UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

"Uso de XML para mejorar la presentación, el manejo y el intercambio de datos para el SIMIP en la SEC Veracruz"

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

INGENIERO EN COMPUTACIÓN

P R E S E N T A:

ARTURO ISAAC FERNÁNDEZ DEL CASTILLO MOLINA

DIRECTOR DE TESIS:

ING. SALVADOR PÉREZ VIRAMONTES

CIUDAD UNIVERSITARIA
MÉXICO, D.F. **2006**

"El presente es sólo un instante, prácticamente todo es pasado y futuro. Y es hurgando nuestra memoria (pasado) e imaginando el futuro como construimos nuestro presente."

Albert Einstein

Agradecimientos

A Dios:
Porque me ha dado la fortaleza, la salud y con ello la vida que sin ella me hubiera sido imposible concluir mi carrera profesional.

A mi familia, Mariana, Antonio y Marisol:
Porque con su apoyo, confianza y ánimo me motivaron a concluir este proyecto y aunque se que no se los di en tiempo si lo hice al menos en forma. Por toda su comprensión y cariño.

A mi madre:
Con gran agradecimiento por ser el motor de mis sueños, guía y apoyo incondicional para todos mis proyectos.

A mi padre:
Por ayudarme con sus consejos y mantenerme durante mi carrera.

A mi tía Pili:
A donde quiera que se encuentre por saber que siempre tuve frente a ella una figura cuyo apoyo moral lo tendré presente toda la vida.

A mi novia Claudia:
Por ser el motivo de mis nuevos proyectos y ambiciones, por su amor, comprensión y fomento a mi creatividad; y también por su apoyo en la presentación de esta tesis, que por cierto no le quedó nada mal.

A la UNAM:
Por ser el reflejo de los ideales de educación de nuestros héroes, dándole la oportunidad de estudiar a quien tiene simplemente el empeño de hacerlo.

A la Facultad de Ingeniería:
Por ser la segunda casa que albergó mis desvelos, mis motivos de superación, mis retos y que fue siempre testigo de mi creatividad y empeño, fortaleciéndome el am vital para enfrentar los caminos difíciles de la vida.

A mis profesores:
Agradezco sinceramente a todos y cada uno de ellos por su disposición, paciencia y confianza de transmitirme su conocimiento y experiencias, sin los cuales no sería el ingeniero que soy en este momento.

Al Ing. Salvador Pérez:
Por ser mi amigo y asesor al darme todas las facilidades para las revisiones periódicas de este trabajo, sus siempre atinados comentarios y sus correcciones.

Índice

Introducción 1

Capítulo 1 El lenguaje de marcado XML y sus ventajas 3
1.1 ¿Qué es el XML? 3
1.2 Ventajas de XML 4
1.3 Aplicaciones de XML 8
1.4 XML y HTML 12

Capítulo 2 Componentes de XML 13
2.1 Definición de elementos y atributos 13
2.2 Visualización de documentos básicos (Creación de documentos XML) 18
2.3 Creación de Documentos XML estructurados (estructura de DTD) 22
2.4 Esquemas XML como sucesores de los DTD 36
2.5 Entidades 37
2.6 Notaciones 48
2.7 Espacios de Nombres 50

Capítulo 3 Herramientas para la creación y validación de documentos XML ... 54
3.1 Herramientas de creación de documentos XML 54
3.2 Herramientas de validación de sintaxis XML y de validación de semántica DTD 55
3.3 XML y las hojas de estilo 59
3.3.2 Utilización de CSS 60
3.3.3 Aplicación de XSL 63

Capítulo 4 Antecedentes del Subsistema de Manejo de Trámites Administrativos (SIMIP) del Sistema de Personal de la SEC Veracruz. 70
4.1 Descripción del proceso administrativo de Operación de Trámites 71
4.2 Automatización de la Operación de Trámites 80
4.3 Manejo actual en ASPs con ADO-DB 82
4.4 Glosario de Términos 92

Capítulo 5 Aplicación de las técnicas XML para mejorar el sistema descrito ... 95
5.1 Ventajas de XML sobre la técnica basada en ASPs-ADODB 95
5.2 Herramientas XML de explotación de la BD del personal 96
5.3 Consultas automáticas en SQL Server usando XML 97
5.4 Explotación dinámica de contenidos XML mediante interfases API 105

Capítulo 6 Evaluación de Resultados 125
6.1 Definición de Criterios de Evaluación 125
6.2 Uso de criterios para evaluar los resultados obtenidos 134

Capítulo 7 Conclusiones 140
Bibliografía 141

Índice de Figuras

Capítulo 1

Figura 1.1 XML es independiente del medio 9
Figura 1.2 Estructura de un documento en XML (a) 10
Figura 1.3 Estructura de un documento en XML (b) 10
Figura 1.4 Estructura de un entorno Web 10

Capítulo 3

Figura 3.1 Interfaz de usuario de XML Notepad 54
Figura 3.2 Validación de un documento XML en Dreamweaver MX 55
Figura 3.3 Niveles básicos de dependencia entre componentes de Xerces 57
Figura 3.4 Herramientas de validación en Stylus Studio 5 Professional Edition 58
Figura 3.5 Uso de XSL para transformar un documento XML 64

Capítulo 4

Figura 4.1 Diagrama de operación de trámites en Ventanilla 73
Figura 4.2 Diagrama de operación de trámites en Análisis 74
Figura 4.3 Diagrama de operación de trámites en Plazas 76
Figura 4.4 Diagrama de operación para trámite normal (Informática-Análisis) 77
Figura 4.5 Diagrama de operación para Control y Seguimiento 79
Figura 4.6 Recepción de trámites en Ventanilla 84
Figura 4.7 Captura de propuestas 85
Figura 4.8 Módulo de Validación 86
Figura 4.9 Rutina de Encadenamiento 87
Figura 4.10 Validación de trámites que no están encadenados 88
Figura 4.11 Diagrama de Conexiones de Base de Datos del SIMIP 90

Capítulo 5

Figura 5.1 Procedimiento de exportación de datos usando el método EXPLICIT 99
Figura 5.2 Procedimiento de importación de datos utilizando DTS-MSXML 101
Figura 5.3 Funcionamiento de DOM con MSXML 106
Figura 5.4 Realización de movimientos sobre plazas y validación de las mismas 108
Figura 5.5 Pantalla de ingreso de datos al sistema 109
Figura 5.6 Verificación del movimiento cuando éste procede y cuando no lo hace 110
Figura 5.7 Modelo funcional del motor de validación en Java 111

Introducción

Actualmente los diversos equipos tanto de cómputo móvil como fijo incluyendo también a celulares, tienden a la convergencia dado el constante avance en la tecnología empleada en el enlace de voz y datos. La red de Internet fue pionera en la realización de este proyecto de integración por lo que cualquier servicio actual de red de datos o incluso de voz y/o vídeo tiene alguna relación con esta que se ha denominado la "Red de Redes".

Desde un inicio se han incorporado una serie de protocolos y estándares que le han dado forma a la manera de ver la Internet. XML ha incursionado en este campo por ser un lenguaje que viene a darle una nueva orientación en el sentido de un mejor manejo a la información que se tiene disponible y cuya variedad y contenido parece inagotable.

