml.com/

- Páginas relacionadas con software XML

  http://www.xmlspy.com/

  http://www.xmetal.com/

  http://www.arbortext.com/xmlstyler/

  http://msdn.microsoft.com/xml/notepad/intro.asp

  http://interaction.in-progress.com/xpublish/index

  http://www.jclark.com/xml/xp/index.html

  http://www.textuality.com/Lark/

  http://www.hubick.com/software/HXA/

  http://www.microsoft.com/workshop/xml/parser/jparser.asp

  http://www.alphaworks.ibm.com/tech/xml4j

  http://www.datachannel.com/xml_resources/index.shtml

  http://www.softwareag.com/tamino/

  http://www.datachannel.com/products/

  http://www.us.balise.com/

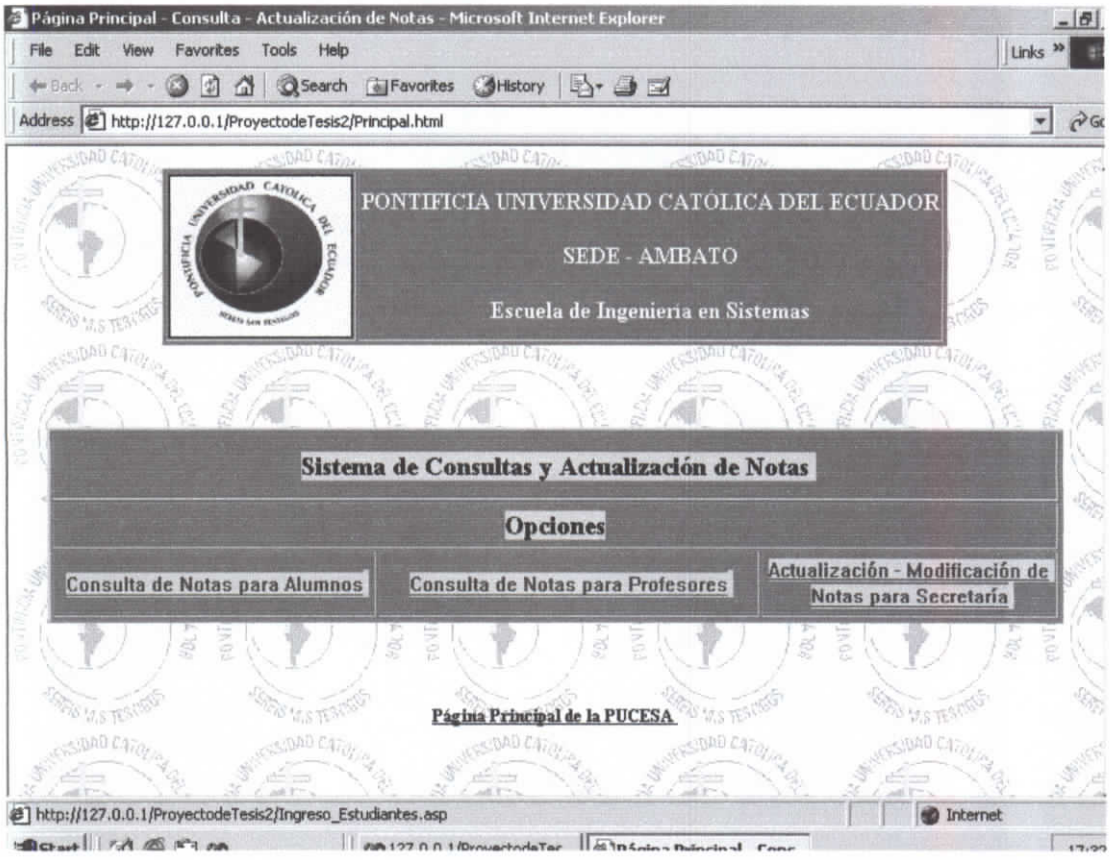
  http://www.omnimark.com/develop/Konstructor/docs/narrativ/63.htm

ANEXOS

PANTALLAS DE LA APLICACIÓN

A continuación se presentan todas las pantallas que conforman la aplicación, junto a las cuales se detalla su descripción y uso:

1. Pantalla Principal

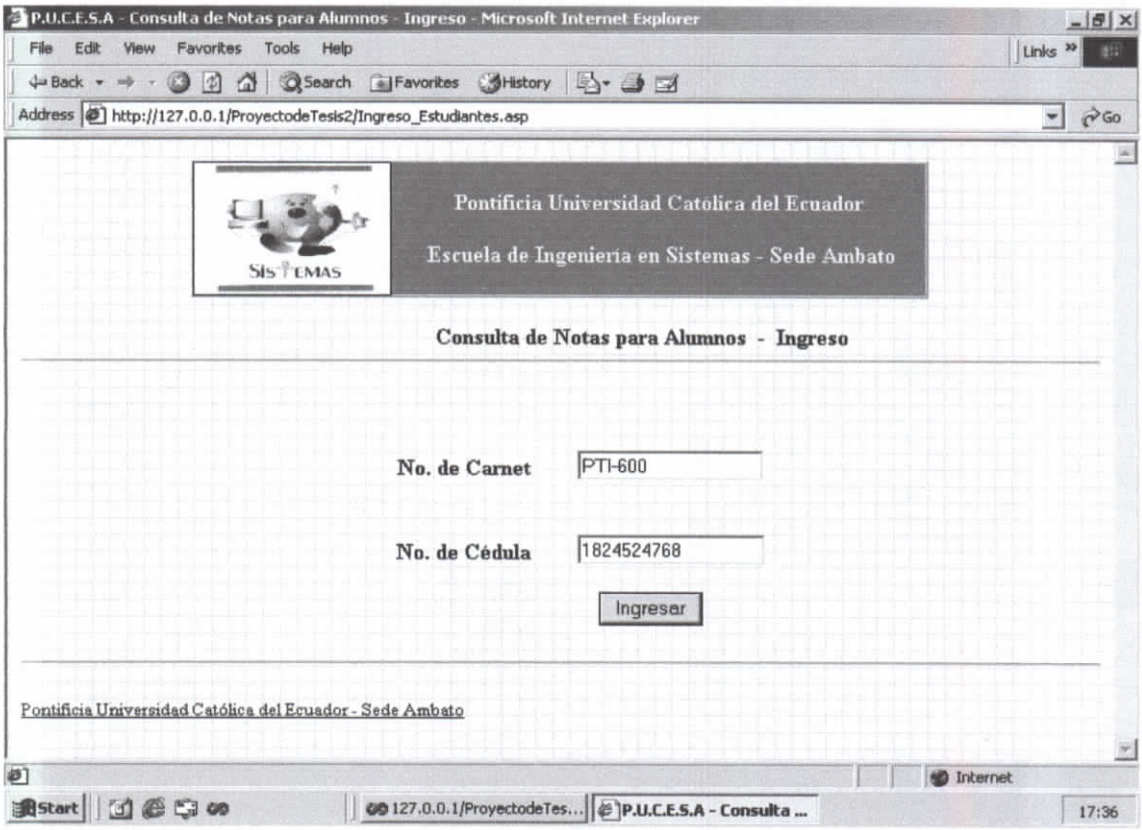


Esta pantalla es la primera de la aplicación, y su función es la de brindar al usuario opciones de ingreso a cada módulo; como se señaló anteriormente, los módulos (opciones en la pantalla) son 3: "Consulta de Notas para Alumnos", "Consulta de Notas para Profesores" y "Actualización y Modificación de Notas para Secretaría", de esta

manera, dependiendo del usuario, se tendrá acceso a cada pantalla de ingreso, correspondiente a la opción elegida.