El lenguaje que se ha venido utilizando, el HTML, incorpora tanto la presentación como los datos que contiene, la mayoría sin orden ni contexto, haciéndose más complicado el proceso de búsqueda de información sobre las páginas. XML hace posible una mayor automatización al permitir el procesamiento de estos archivos (*.xml) con motores de búsqueda que pueden identificar datos con conceptos claros e incluso, prepararlos para su incorporación a bases de datos relacionales.

La orientación de este trabajo es por un lado la de proporcionar todas las bases teóricas en lo que respecta a XML haciendo referencias al trabajo realizado en la Secretaría de Educación y Cultura del Estado de Veracruz (SEC) y por otro la de presentar una propuesta de aplicación del lenguaje utilizando Bases de Datos e interfases de programación API en Java sobre un módulo del sistema implantado que a la vez ilustre la aplicación de los conceptos.

Contenido

Este trabajo consta de 7 capítulos, los cuales se pueden dividir en tres grupos:

1. Capítulos del 1 al 3. Descripción teórica de los fundamentos de XML.
2. Capítulos del 4 al 5. Descripción del Proceso de trámites y aplicación de herramientas y conceptos a módulo de Ventanilla de SIMIP.
3. Capítulos 6 y 7. Resultados de la Propuesta y conclusiones de la Tesis.

El capítulo 1 describe qué es el lenguaje XML, las ventajas y características que tiene dicho lenguaje, así como una breve comparación con HTML.

El capítulo 2 menciona a las elementos y atributos, a la vez que precisa las definiciones y componentes de cada uno y las jerarquías de los elementos. Define también los pasos recomendados para generar un documento XML que cumpla con la especificación. Realiza

una descripción de cómo construir una estructura DTD, tanto interna como externa que nos permita validar nuestro documento. Hace referencia a los elementos que nos permiten incluir objetos binarios y de texto mediante Entidades.

El capítulo 3 nos orienta sobre las herramientas que podemos utilizar para crear nuestro XML de manera más ágil y rápida, a la vez que nos muestra cómo éstas realizan la validación del mismo, ya sea en el momento que lo construimos, o a la hora de analizarlo o hacerle un *parsing*. Nos muestra también las capas sobre las que trabaja la API de Java Xerces. Para mejorar la presentación, nos da un vistazo de cómo hacerle para añadirle presentación a los documentos XML mediante CSS y XSL.

El capítulo 4 pone en antecedentes al lector acerca del Sistema Integral de Manejo de Trámites (SIMIP) para que pueda saber qué parte del proceso estamos abarcando y darle así la visión general del procedimiento para operación de trámites y facilitar de esta manera la comprensión del mismo. Incluye varios esquemas que ilustran el procedimiento de trámites y un Diagrama de Flujo de Datos sobre cómo se programó originalmente la captura de Ventanilla y su motor de validación. Al final del capítulo se incorpora un glosario de términos burocráticos de operación de trámites en la SEC.

El capítulo 5 describe las técnicas XML que se emplearon para la propuesta de mejora del SIMIP. Aplica método de importación / exportación a Bases de Datos e incorpora un ejemplo de aplicación utilizando Xerces como propuesta para incorporarle las ventajas de XML al Motor de Validación.

El capítulo 6 continúa con la evaluación de los conceptos aplicados en el capítulo anterior basándose en 8 criterios principales que nos dan las pautas con las cuales podemos evaluar el rendimiento del XML sobre aspectos del sistema de la SEC.

El capítulo 7 remata con una breve síntesis de lo visto en todo este trabajo, haciendo énfasis en sus puntos a favor y sus correspondientes relativos a los costos que hay que cubrir en su implementación.

Capítulo 1

El lenguaje de marcado XML y sus ventajas

1.1 ¿Qué es el XML?

Antes de describir lo que es el lenguaje XML es importante dar una idea general de su materia de trabajo que es la información, ya que es un lenguaje orientado a mejorar el manejo de la misma.

En breve, la información durante su ciclo de vida pasa por procesos que involucran:

- La concepción de la información que se desea obtener.
- Generación de nueva información a partir de datos base.
- Localización y almacenamiento
- Actualización de la información.
- Transmisión.
- Eliminación de información que ha dejado de ser útil.

El objetivo es siempre lograr un buen manejo de información, con características tales como coherencia (que esté relacionada), nivel de detalle preciso y que se pueda obtener de manera rápida.

Para lograr los objetivos planteados anteriormente y poniendo como base a las computadoras como medio para obtener un buen manejo de la información, debemos "quitar" el vicio de origen que conllevan los documentos informáticos actuales; este vicio radica en la forma en que se expresan, en las limitaciones que imponen los procesos informáticos que corren a su alrededor. Aumentando el nivel de interconectividad entre sistemas se puede mejorar el manejo y utilidad de la información.

XML (Extensible Markup Language) es una herramienta que puede ayudar a:

- Mejorar el manejo de información
- Crear nuevos procesos alrededor de la información
- Simplificar tareas asociadas con el manejo de información.

Conceptualmente, el XML es un estándar internacional desarrollado por un Grupo de Trabajo de XML (conocido como el Comité de Revisión Editorial de SGML) formado bajo el auspicio del World Wide Web Consortium (W3C) en 1996.

La recomendación dice textualmente ***"El Lenguaje extensible de Marcas, abreviado XML, describe una clase de objetos de datos llamados documentos XML y parcialmente describe el funcionamiento de programas de computador que pueden procesarlos.".***

Para entender correctamente el párrafo anterior, es necesario conocer lo que significan los siguientes términos:

- *"Marcas"*: son señales con un propósito definido que se añaden a un texto para ayudar a su procesamiento automático.

- *"Extensible"*: Se pueden definir las marcas requeridas para cada situación en particular, pudiendo definir tantas marcas como sean necesarias para abordar el problema presentado.

- La *"clase de objetos de datos"* se define como una gran cantidad de fuentes de información tales como Bases de Datos, documentos de texto, páginas Web, etc.

- *"Parcialmente describe"* quiere decir que XML se utiliza sólo para describir datos, XML no define el manejo de los mismos; sólo mediante aplicaciones informáticas específicas puede demostrarse el potencial de XML.

1.2 Ventajas de XML

- Independencia de los datos respecto de las aplicaciones.
- Información sobre la información (metainformación).
- Elementos para describir la estructura de un documento.
- Herramientas para organizar la información según nuestras necesidades.
- Formas de garantizar que los datos cumplen ciertas restricciones.
- Métodos para visualizar la información.
- Capacidad para pasar información a otras aplicaciones.
- Enlaces para datos relacionados.

A continuación se detalla cada una de estas ventajas:

1.2.1 Independencia de los datos respecto de las aplicaciones

Actualmente muchas de las aplicaciones con las que se trabaja mezclan la información de documentos con información de procesamiento propia de aplicaciones. Este hecho hace que sea difícil intercambiar información entre sistemas distintos.

Por ejemplo, a continuación se muestra un documento guardado con Microsoft Word, al abrirlo con un editor de MS-DOS, se muestra lo siguiente:

```
C:\docs>type curr.doc
ðø◄Óí▒
```

El concepto de *"Gestión de Información Abierta"* significa que las aplicaciones manejen la información de manera que esté disponible para cualquier programa y no solamente para la que la creó.

XML está especificado mediante un formato de documentos común normalizado e internacional, por lo que XML cumple con el principio de independencia de los datos. Lo anterior permite que la información que esté en dicho formato esté accesible para todo aquel que lo desee, no depende de ningún fabricante y no es necesario pagar por su uso. Además, XML está diseñado para contener documentos en formato internacional, ya que elige al estándar Unicode / ISO/IEC 10646 para la codificación de caracteres, con lo que los textos pueden estar escritos en cualquier idioma.

1.2.2 Información sobre la información, un contexto.

Un documento se compone de tres partes: semántica, estructura y presentación. Las etiquetas XML pueden utilizarse para marcar la estructura y la semántica de un documento.

En una aplicación de manejo documental, se tiene el siguiente texto:

"Factura No. 8314: Se hace constar que se recibieron los siguientes tres documentos: Hoja de filiación al IMSS, Orden de presentación, Acta de nacimiento. La factura es emitida por el Analista de ventanilla, Juan Pérez Cárdenas y la Recibe el interesado María Acosta López el 22 de agosto de 2005."

Al leer el texto una persona puede identificar los elementos de los que se compone la Constancia de Recepción de documentos; por el contrario, para un programa informático es algo sin estructura y sin contexto, y no haría más que copiarlo, mostrarlo o buscar una cadena de caracteres, pero no se puede procesar (explotar su contenido).

Incluyendo un poco de información adicional, es posible automatizar el proceso para búsquedas, filtrado de datos, reutilización de información, automatización de tareas, etc. En un texto como el siguiente se facilitan estas tareas:

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

<CONSTANCIA>
      Factura No.<NUMERO>8314</NUMERO>: Se hace constar que se recibieron
los siguientes <CANT_DOCUMENTOS>tres</CANT_DOCUMENTOS>documentos:
<DOC>Hoja de filiación al IMSS</DOC> <DOC>Orden de Presentación</DOC>
<DOC>Acta de Nacimiento</DOC>. Emite el <EMISOR>Analista de Ventanilla,
Juan Pérez Cárdenas</EMISOR> y recibe el <RECEPTOR> interesado María
Acosta López</RECEPTOR> el <FECHA_DE_EMISION>22 de agosto de 2005</FECHA
DE_EMISION>.
</CONSTANCIA>
```

1.2.3 Elementos para describir la estructura de un documento

XML puede utilizarse también para describir la estructura de un documento. En el texto anterior para un mejor entendimiento diremos que doc es la abreviación de documento y que dentro de la etiqueta **<doc>** y de la etiqueta **</doc>** el texto contiene información sobre el documento. Trabajar de esta manera es un poco confuso ya que no existe información sobre la estructura y todo está mezclado desordenadamente y lo que nos aporta el documento anterior sólo es su semántica.

Añadiendo estructura, organización e interrelación de componentes, el documento XML se organizaría de la siguiente forma:

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

<CONSTANCIA>
      <TITULO>Factura de Recepción de Documentos</TITULO>
      <NUMERO>8314</NUMERO>
      <DOCUMENTOS>
      La factura hace constar que se recibieron los siguientes tres
      documentos:
            Hoja de filiación al IMSS
            Orden de Presentación
            Acta de Nacimiento.
      </DOCUMENTOS>
      <FIRMANTES>
      Emite el Analista de Ventanilla, Juan Pérez Cárdenas y recibe el
      interesado María Acosta López
      </FIRMANTES>
      <FECHA DE EMISION>
       El día 22 de agosto de 2005.
      </FECHA>
</CONSTANCIA>
```

Con esta estructura la Constancia de Recepción de Documentos se compone de un título, un número de factura, Documentos Recibidos, Fecha de emisión y Firmantes de la Constancia. Las etiquetas como `<TITULO>`,`<NUMERO>`,`<DOCUMENTOS>`, etc., añaden información sobre la estructura pero aportan poca semántica.

En XML se pueden mezclar las dos (semántica y estructura) tal y como se muestra a continuación.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<CONSTANCIA>
      <NUMERO>8314</NUMERO>
      <DOCUMENTOS>
            <CANT DOCUMENTOS>3</CANT DOCUMENTOS>
            <DOCUMENTO>Hoja de Filiación al IMSS<DOCUMENTO>
            <DOCUMENTO>Orden de Presentación<DOCUMENTO>
            <DOCUMENTO>Acta de Nacimiento<DOCUMENTO>
      </DOCUMENTOS>
      <FIRMANTES>
        <EMISOR>
            <PUESTO>Analista de Ventanilla</PUESTO>
```