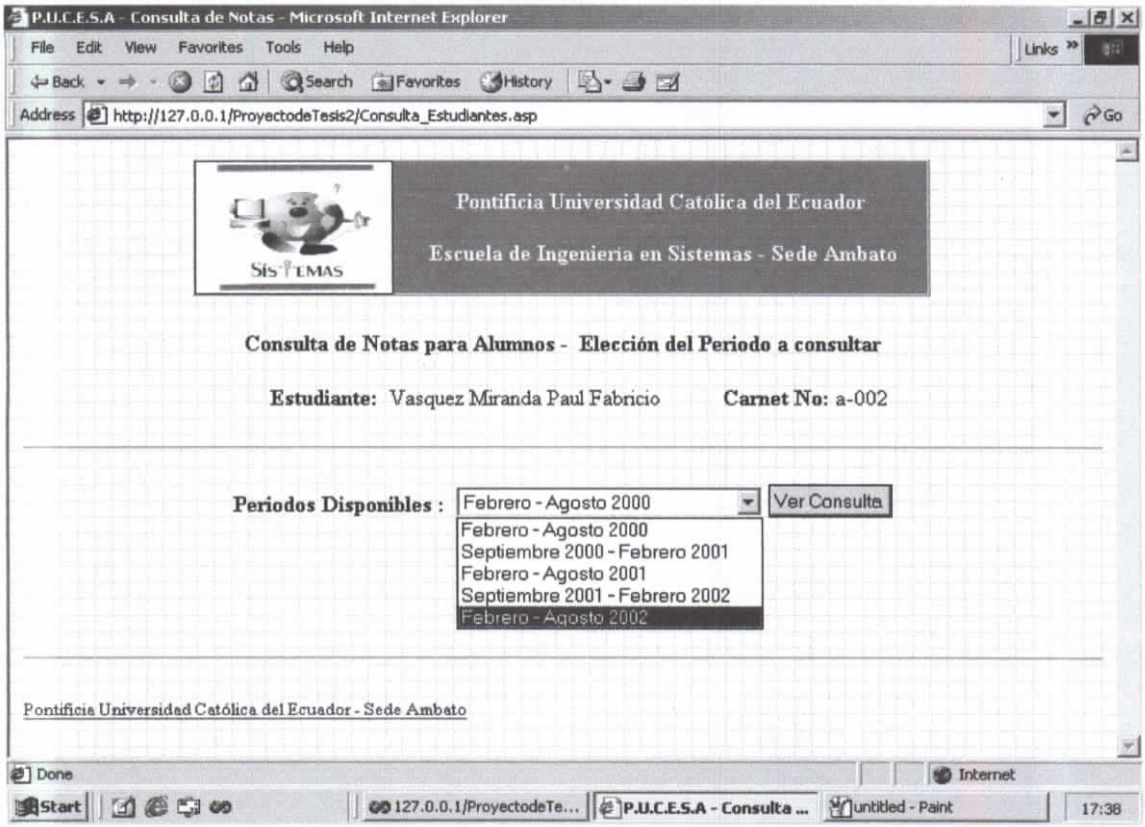
2. Pantallas del Módulo de "Consulta de Notas para Alumnos"

2.1 Pantalla de Ingreso de Estudiantes



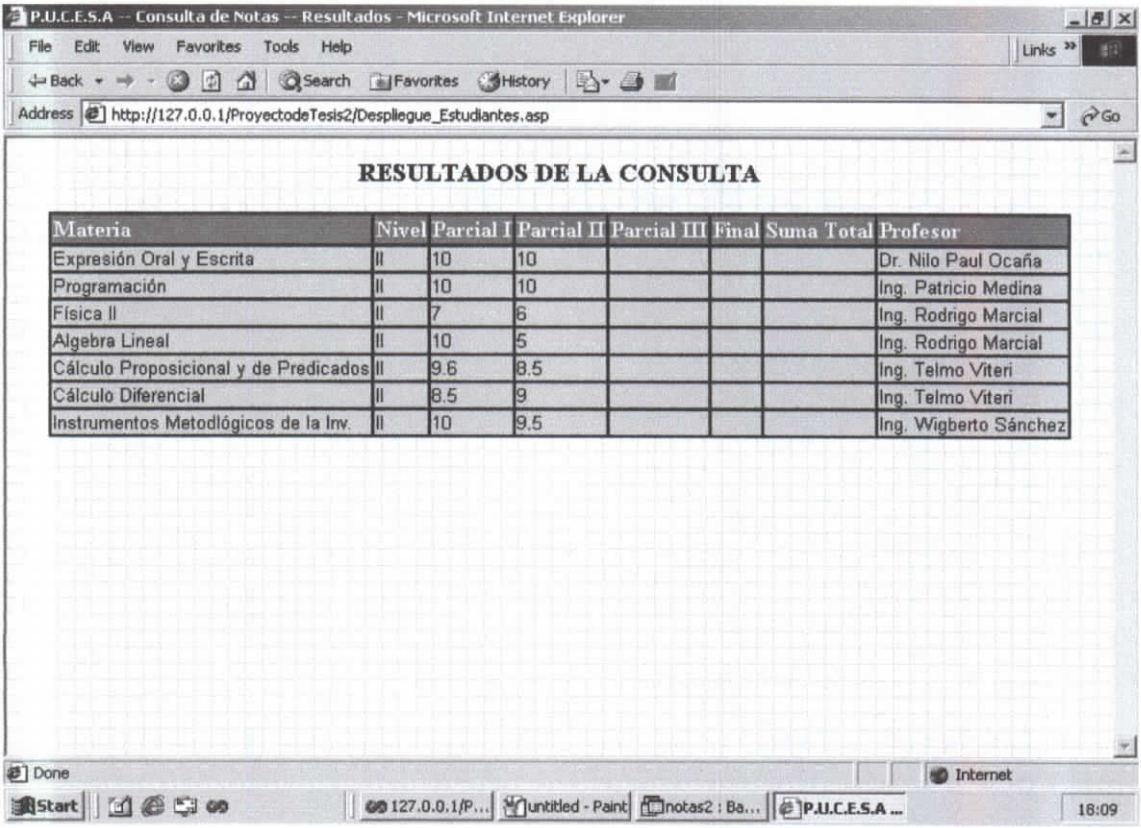
Esta pantalla, define el ingreso del alumno de una manera privada, a través del No. de Carnet y No. de Cédula del mismo, de esta manera, cualquier persona no puede entrar a observar notas que no son de su interés.

2.2 Pantalla de Elección del Periodo a consultar



Una vez que el estudiante a ingresado correctamente, la siguiente pantalla que se le presenta es la de elección de periodo, la cual le brinda la posibilidad de elegir el semestre (establecido como periodo), del cual desea observar sus registros de notas, es decir, el alumno tiene la facultad de consultar notas desde que inicio su carrera hasta notas del periodo en curso.