```
            <NOMBRE>Juan Pérez Cárdenas</NOMBRE>
        </EMISOR>
        <RECEPTOR>
            <TIPO>Interesado</TIPO>
            <NOMBRE>María Acosta López</NOMBRE>
        </RECEPTOR>
      </FIRMANTES>
      <FECHA DE EMISION>
            <DIA>22</DIA>
            <MES>AGOSTO</MES>
            <AÑO>2002</AÑO>
      </FECHA DE EMISION>
</CONSTANCIA>
```

Así, con la estructura anterior se pueden acelerar las búsquedas consultando sólo en una parte del documento.

1.2.4 Herramientas para organizar la información

XML permite organizar la información según se necesite, mediante diversas técnicas como las siguientes:

- Un documento se puede guardar en varios archivos, colocando una parte en uno y otra parte en otro.
- Se pueden nombrar cadenas de textos y mediante referencias a ellos incluidas en el documento, facilitar actualizaciones simultáneas.
- XML permite la integración de diversos formatos, gráficos, de vídeo, de sonido, etc., dentro de un mismo documento.

1.2.5 Formas de garantizar que los datos cumplen ciertas restricciones

XML nos permite verificar que el documento cumple con ciertas reglas como por ejemplo, la inclusión del número de la factura, la fecha de emisión, los firmantes, etc. En general, se pueden definir ciertas restricciones adicionales:

- Declarar las marcas que componen el documento.
- Definir ciertos elementos mínimos: La factura debe especificar la cantidad de documentos y la fecha de emisión.
- Se puede imponer una estructura determinada a los documentos XML para facilitar su lectura.

Lo anterior nos permite garantizar que la información de los documentos XML tiene una mínima calidad y que las operaciones que se realicen con ellos pueden tener éxito.

1.2.6 Métodos para visualizar la información

Proporcionan una forma para variar la presentación de la información según se requiera y capacidad para visualizar la información en función del contenido, es decir, de la semántica del texto.

Utilizamos estos procedimientos para mostrar la información, en el caso de las facturas colocamos el número de factura en negrita, la lista de documentos en cursiva, etc. No es necesario cambiar todas las facturas si se requiere en otro formato, sólo basta con modificarlo en un punto.

1.2.7 Paso de información a las aplicaciones

XML permite pasar instrucciones a las aplicaciones que procesan datos, esto se especifica por ejemplo en la primera línea del ejemplo, donde se indica a los procesadores XML la versión XML para la que se elaboró el documento y la codificación utilizada en el documento.

1.2.8 Enlaces para los datos relacionados

XML permite, al igual que HTML organizar la información de manera no lineal, enlazando las partes que tienen relación unas otras. En XML los enlaces están mejorados con respecto al HTML.

1.3 Aplicaciones de XML

1.3.1 Clasificación de aplicaciones según el consumidor de información

Las aplicaciones de XML se clasifican dentro de los siguientes dos tipos:

1. **Aplicaciones de proceso de documentos** las cuales manipulan información que está orientada principalmente para consumo humano.
2. **Aplicaciones de proceso de información** las cuales manipulan información que está orientada para consumo de software.

Aunque ambos tipos de aplicación utilizan el mismo estándar XML y se implementan mediante las mismas herramientas, tienen metas distintas. Esto es importante porque quiere decir que se pueden reutilizar herramientas y experiencia en un número grande de aplicaciones.

1.3.1.1 Aplicaciones de Proceso de Documentos

La primera aplicación de XML sería la publicación de documentos. La ventaja de XML en este rubro es que XML se concentra en la estructura del documento y esto lo hace independiente del medio de entrega.

Por tanto, es posible editar y mantener documentos en XML y publicarlos automáticamente en medios distintos.

Es importante tener la habilidad de manejar múltiples medios porque muchas publicaciones se encuentran disponibles tanto en línea como en formato impreso. La Web

está cambiando rápidamente también y es necesario estar reformateando el sitio constantemente.

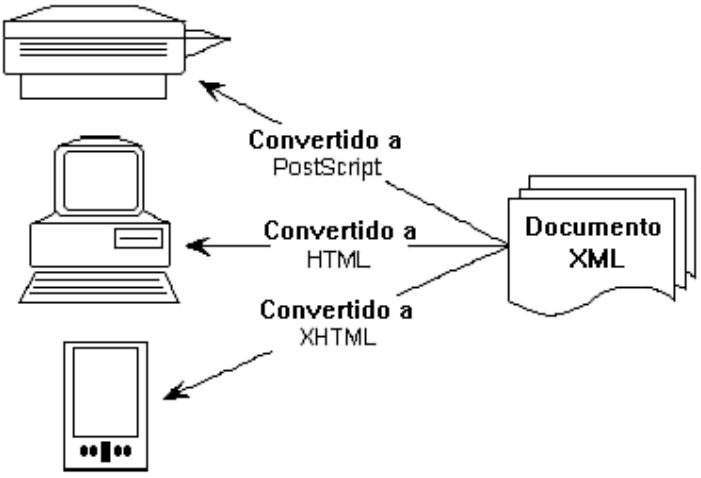


Fig. 1.1.- *XML es independiente del medio.*

Los sitios Web están optimizados para navegadores específicos, tal como Netscape o Internet Explorer. Esto nos conduce al desarrollo de dos o más versiones del mismo sitio; si esto se hace manualmente, resulta muy costoso.

Por estas razones, tiene sentido mantener una versión común de la documentación en un formato independiente del medio, tal como XML, y así convertirlo automáticamente en formatos de publicación como HTML, PostScript, PDF, RTF y más.

1.3.1.2 Aplicaciones de Proceso de Información

Una de las principales metas de SGML era proporcionar a la administración de documentos, acceso a las herramientas de software que se han utilizado para administrar información, tales como las bases de datos que conocemos en la actualidad.

XML cierra el círculo porque proporciona una forma de distribución de información. Esto nos lleva al concepto de que la aplicación es el documento en sí, y en donde no existe diferencia alguna entre documentos y aplicaciones.

Podemos observar que la estructura de una base de datos puede expresarse en XML en forma semejante a la estructura de un documento, tal como se ilustra en la Figura 1.3.

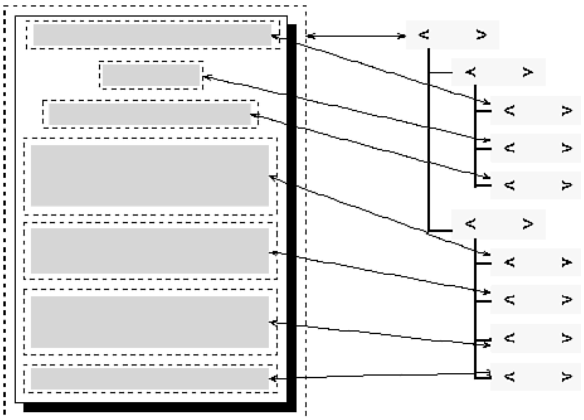


Figura 1.2 *Estructura de un documento en XML (a).*

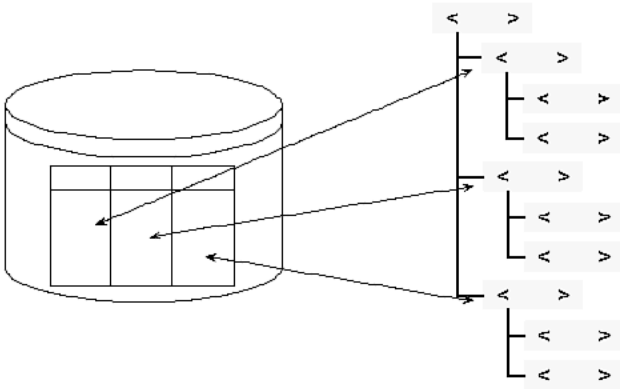


Figura 1.3. *Estructura de un documento en XML (b).*

En este contexto, XML se usa para intercambiar información entre organizaciones. El Web basado en XML es una base de datos amplia sobre la cual las aplicaciones pueden actuar.

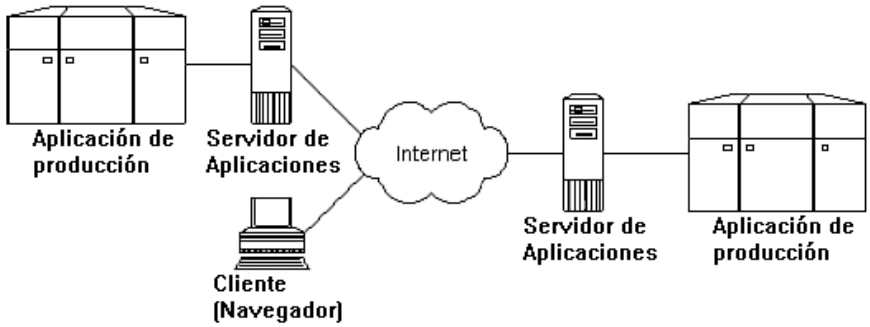


Figura 1.4. *Estructura de un entorno Web.*

Podría observarse esto como una extensión a las extranets, dado que el objetivo de una extranet es que una organización publique parte de su información en la Web para sus asociados.

En el caso particular de la SEC-Veracruz (Secretaría de Educación y Cultura del Estado de Veracruz), el SIMIP (Sistema Integral de Movimientos e Incidencias de Personal) extiende su alcance a Nóminas. Los Avisos de Movimientos de Personal (AMP) generados en el Área de Recursos Humanos son dinámicos, ya que se crean día con día. Con la información disponible en un sitio web, el Área de Nóminas puede obtener la información generada recientemente.

Actualmente, la lista de AMPs (Avisos de Movimiento de Personal) se publica en HTML y está hecha para ser vista por personas. Esto sería aceptable para una pequeña organización o empresa con pocos empleados pero en el caso de la SEC se requiere de una solución automatizada.

Con XML, cualquier software (incluyendo el sistema en Smalltalk que tiene el Área de Nóminas operando actualmente) puede visitar automáticamente la lista de AMPs generados, extraer los datos del movimiento y actualizar la información en su propia base de datos. Esto se muestra en la parte correspondiente a la pareja Aplicación de Producción-Servidor de Aplicaciones de la Figura 1.4.

1.3.2 Aplicaciones Prácticas de XML

Las ventajas que nos ofrecen las computadoras, tales como búsquedas, cálculos, organización de información ya las conocemos, sin embargo XML nos permite explotarlas más. Algunas aplicaciones prácticas que ejemplifican lo anterior son las siguientes:

a) *Búsquedas de información*.- Con XML se pueden realizar búsquedas específicas tales como las facturas expedidas en un día determinado, por cierta persona, etc.
b) *Procesos Inteligentes*.- Los documentos XML nos pueden arrojar información de acuerdo con el caso específico que estamos tratando, hacer estadísticas, etc. En este ejemplo se pueden saber cuántos trámites recibió el Analista de la Ventanilla Juan Pérez Cárdenas en el lapso de un día.
c) *Cambios de medio*.- No importando en el medio que se vaya a almacenar la información, XML permite tener un único archivo fuente que se puede transformar al medio que se destine, es decir, se puede generar un único documento fuente, archivos HTML en distintas versiones, etc.
d) *Personalización.*- Se puede definir qué usuarios pueden tener acceso a la información o qué tipos de documentos son los que el usuario recibe más frecuentemente para eficientar el proceso de captura. Otra forma de personalización es la forma en que cada usuario visualiza los datos.
e) *Información reutilizable.*- Para evitar rescribir todo un texto, se puede extraer información relevante para generar nueva información, por ejemplo, en el caso de rechazos se pueden extraer los documentos que se proporcionaron por parte del interesado y elaborar una factura de rechazo indicando los documentos que se entregan.
f) *Automatización.*- Con XML se pueden explotar los datos almacenados y hacer cálculos o elaborar nuevos documentos automáticos con la información almacenada. Mediante procedimientos SQL generados por una aplicación se pueden manipular datos en una BD con la información extraída del documento XML.

1.4 XML y HTML

HTML es un lenguaje de marcado extremadamente popular. De acuerdo con algunos estudios, existen más de 800 millones de páginas Web y todas éstas se basan en HTML. Existe un gran número de herramientas que trabajan con él e incluyen navegadores, editores, software de correo electrónico, etc.

A través del tiempo, el HTML se ha extendido y con ello se han incorporado nuevas etiquetas. La primera versión de HTML tenía una docena de ellas; la última versión (HTML 4.0) tiene cerca de 100 etiquetas (sin contar las etiquetas específicas de cada navegador).

Sin embargo, no todo está bien con HTML. Este ha evolucionado hasta llegar a convertirse en un lenguaje complejo; con casi 100 etiquetas es en definitiva un lenguaje muy distinto al que fue originalmente. Las combinaciones de etiquetas se vuelven casi interminable con el resultado desfavorable de que una combinación de ellas puede producir resultados distintos de un navegador a otro.

Además de todas las etiquetas ya incluidas en el HTML, se requieren más. Las aplicaciones de comercio electrónico necesitan etiquetas para referencias de productos, precios, nombres, direcciones y más. Las aplicaciones de flujo de información necesitan etiquetas para controlar el flujo de las imágenes y el sonido. La lista de aplicaciones que necesitan nuevas etiquetas HTML es casi interminable.

Podemos observar contradictoriamente, que mientras algunas aplicaciones requieren de más etiquetas, algunas otras se beneficiarían en gran medida si hubiera un menor número de ellas en HTML. Cada vez más gente está accesando información en la Red utilizando un PDA (Personal Digital Assistant), un asistente digital personal, tal como la popular PalmPilot, o desde los llamados teléfonos inteligentes (smart phones).

Otro problema es que se requiere de varios tags para formatear una página. Es común observar páginas que llevan consigo más código de marcado que contenido.

Sintetizando, aún cuando HTML es un lenguaje popular y exitoso, tiene algunas limitaciones considerables. XML fue diseñado para hacer frente a estas limitaciones.

XML existe porque HTML tuvo éxito. Por tanto, XML incorpora algunas características acertadas de HTML. XML existe también porque el HTML de hoy en día no pudo competir con las nuevas demandas que el continuo avance del Internet está exigiendo.

2.1 Definición de elementos y atributos

2.1.1 Definición de Elementos

Un elemento es "la unidad lógica con capacidad para representar la estructura lógica y la semántica de un documento XML, en definitiva, su contenido".

Los elementos son en sí las piezas básicas de un documento XML y aportan información acerca de la estructura y el significado de los diversos componentes de un documento.

```
<CONSTANCIA>
...
</CONSTANCIA>
```

En este ejemplo el documento XML es muy reducido y define un único elemento, <Constancia>, mediante el cual podemos saber que el documento contiene una constancia y podría diferenciarse de otros elementos como Relaciones, Reportes, etc.

Como la información que provee este elemento no es suficiente, se le pueden añadir otros elementos al documento original.

```
<CONSTANCIA>
      <NUMERO>8314</NUMERO>
      <DOCUMENTOS>
            <DOCUMENTO>Acta de Nacimiento</DOCUMENTO>
            <DOCUMENTO>Perfil de Puesto</DOCUMENTO>
      </DOCUMENTOS>
...
</CONSTANCIA>
```

En este ejemplo, existen nuevos elementos que representan el `<NUMERO>`, y `<DOCUMENTOS>` de una constancia; al explotar el contenido se podría buscar los números de constancia que contengan una Acta de Nacimiento.

2.1.1.1 Partes que conforman un elemento

Los elementos están formados de tres componentes, una etiqueta de inicio que identifica al elemento, por ejemplo "`<CONSTANCIA>`", una etiqueta de finalización, en este caso "`</CONSTANCIA>`", y un contenido situado entre ambas etiquetas y que en este caso trata de una constancia de recepción de documentos. A diferencia de HTML, todos los elementos están formados con estas tres partes, siendo las etiquetas sólo una parte del elemento.

2.1.1.2 El elemento raíz

Volviendo al ejemplo de partida, podemos observar que "Constancia" es un elemento especial y agrupa todo el contenido del documento. Por esa razón "Constancia" recibe el nombre de "raíz"; dicho de otro modo, Constancia es el elemento raíz del documento. El elemento raíz también recibe el nombre de elemento documento porque incluye a todo el contenido del documento.

Todo documento XML forzosamente tiene que contener un elemento raíz, si no es así no es un documento XML bien formado.

En los documentos HTML, el elemento raíz es <html>; los documentos XML no pueden utilizar ningún elemento con la etiqueta <xml>, ya que la cadena XML está reservada.

2.1.1.3 La Declaración XML

La declaración XML es opcional y sirve para definir la versión de XML empleada para crear el documento. Es muy conveniente incluir la declaración XML que permita señalar estos parámetros los cuales dan la pauta para que nuestro documento pueda interpretarse correctamente.

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

2.1.1.4 Usos de los elementos

Ya se había mencionado que los elementos se utilizan para describir la estructura y la semántica de un documento. XML es flexible y puede emplearse para abarcar también la funcionalidad del HTML. Atendiendo a esto, XML puede tener distintos usos como los siguientes:

Elementos orientados a la presentación

Los elementos XML pueden usarse para funciones de presentación (Ej: negrita, itálica, cursiva, etc.)

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<CONSTANCIA>
      <Negrita>Factura de Recepción de Documentos</Negrita>
      <Cursiva>8314</Cursiva>
</CONSTANCIA>
```

El uso de elementos para fines de presentación debe evitarse ya que desvirtúa la característica esencial de XML de orientación hacia contenido.