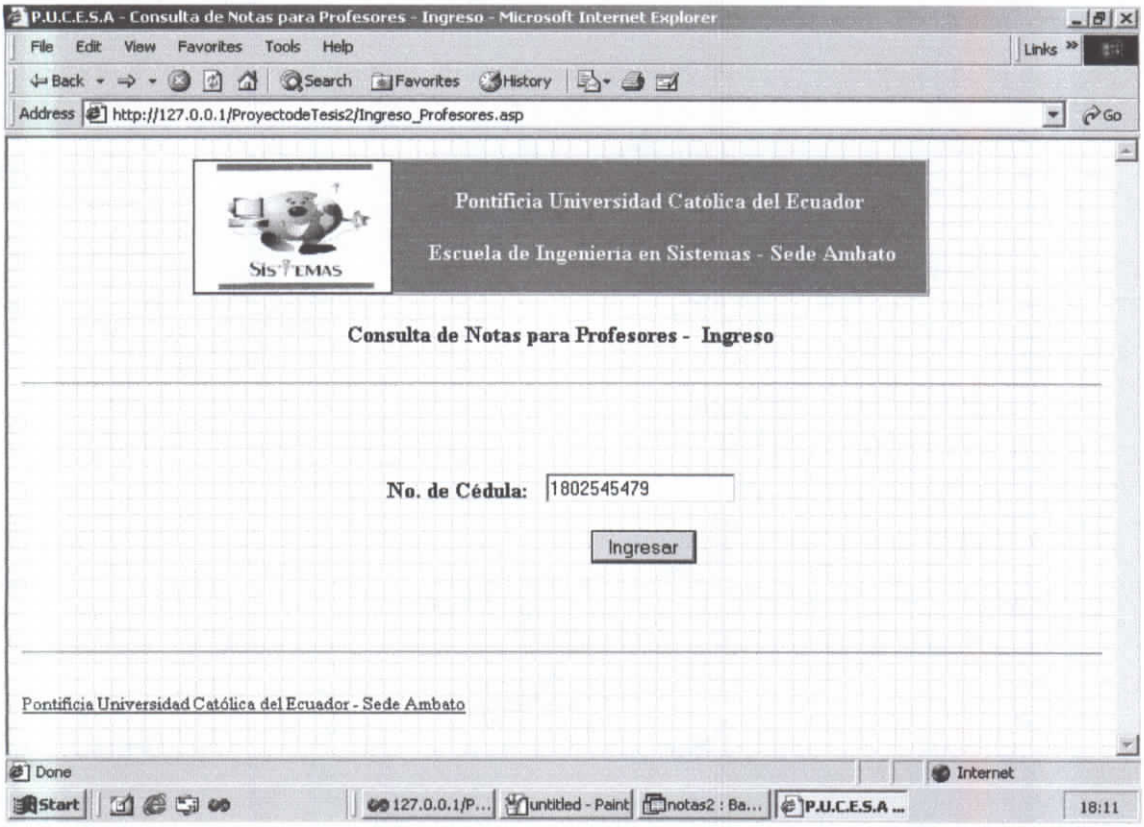
2.3 Pantalla de Despliegue de Notas



Luego de haber elegido el periodo del cual el alumno desea ver los registros de notas, se despliegan los mismos, detallando en éstos la materia, el nivel, los cuatro registros de notas, la suma total de notas y el nombre del profesor de la materia.

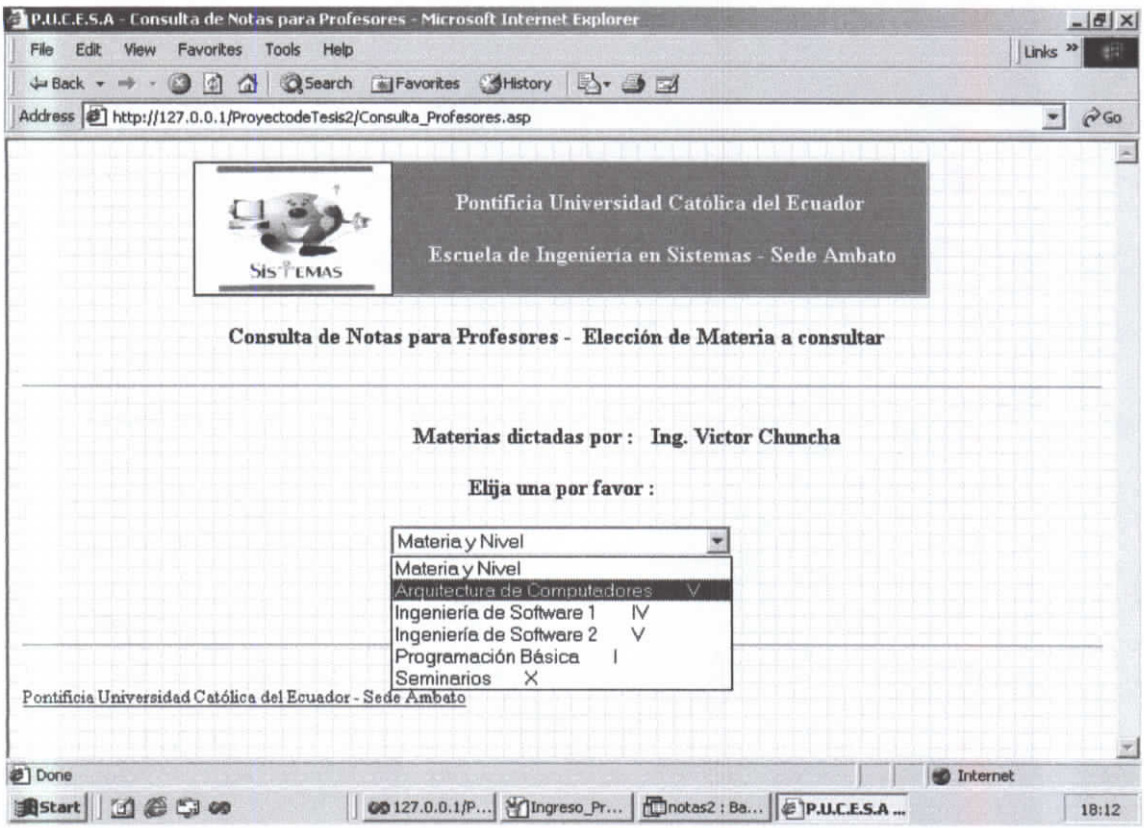
3. Pantallas del Módulo de "Consulta de Notas para Profesores"