Delimitación del contenido

En el siguiente ejemplo tenemos una gran cantidad de trámites especificados en los Avisos de Movimiento de Personal (AMP) y cada uno se guarda en un documento XML como el siguiente:

```
<?xml version=”1.0” encoding=”ISO-8859-1” ?>
<AMP>
<Dependencia> Secretaría de Educación y Cultura </Dependencia>
<Movimiento> Alta – Cambio con modificación de percepciones </Movimiento>
El movimiento se encadena a la Baja anterior con Folio 328.
</AMP>
```

En caso de que alguien conociera el nombre de la dependencia, en este caso la “Secretaría de Educación y Cultura”, su búsqueda sería fácil, pero en el caso de que quisiera solamente mostrar los movimientos a los que está encadenado no sería lo mismo, porque no hay definición explícita de elementos a movimientos encadenados.

Los elementos se pueden utilizar para delimitar las distintas partes de un documento, separándolas y facilitando su manejo informático. Aportan información estructural al documento y esto ayuda cuando se requiera buscar la información en una parte específica del documento. El problema anterior se resuelve definiendo un elemento que identifique el movimiento encadenado y lo aísle del resto del texto, como se muestra a continuación:

```
<?xml version=”1.0” encoding=”ISO-8859-1” ?>
<AMP>
<Dependencia> Secretaría de Educación y Cultura </Dependencia>
<Movimiento> Alta – Cambio con modificación de percepciones </Movimiento>
El movimiento de encadena a la Baja anterior con Folio.
<encadenado>328</encadenado>
</AMP>
```

2.1.1.5 Unidades con significado propio e indivisible

Los elementos anteriores engloban entre sus etiquetas textos con un significado unitario podrían ser utilizados en búsquedas o diversos procesos ya que si son muy extensos o se pueden dividir en más componentes se dificultaría su utilización. Este marcado es de tipo semántico ya que define el significado de la cadena de texto.

2.1.1.6 Agrupamiento de elementos

Los elementos también se pueden emplear para agrupar otros elementos bajo una misma etiqueta. Los elementos se van agrupando mediante estructuras más globales; en los ejemplos anteriores pueden verse ejemplos de agrupamiento.

Especificar un contexto

Los elementos se emplean para especificar un contexto que describa la idea general sobre la que se aplica la información. Por tanto si en nuestro ejemplo, existe un elemento movimiento que describe el trámite de una persona, se tiene que proporcionar otro elemento que nos diga que es un movimiento de personal de la Secretaría de Educación y Cultura.

Los elementos pueden, además de describir el contenido, dar información sobre la estructura lógica del documento y la correspondencia entre sus componentes; así las relaciones entre los elementos aportan información sobre el significado de un elemento en concreto. En el ejemplo, aunque Aviso de Movimiento de Personal es el elemento raíz, pueden existir marcas relacionadas que aporten mayor significado al documento.

```
<PERSONAL> <NOMBRE></NOMBRE> </PERSONAL>
<CT> <NOMBRE></NOMBRE> </CT>
```

El contenido "Nombre" asociado a "Personal", nos lleva a relacionar a la persona afectada por el trámite; el contenido de "Nombre" asociado al "Centro de Trabajo" (CT) hace pensar en el Centro de Trabajo en que se desempeña o desempeñará esta persona.

2.1.1.7 Jerarquías de elementos XML

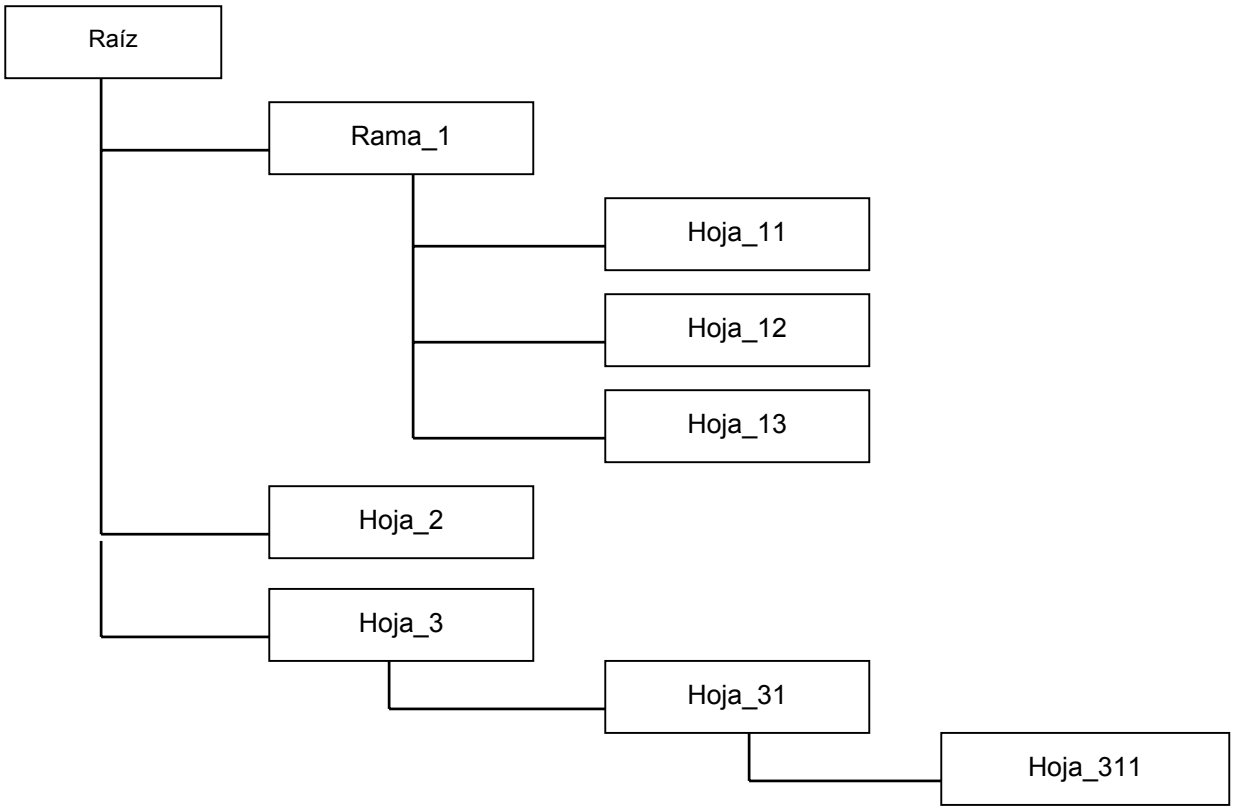
Enmarcados dentro del concepto de árboles, los elementos pueden anidarse unos con otros sin traslaparse y entremezclarse con información real del documento (datos carácter), para representar la estructura lógica y la semántica del contenido del documento. Esta composición de elementos dentro de la estructura de árbol mencionada, parte de un único elemento *raíz*, tal y como se muestra en la siguiente estructura general:

```
<raíz>
     <rama_1>
           <hoja_1_1> </hoja_1_1>
           <hoja_1_2> </hoja_1_2>
           <hoja_1_3> </hoja_1_3>
     </rama_1>

     <hoja_2> </hoja_2>

     <rama_3>
           <rama_3_1>
                 <hoja_3_1_1> </hoja_3_1_1>
           </rama_3_1>
     </rama_3>
</raíz>
```

Dado que existe una estructura de árbol, la podemos representar gráficamente de la siguiente manera:



Los elementos terminales en jerarquía y que no derivan en más elementos, reciben el nombre de hojas y los intermedios son las ramas.