3.1 Pantalla de Ingreso de Profesores



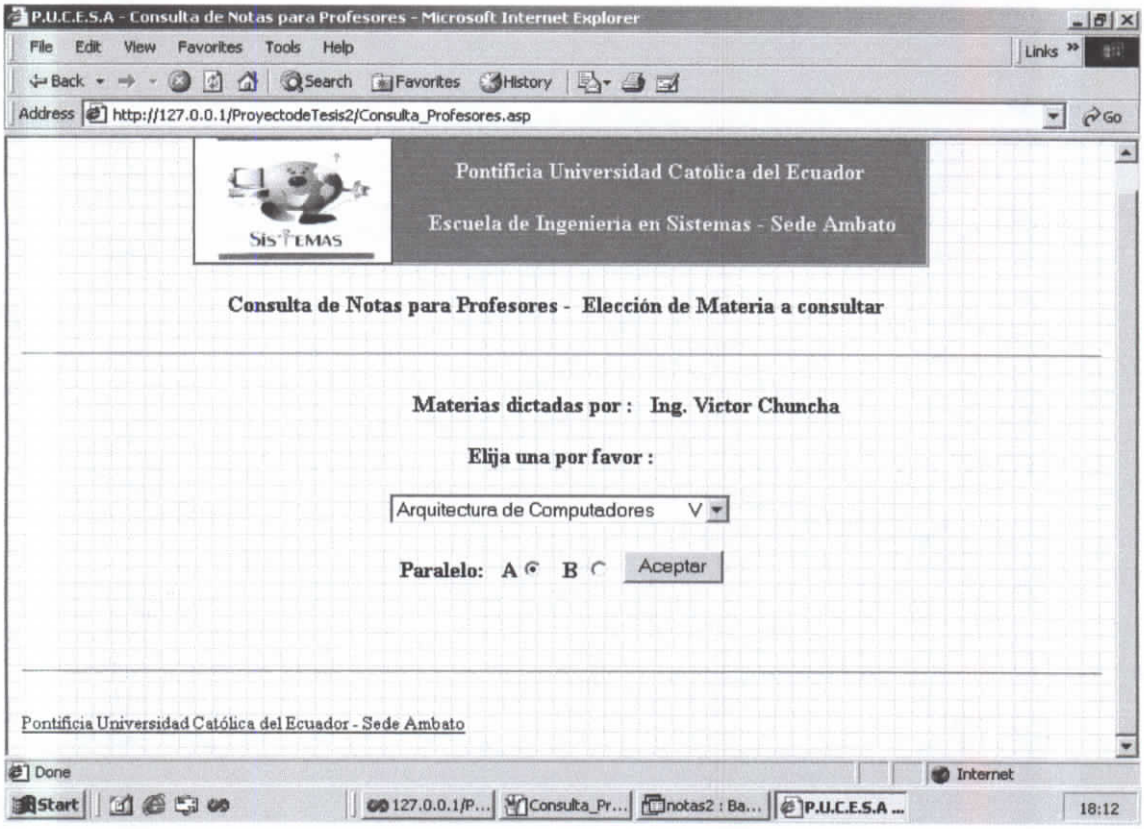
Esta pantalla permite el ingreso a los profesores de la Escuela de Sistemas, a través de la digitación de su número de cédula.

3.2 Pantalla de Elección de Materia

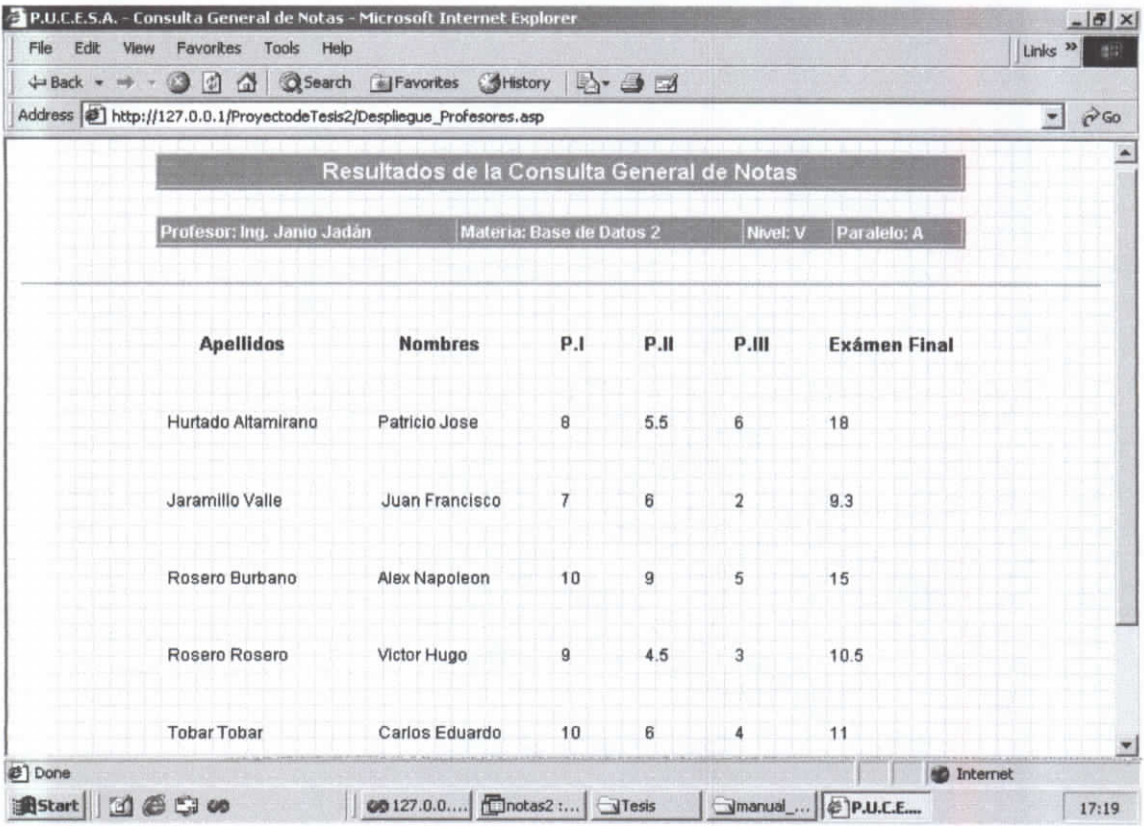


Luego de que el login del profesor ha sido correcto, se despliega la descripción de las materias que dicho profesor dicta en la Escuela de Sistemas, adjuntando también el nivel en el que se dicta cada materia, teniendo así la posibilidad de elegir una de ellas.

Se puede dar el caso de que el profesor escogió una materia que se dicta en dos o más paralelos, entonces, una vez que haya elegido la asignatura, se desplegará también opciones que incluyen los paralelos disponibles, así:


P.U.C.E.S.A - Consulta de Notas para Profesores - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Links
Back Search Favorites History
Address http://127.0.0.1/ProyectodeTesis2/Consulta_Profesores.asp
Go
SISTEMAS
Pontificia Universidad Católica del Ecuador
Escuela de Ingenieria en Sistemas - Sede Ambato
Consulta de Notas para Profesores - Elección de Materia a consultar
Materias dictadas por : Ing. Victor Chuncha
Elija una por favor :
Arquitectura de Computadores
Paralelo: A B
Aceptar
Pontificia Universidad Católica del Ecuador - Sede Ambato
Done
Internet
Start
127.0.0.1/P...
Consulta_Pr...
notas2 : Ba...
P.U.C.E.S.A ...
18:12

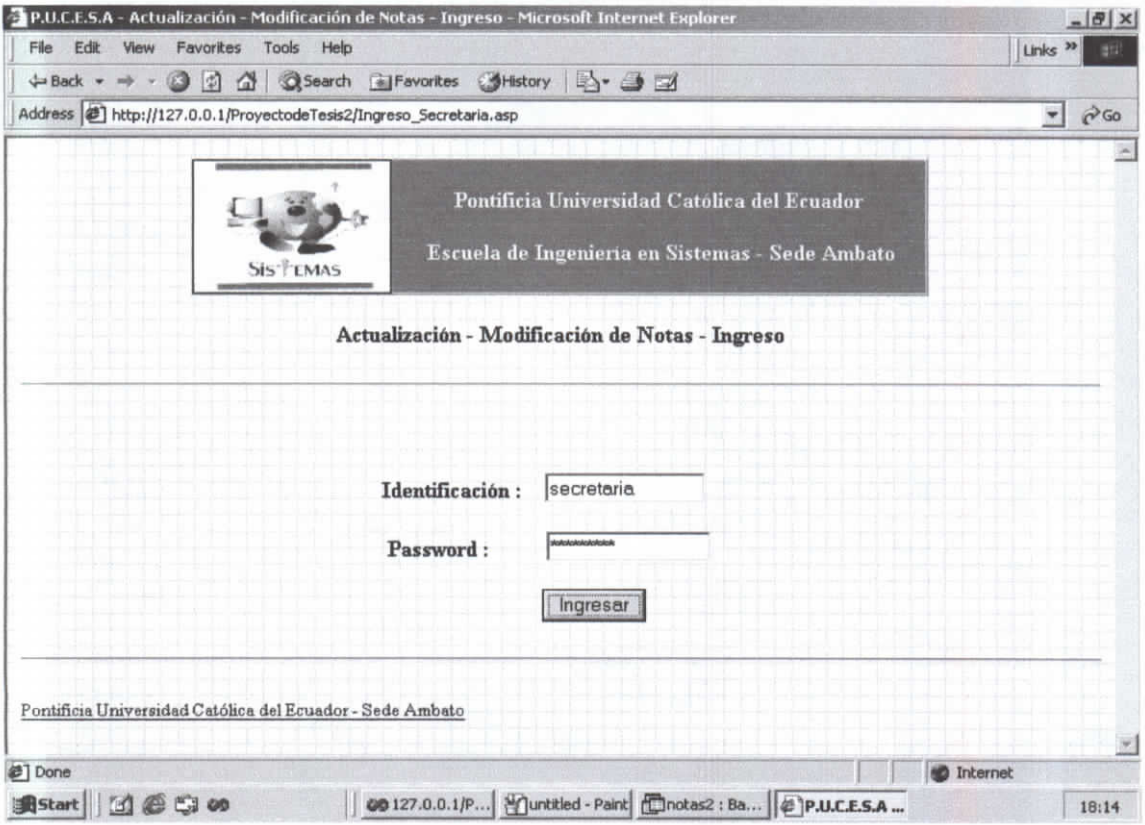
3.3 Pantalla de Despliegue de Notas de la consulta del profesor



Una vez que el profesor ha elegido la materia y el paralelo, del cual quiere observar las notas, estas son desplegadas, detallando apellidos, nombres, nota del parcial 1 (P.I), nota del parcial 2 (P.II), nota del parcial 3 (P.III) y la nota del exámen final, de cada estudiante matriculado en esa asignatura.

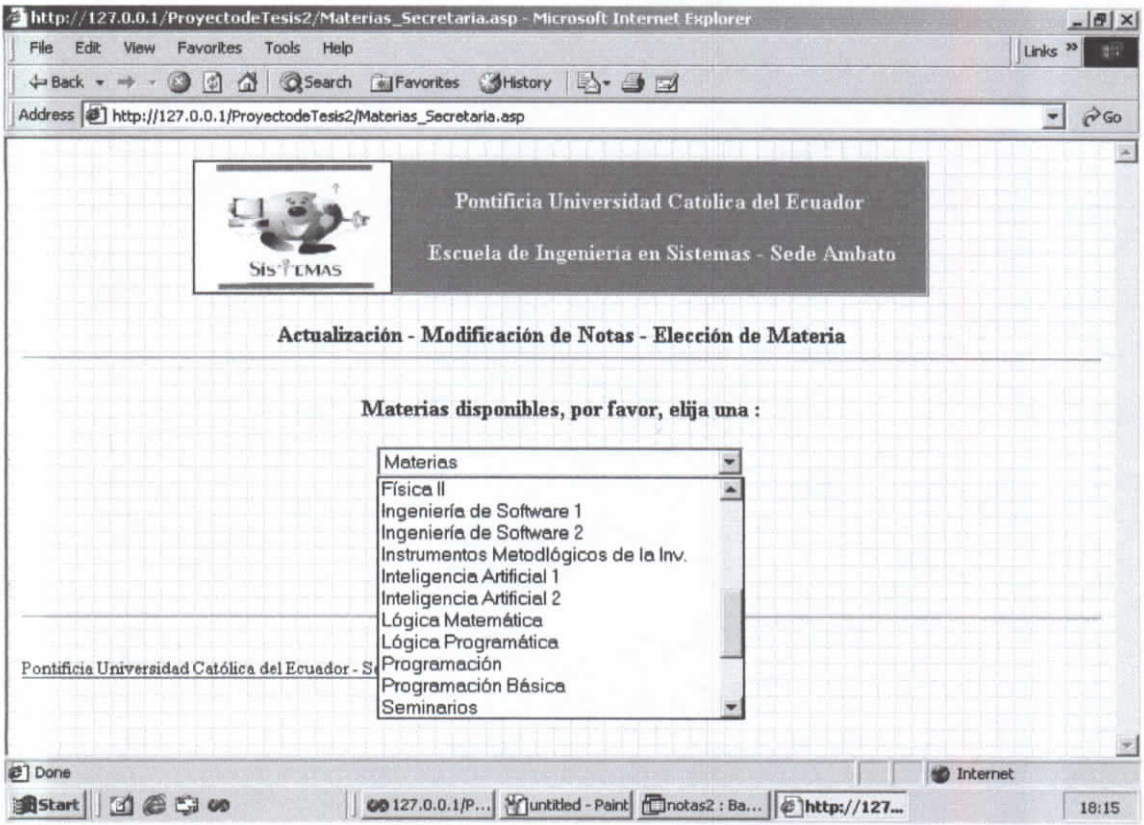
4. Pantallas del Módulo de "Actualización – Modificación de Notas"

4.1 Pantalla de Ingreso



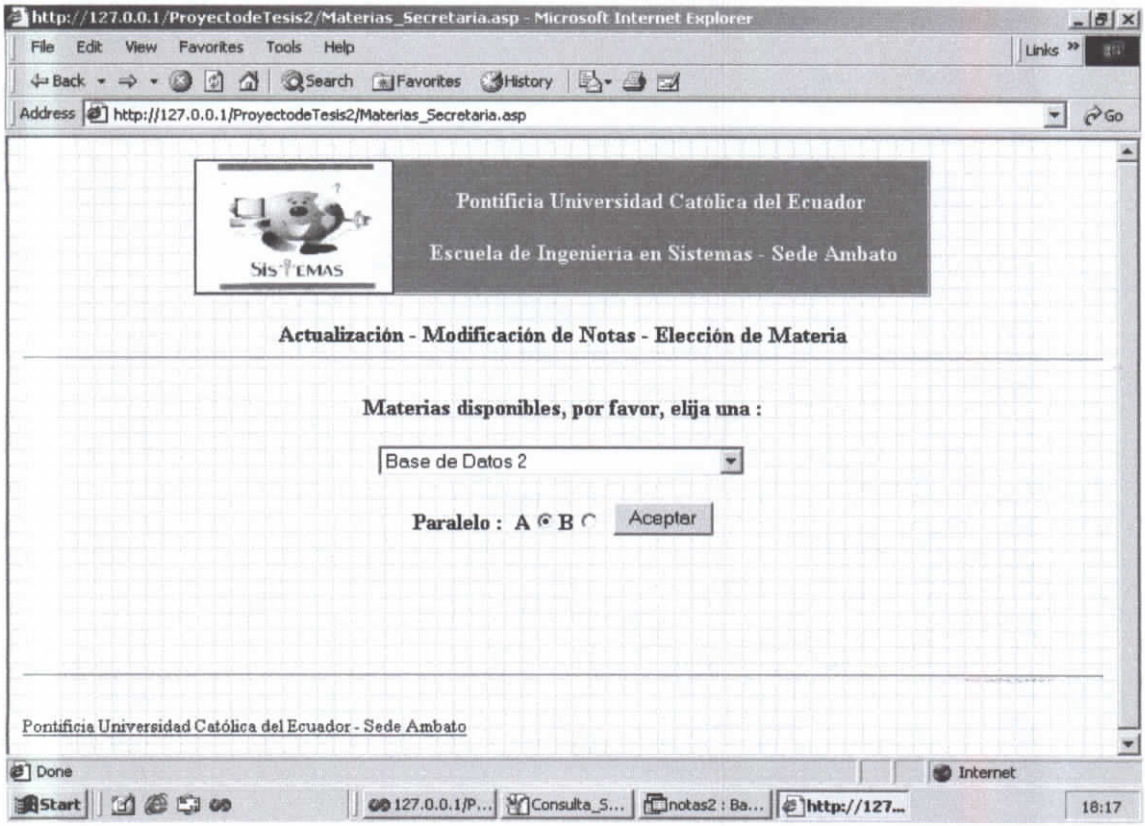
Ya que este módulo está dirigido al departamento de Secretaría, la persona responsable de los registros de notas, tendrá la facilidad de manipularlos a través de esta aplicación. Para esto, debe en primera instancia ingresar su identificación y su password, con el propósito de que únicamente dicha persona tenga acceso a las opciones de esta parte del sitio.

4.2 Pantalla de despliegue de todas las materias dictadas en la Escuela de Sistemas

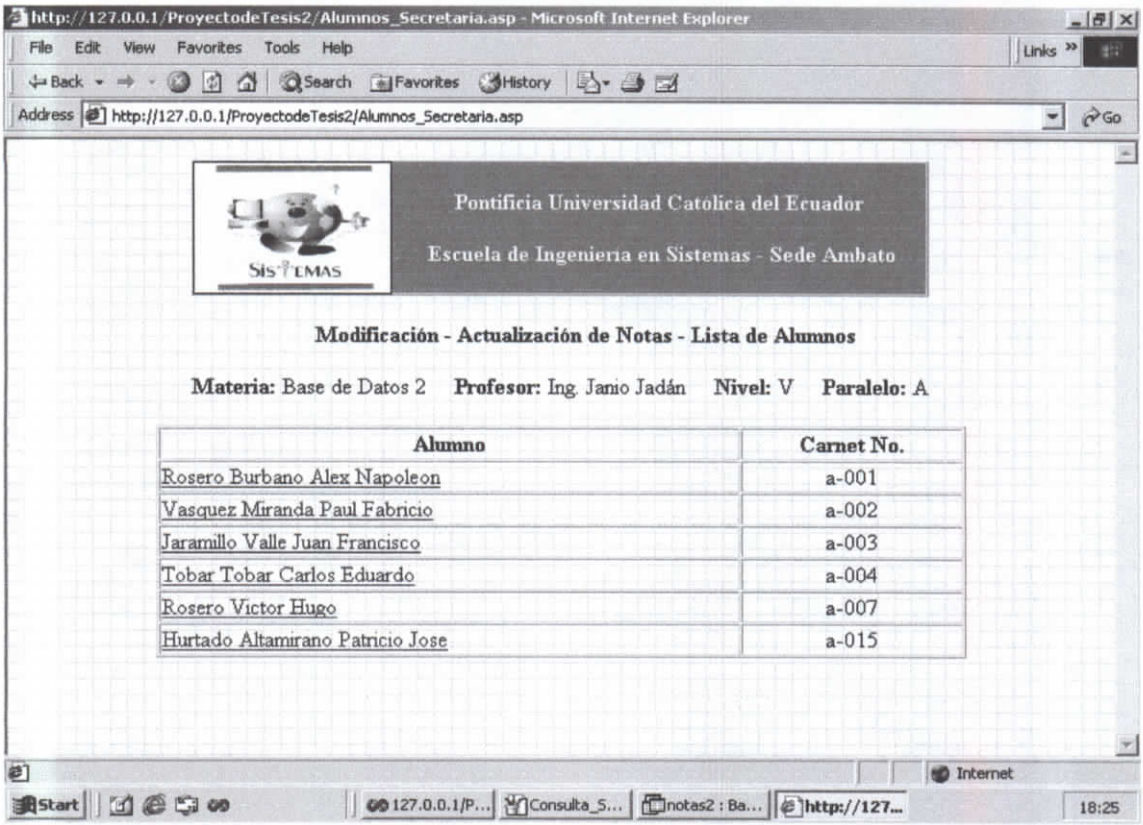


Una vez que la persona responsable ha digitado correctamente su identificación y su password, se presenta la pantalla de arriba, en la cual se despliegan todas las materias que se están dictando en el periodo en curso, posibilitando de esta forma la elección de una de ellas, para posteriormente tener acceso a todos los alumnos que están matriculados en esa materia.

Aquí también existe la posibilidad de que una materia se dicte en dos o más paralelos, por lo que es necesario verificar tal situación; si es el caso, se desplegará también un grupo de opciones con los paralelos disponibles, así:

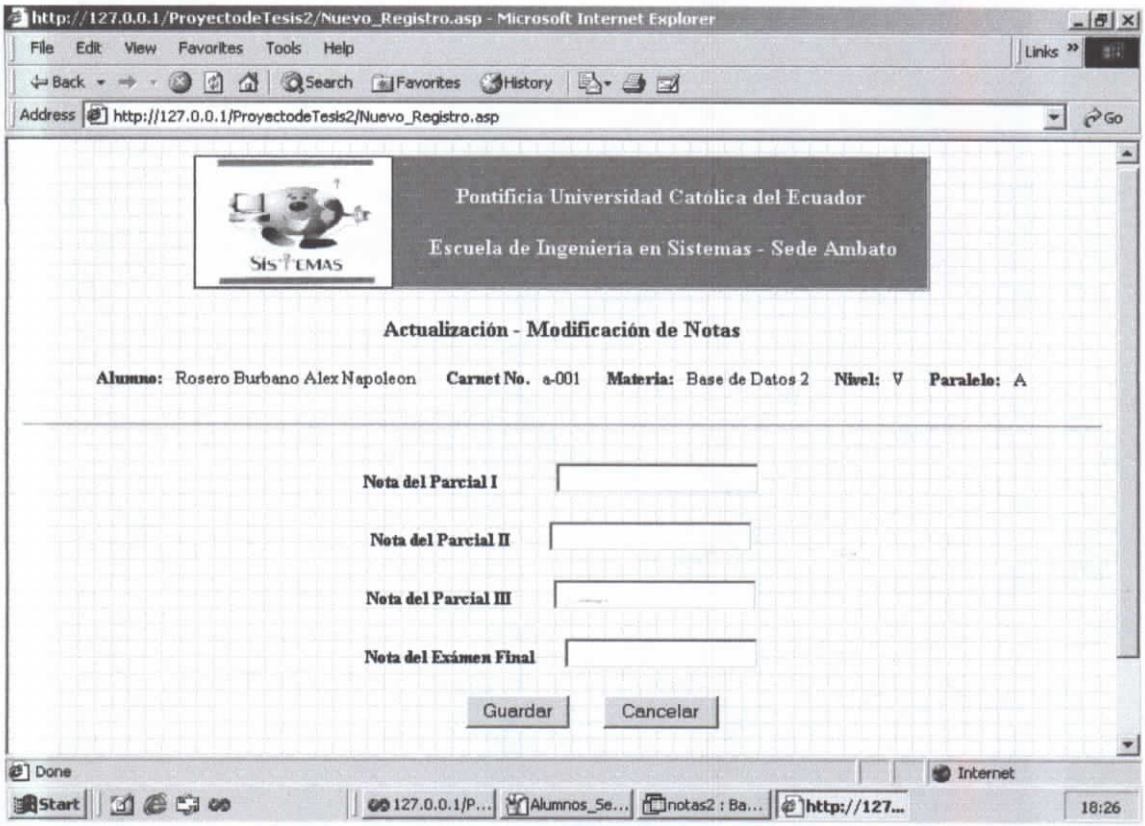

http://127.0.0.1/ProyectodeTesis2/Materias_Secretaria.asp - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://127.0.0.1/ProyectodeTesis2/Materias_Secretaria.asp
Go
Pontificia Universidad Católica del Ecuador
Escuela de Ingenieria en Sistemas - Sede Ambato
Actualización - Modificación de Notas - Elección de Materia
Materias disponibles, por favor, elija una :
Base de Datos 2
Paralelo : A B
Aceptar
Pontificia Universidad Católica del Ecuador - Sede Ambato
Done
Internet
Start
127.0.0.1/P...
Consulta_S...
notas2 : Ba...
http://127...
18:17

4.3 Pantalla de despliegue de todos los alumnos matriculados en una materia



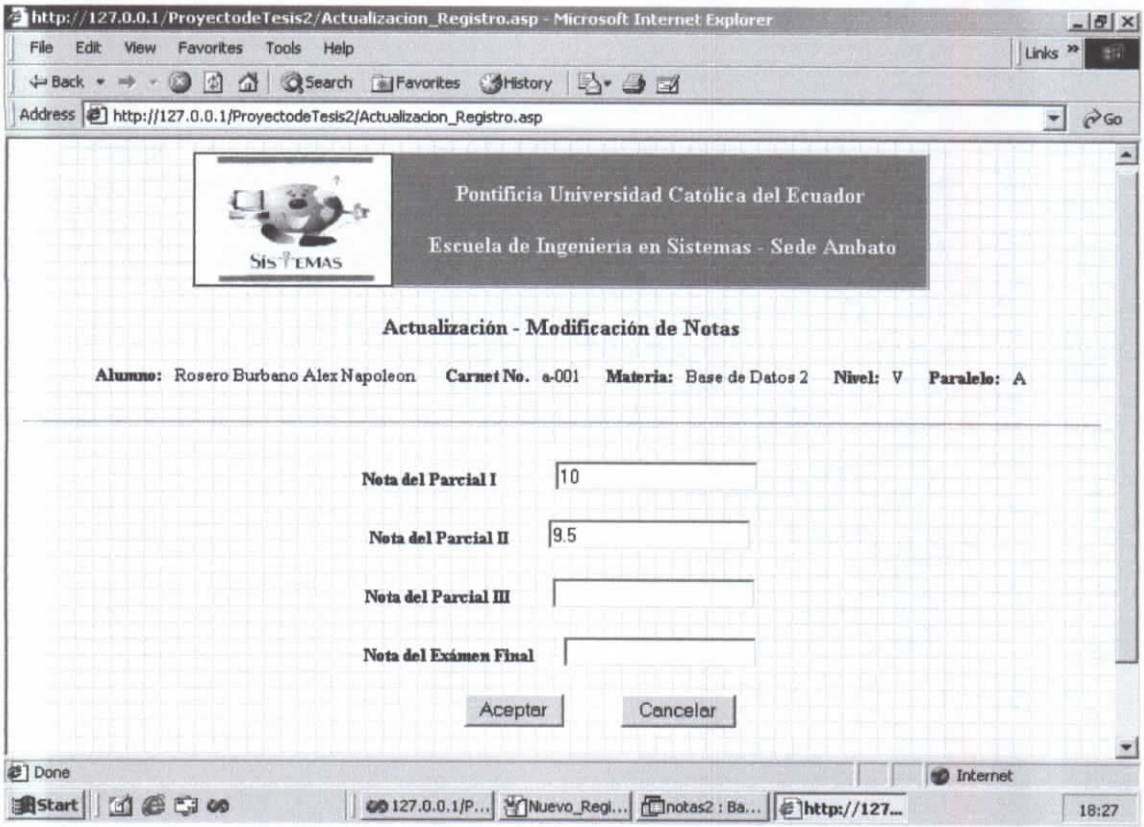
Luego de la elección de la materia, se despliega una tabla con todos los alumnos matriculados en dicha asignatura, teniendo cada uno de ellos, un enlace hacia su registro individual de notas.

4.4 Pantallas de Ingreso-Modificación de Notas



Una vez que se haya elegido el estudiante, del cual se desea tener acceso a sus registros, se presenta esta pantalla, que en el caso de que el alumno no tenga ningún registro de notas del semestre en curso, las cajas de texto que despliegan las diferentes notas, aparecerán en blanco. De esta forma la persona responsable del ingreso de notas, tendrá la capacidad de ingresar los registros de notas que sean necesarios.

Si el alumno elegido ya posee notas de alguno de los parciales, la pantalla se presenta de la siguiente manera:



En este caso, se podrá ingresar notas de los parciales restantes, o a su vez también se podrá modificar las notas ya existentes en la base de datos.

DICCIONARIO DE DATOS

ACTIVE X Tecnología desarrollada por Microsoft con el fin de elaborar aplicaciones exportables a la red las cuales deben ser capaces de operar sobre cualquier plataforma a través de navegadores WWW de forma que le da dinamismo a las páginas web.

API (Application Program Interface) Conjunto de convenciones de programación que definen cómo se invoca un servicio desde un programa.

APPLET Mini programa en lenguaje de programación Java integrado en una página web.

ASCII (American Standard Code of Information Interchange) Código Normalizado Estadounidense para el intercambio de la información. Código que permite definir caracteres alfanuméricos; se lo usa para lograr compatibilidad entre diversos procesadores de texto.

ASP (Active Server Page) Página de Servidor Activo. Tipo especial de página HTML la cual contiene pequeños programas (también llamados scripts) los cuales son ejecutados en servidores Microsoft Internet Information Server, antes de ser enviados al usuario, para su visualización en forma de página HTML. Habitualmente esos programas realizan consultas a bases de datos y los resultados de esas consultas determinan la información que se envía a cada usuario específico. Los archivos de este tipo llevan la extensión .asp.

BROWSER (Navegador, Visualizador, Visor) Aplicación para visualizar documentos WWW y navegar por el Internet. En su forma mas básica son aplicaciones hipertexto que facilitan la navegación por los servidores de información Internet; los mas avanzados cuentan con funcionalidades multimedia y permiten indistintamente la navegación por servidores WWW, FTP el acceso a grupos de noticias, la gestión del correo electrónico, etc.

CGI (Common Gateway Interface) Interfaz de intercambio de datos estándar en WWW a través del cual se organiza el envío y recepción de datos entre visualizadores y programas residentes en servidores WWW.

COOKIE Pequeño archivo de texto que un sitio web coloca en el disco rígido de una computadora que lo visita. Al mismo tiempo, recoge información sobre el usuario. Agiliza la navegación en el sitio. Su uso es controvertido, porque pone en riesgo la privacidad de los usuarios.

CSS (Cascade Stylesheet) Hojas de Estilo en Cascada. Archivos que proporcionan diseño de presentación a etiquetas, tanto de documentos HTML como XML.

DHTML (Dynamic Hypertext Transfer Metalenguaje) Variante del HTML que permite crear páginas web más animadas e interactivas.

DOM (Document Object Model) Modelo de Objetos del Documento. Interfaz de Programación de Aplicaciones (API) para documentos HTML y XML. Define la estructura lógica de los documentos y el modo en que se acceden y manipulan los mismos

DTD (Document Type Definition) Definición del Tipo de Documento. Archivo que define la gramática que los elementos de un documento XML deben cumplir a cabalidad.

FIREWALL Mecanismo de seguridad que impide el acceso a una red.

HARDWARE Parte física de la cual se compone un ordenador y sus periféricos.

HTML (Hypertext Transfer Metalenguaje). Lenguaje de Marcado de Hipertexto Lenguaje en el que se escriben los documentos que se acceden a través de visualizadores WWW. Admite componentes hipertexto y multimedia.

HTTP (Hypertext Transfer Protocol) Protocolo de Transmisión de Hipertexto Protocolo usado para la transferencia de documentos WWW.

INTERNET Red de redes. Sistema mundial de redes de computadoras interconectadas. Fue concebida a fines de la década de 1960 por el Departamento de Defensa de los Estados Unidos; más precisamente, por la ARPA. Se la llamó primero ARPAnet y fue pensada para cumplir funciones de investigación. Su uso se popularizó a partir de la creación de la World Wide Web. Actualmente es un espacio público utilizado por millones de personas en todo el mundo como herramienta de comunicación e información.

INTRANET Red propia de una organización, diseñada y desarrollada siguiendo los protocolos propios de Internet, en particular el protocolo TCP/IP. Puede tratarse de una red aislada, es decir no conectada a Internet

ISO (International Organization for Standardization). Organización de carácter voluntario fundada en 1946 que es responsable de la creación de estándares internacionales en muchas áreas, incluyendo la informática y las comunicaciones. Está formada por las organizaciones de normalización de sus 89 países miembros.

JAVA Lenguaje desarrollado por Sun para la elaboración de aplicaciones exportables a la red y capaces de operar sobre cualquier plataforma a través, normalmente, de visualizadores WWW.

KERNEL Núcleo o parte esencial de un sistema operativo. Provee los servicios básicos del resto del sistema.

LOGIN Nombre de usuario utilizado para obtener acceso a una computadora o a una red. A diferencia del password, login no es secreto, ya que generalmente es conocido por quien posibilita el acceso mediante este recurso.

MARKUP Conjunto de etiquetas con características propias, como las de HTML o independientes como las utilizadas en XML.

MOSAIC Visualizador WWW promovido por la NCSA. Fue el primero que tuvo funcionalidades multimedia y sentó las bases del modelo de publicación y difusión WWW.

MULTIMEDIA Información digitalizada que combina texto, gráficos, imagen fija y en movimiento, así como sonido.

NAMESPACE Colección de nombres que pueden ser usados como elementos o nombres de atributos en un documento XML

NETSCAPE Visualizador WWW creado por la empresa de ese mismo nombre y que en la actualidad se ha convertido en un estándar de facto en este tipo de aplicaciones Internet.

PARSER Aplicación que analiza la estructura de un documento XML, con la finalidad de que este conforme a las reglas sintácticas que lo rigen.

PASSWORD Palabra clave utilizada para obtener acceso a una computadora a una red o a un sistema. Un password generalmente contiene una combinación de números y letras que no tienen ninguna lógica.

PDF (Portable Document Format) Formato de archivo que captura un documento impreso y lo reproduce en su apariencia original. Los archivos PDF se crean con el programa Acrobat.

SCHEMA Documento Esquema. Extensión del XML, que brinda la posibilidad de establecer reglas gramaticales a los elementos de un documento XML.

SCRIPT Conjunto de secuencia de comandos dentro de páginas web, los cuales pueden realizar tareas específicas tanto en el cliente como en el servidor. Un Script puede ser escrito en diverso lenguajes como: Visual Basic Script, Java Script, etc.

SERVER Servidor. Sistema que proporciona recursos (por ejemplo, servidores de ficheros, servidores de nombres). In Internet este término se utiliza muy a menudo para designar a aquellos sistemas que proporcionan información a los usuarios de la red.

SGML (Standardized Generalized Markup Language) Lenguaje Estandarizado y Generalizado de Marcado. Estándar internacional para la definición de métodos de representación de texto en forma electrónica no ligados a ningún sistema ni a ningún dispositivo.

SOFTWARE Término general que designa los diversos tipos de programas usados en computación.

SQL (Structured Query Language) Lenguaje de programación que se utiliza para recuperar y actualizar la información contenida en una base de datos. Fue desarrollado en los años 70 por IBM. Se ha convertido en un estándar ISO y ANSI.

SSL (Secure Socket Layer) Sistema de seguridad en el cual los mensajes son encriptados de manera que solamente quien los emite y quien los recibe podrán descifrarlos.

URL/URI Universal Resource Locator/Universal Resource Identifier (Localizador Universal de Recursos/Identificador Universal de Recursos) Sistema unificado de identificación de recursos en la red. Las direcciones se componen de protocolo, FQDN y dirección local del documento dentro del servidor. Este tipo de direcciones permite identificar objetos WWW, Gopher, FTP, News.

UTF-16 Esquema de codificación de 16 bits. Es lo suficientemente grande para todos los caracteres de todos los alfabetos del mundo, con la excepción de idomas basados en ideogramas como el Chino. Todos los caracteres en UTF-16 usan 2 bytes. Un documento escrito en Inglés que usa UTF-16 es dos veces más grande que el mismo documento codificado con UTF-8. Sin embargo, los documentos escritos en otros idiomas, serán bastante más pequeños usando UTF-16.

UTF-8 Esquema de codificación de 8 bits. Los caracteres del alfabeto Inglés se utilizan usando bytes de 8 bits, y los caracteres de otros lenguajes usando 2, 3 o 4 bytes. Por lot anto produce documentos compactos para el lenguaje Ingkés, pero muy grandes para otros idiomas. Si la mayoría del documento está en Inglés, UTF-8 es una buena elección.

W3C (Consorcio W3) Organización cuyo cometido es el establecimiento de los estándares relacionados con WWW.

WEB (Malla, Telaraña) Servidor de información WWW. Se utiliza también para definir el universo WWW en su conjunto.

WORK DRAFT Publicación de un trabajo o estudio que aún no está completado, es decir el borrador de una recomendación ha darse a conocer.

WORLD WIDE WEB (WWW) (Malla Mundial) Sistema de información distribuido, con mecanismos de hipertexto creado por investigadores del CERN en Suiza. Los usuarios pueden crear, editar y visualizar documentos de hipertexto. Sus clientes y servidores puede accederse fácilmente.

XLINK Lenguaje de enlace. Permite introducir enlaces en un documento XML, de modo que se pueda enlazar archivos(documentos) entre sí.

XML (Extensible Markup Language) Lenguaje Extensible de Marcas. Lenguaje de meta-marcaje (meta-markup) que proporciona un formato para describir datos estructurados.

XPATH Lenguaje de búsqueda. Define la manera en que un nodo individual o un conjunto de nodos puede ser seleccionado en un documento XML.

XPOINTER Lenguaje de localización. Permite apuntar (localizar) elementos específicos de un documento XML.

XSL (Extensible Stylesheet Language) Lenguaje Extensible de Estilo. Lenguaje que además de brindar la posibilidad de dar estilo a documentos XML, posee la capacidad de transformarlos a otro formato (HTML,RTF,WML,etc)