Un documento XML representa a sus elementos bajo una estructura lógica, este tipo de gráfica puede ser útil para realizar un documento XML o para comprenderlo.

2.1.2 Atributos

Dentro del lenguaje XML, así como en HTML, existe otro componente que son los atributos, los cuales puestos en conjunto con los elementos, actúan como modificadores que aportan información adicional a los mismos.

Revisando el ejemplo del Aviso de Movimiento de Personal

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

<dependencia></dependencia>
<tipo_mov></tipo_mov>


<plaza num_plaza=”2311020102300015” departamento=”021” tabulador=”1024” />
```

```
<observaciones>
El fundamento legal es el 51, le precede el trámite 1967.
</observaciones>
```

El elemento <plaza> tiene los atributos num_plaza, departamento y tabulador que dan información adicional sobre la plaza asignada a un trabajador.

2.1.2.1 Usos de los atributos

La distinción entre un elemento y un atributo a veces es complicada dado que puede resultar lo mismo. Una característica importante de los atributos es que no pueden contener subelementos, ni subatributos, tampoco pueden organizarse en ninguna jerarquía, por lo tanto tienen una capacidad de representación más reducida en comparación con los elementos.

Como se ha visto en el ejemplo anterior, los atributos de la plaza son cortos e indivisibles en otras piezas lógicas, contrario a observaciones, las cuales son un pedazo de texto que el usuario escribió y la plaza no depende de ella.

Num_plaza puede considerarse como atributo o como elemento, porque puede dividirse en Unidad Presupuestal, Departamento, Tabulador y consecutivo de la Plaza, se tomó la decisión de manejarlo como atributo.

2.2 Visualización de documentos básicos

Para poder crear un documento XML, necesitamos de tres etapas:

- Especificación de los requisitos.
- Diseño de las etiquetas.
- Marcado de los documentos.

2.2.1 La especificación de requisitos

Para obtener buenos resultados, es necesario especificar lo que se pretende conseguir, las expectativas de crecimiento en función de los recursos disponibles.

La Especificación de Requisitos, depende de factores de costo, cantidad de personal, tiempo máximo de duración, etc., debemos tomar en cuenta la cantidad de automatización en el proceso de marcado. Cualquiera que sea la metodología empleada en el análisis de requisitos, ésta tiene que enfocarse a describir los procesos que pueden automatizarse y sobre estos procesos debe detallar la forma en que puede automatizarse el proceso de marcado, es decir, si el marcado debe hacerse de forma manual o automática lo cual afecta directamente a los costos mencionados.

Ejemplo:

A continuación se detalla una especificación de requisitos que enmarca una situación a la que se enfrentan los analistas de la Secretaría de Educación y Cultura en la elaboración de Avisos de Movimientos de Personal (AMP).

La Secretaría de Educación y Cultura de Veracruz, recibe a través de su Ventanilla principal, trámites procedentes de los Niveles Educativos, los Interesados o los Sindicatos. Cada trámite viene acompañado de un oficio (en caso de los Sindicatos o los Niveles) en donde se detallan los movimientos a realizarse y el orden de los mismos; también viene con la documentación soporte.

Los trámites recibidos tienen que ser separados por Áreas (primarias, secundarias, enseñanza media superior, superior) para que posteriormente el Área de Análisis los valide y genere el Aviso de Movimiento de Personal apropiado para el movimiento.

Se requiere que se tenga control sobre la entrega-recepción entre las Áreas de Ventanilla y Análisis ya que en ocasiones existe pérdida de documentos y en ocasiones el personal no canaliza los trámites adecuadamente.

Algunas de las búsquedas que se necesitan son:

- Por personal de Ventanilla
- Por tiempo empleado por trámite
- Por historial de los trámites que han ingresado a la Ventanilla
- Por vigencia de plazas en caso de bajas Ej. por fallecimiento.
- Por seguimiento del trámite, para saber en qué Área se encuentra y qué personal lo está trabajando.

El sistema realiza búsquedas automáticas para las plazas y muestra al personal de Ventanilla las validaciones resultantes con el objeto de que el personal pueda Rechazar el trámite desde esta primera entrada, o en su defecto aceptarlo con salvedades conforme a su criterio.

Como reporte y para personal gerencial, se requiere que el sistema proporcione tiempos utilizados por el personal de Ventanilla y Análisis y el seguimiento de cualquier trámite (el personal de Ventanilla y Análisis sólo podrá consultar este último).

2.2.2 Diseño de Etiquetas

En esta fase se deben definir cuáles son las marcas a incluir, sus nombres, la información que contienen, la relación entre marcas, etc.

Primer Paso: determinar las marcas necesarias

Basándonos en la especificación de requisitos, podemos determinar cuáles son los elementos de información que necesitamos incluir en nuestro documento XML.

Basándonos en nuestro ejemplo necesitamos la siguiente información:

- La pieza principal de información debe llamarse "RECEPCIÓN", debe contener un identificador o "FOLIO".
- El origen del movimiento debe almacenarse, para efectuar un posible rechazo posterior. Esta marca puede nombrarse como "ORIGEN" y tener una "DESCRIPCION".
- Para marcar el movimiento que se está realizando podemos utilizar a "MOTIVO".
- Los documentos recibidos y los faltantes deben nombrarse bajo "DOCUM_RECIBIDA" y "DOCUM_FALTANTE". Su jerarquía puede denominarse "PONDERACIÓN".
- La plaza que está afectando el trámite puede ser "PLAZA".
- El historial de trámites debe guardar el día y la hora en la que se realizó el movimiento, por lo que podemos citarlos como "FECHA_RECEPCIÓN" y "TMP_RECEPCION" respectivamente.
- Para marcar el tiempo de atención del trámite (antes de que pase a Análisis) podemos incluir "FECHA_ENTREGA" y "TMP_ENTREGA".
- El personal que atendió el trámite está asociado con un número de usuario en el sistema y se puede nombrar "NUM_USUARIO".

Segundo paso: añadir organización y estructura

Ya sabemos qué marcas debemos añadir ahora resta agruparlas y relacionarlas de la manera más coherente; este paso permitirá simplificar la programación.

Podemos identificar a la recepción, que indica que el trámite está en su primera fase y proporciona un identificador que acompañará al trámite; otra parte es la relativa a los datos del trámite recibido por el Área; una última es la correspondiente a la entrega a Análisis.

- De acuerdo con esto, podemos decir que ORIGEN, DESCRIPCION, PLAZA, MOTIVO, DOCUM_RECIBIDA, DOCUM_FALTANTE, PONDERACIÓN y PLAZA pueden englobarse en TRAMITE.
- FECHA_RECEPCION y TMP_RECEPCION pueden estar dentro de RECEPCION.
- Para definir el paso a Análisis podemos incluir a FECHA_ENTREGA y TMP_ENTREGA en ENTREGA_ANALISIS.

Tercer paso: Elementos o atributos

Ahora tenemos que determinar si estos "tipos de datos" pueden ser elementos o atributos, descartando como atributos aquellos que puedan contener subelementos o textos largos.

- El elemento raíz se llama "RECEPCIÓN".
- FOLIO es un elemento ya que identifica a la recepción mediante una llave única facilitando búsquedas.
- ORIGEN puede ser un elemento o un atributo pero es mejor que sea un elemento ya que así agrupa a su descripción.
- TIPO es un atributo de origen.
- NOMBRE por lo tanto es un atributo.
- MOTIVO y NUM_USUARIO son atributos del trámite.
- DOCUM_RECIBIDA es un elemento que contiene los documentos recibidos.
- DOCUM_FALTANTE es un elemento que contiene los documentos faltantes.
- PONDERACION es un atributo de la documentación anterior.
- PLAZA es un atributo de TRAMITE ya que para este ejemplo lo tomamos como indivisible y sólo pertenece a él.
- FECHA_RECEPCION, TMP_RECEPCION, FECHA_ENTREGA y TMP_ENTREGA son atributos de la recepción y de la entrega respectivamente.
- TRAMITE engloba a muchos elementos, por lo que es claramente un elemento. Lo mismo pasa con trámites.

2.2.3 El proceso de marcado

Este proceso cierra las fases de creación de un documento XML y requiere de las fases de diseño anteriores.

Para comprobar que el marcado es el correcto, es conveniente concluir comprobando que el documento XML esté bien formado; de existir errores se nos dificultaría el acceso al documento.

El documento XML quedaría de la siguiente forma:

```
<?xml version=”1.0” encoding=”ISO-8859-1” ?>
<RECEPCION FOLIO=”500” FECHA_RECEPCION="12/03/2003" TMP_RECEPCION="12:45">
      <ORIGEN TIPO=”Interesado” NOMBRE=”JOSÉ LUIS SAAVEDRA”>
       <TRAMITES>
            <TRAMITE MOTIVO=”REINGRESO” NUM_USUARIO=”37”
PLAZA="2311030112500015">
              <DOCUM_RECIBIDA>
                  <DOC_1 PONDERACION="ALTA">Propuesta Sindical<DOC_1>
                  <DOC_2 PONDERACION="ALTA">Perfil de Puesto<DOC_2>
                  <DOC_3 PONDERACION="BAJA">CURP<DOC_3>
              </DOCUM_RECIBIDA>
```

```
            <DOCUM_FALTANTE>
                <DOC_1 PONDERACION="ALTA">Identificación Oficial<DOC_1>
                <DOC_2 PONDERACION="ALTA">Orden de Presentación<DOC_2>
            </DOCUM_FALTANTE>
          </TRAMITE>
      </TRAMITES>
      <ENTREGA_ANALISIS FECHA_ENTREGA="" TMP_ENTREGA="" />
</RECEPCION>
```

2.3 Creación de Documentos XML Estructurados

2.3.1 Estructura de los DTD

La declaración del tipo de documento nos permite restringir el contenido de los elementos y atributos, pudiendo realizar un filtrado de los documentos que el procesador XML puede admitir o para detectar errores en la fase de elaboración de un documento cuando el operador está introduciendo los datos Ej. omisión de atributos

El detalle de la notación interpretada por XML y que indica cuál es el tipo de un documento y qué es lo que implica se realiza a través de "la declaración del tipo del documento".

Tal y como ya se ha mencionado, un documento XML puede constar de varias partes, un prólogo y un ejemplar. El prólogo puede componerse a su vez de una "declaración XML" y una "declaración de tipo".

La declaración de tipo consta de dos partes:

- La declaración de tipo del documento propiamente dicha.
- La definición del tipo del documento.

2.3.2 La Declaración del Tipo del Documento

Todas las declaraciones comienzan con el texto concreto que especifica el nombre del tipo; para tal efecto se utiliza la cadena "<!DOCTYPE" seguida del nombre del tipo. El texto mostrado a continuación muestra una declaración de tipo:

```
<!DOCTYPE ElemRaiz . . . >
```

El nombre asignado al tipo debe ser un nombre XML y debe estar separado de la cadena "`<!DOCTYPE`" por al menos un espacio en blanco.

Un tipo de documento se puede asociar con los documentos XML a través del elemento raíz, el nombre del elemento debe ser el mismo que el tipo, como se muestra a continuación:

```
<!DOCTYPE ElemRaiz . . . >
<ElemRaiz>

     . . .
</ElemRaiz>
```

La declaración debe situarse entre la declaración XML y la etiqueta de inicio del primer elemento (el elemento documento o raíz). En el siguiente ejemplo podemos ver esta estructura.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!-- Aviso de Movimiento de Personal -->
<!DOCTYPE AMP ... >
<AMP>
      . . .
</AMP>
```

A este tipo de elemento le falta la definición que le de algún significado.

2.3.3 La Definición del Tipo del Documento

La declaración de tipo para un documento viene acompañada por la **definición del tipo de datos** (abreviada como DTD) que relaciona al tipo con sus cualidades. La DTD define qué tipos de elementos, atributos, entidades y notaciones se podrían utilizar en el documento, así como las restricciones en cuanto a estructura y contenido, valores por defecto, etc. Para ello XML provee estructuras especiales que son **declaraciones de marcado** que pertenecen a alguno de los siguientes tipos:

- Declaraciones de tipos de elementos
- Declaraciones de listas de atributos para los tipos de elementos
- Declaraciones de entidades
- Declaraciones de notación.

Incorporando al ejemplo anterior algunas **"declaraciones de tipos de elementos"** que forman la **"definición del tipo"** se tiene lo siguiente:

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

<!DOCTYPE AMP [
      <!ELEMENT AMP (Dependencia,Movimiento,Encadenado) >
      <!ELEMENT Dependencia(#PCDATA) >
      <!ELEMENT Movimiento(#PCDATA) >
      <!ELEMENT Encadenado(#PCDATA) >
]>
<AMP>
  <Dependencia>Secretaría de Educación y Cultura</Dependencia>
  <Movimiento>Alta – Cambio con modificación de percepciones</Movimiento>
  El movimiento de encadena a la Baja anterior con Folio
  <encadenado>328</encadenado>
</AMP>
```

Esta definición de tipo especifica que el tipo de documento AMP está formado por elementos de tipo Dependencia, Movimiento y Encadenado. Estos elementos contienen exclusivamente Datos carácter (#PCDATA). El contenido del documento cumple con las restricciones de tipo, estructura y contenido especificadas.

Dependiendo en dónde se ubiquen las declaraciones de marcado, éstas pueden ser internas o externas, siendo las primeras las que se encuentran dentro de la entidad documento que se procesa. De aquí surge lo que se denomina subconjunto interno y el subconjunto externo. El subconjunto interno lo forman todas las declaraciones de marcado que se encuentran dentro del documento y el subconjunto externo todas las declaraciones que se encuentran fuera. Un documento XML puede componerse de una combinación de las dos.

Subconjunto interno

Estas declaraciones están incluidas dentro del documento, ubicándose dentro de los corchetes posteriores a la declaración del tipo de documento.

Las declaraciones del subconjunto interno son específicas de un documento y no pueden compartirse. Un ejemplo de este tipo de declaración de tipo de documento es la mencionada anteriormente.

Subconjunto externo

Se presenta cuando las declaraciones de marcado se encuentran fuera del documento XML. Estas declaraciones de marcado pueden ser llamadas utilizando:

- Una declaración explícita de subconjunto externo.
- Entidades de parámetro externas.

El subconjunto externo lo componen declaraciones comunes que se comparten entre varios documentos XML que pertenecen al mismo tipo y sólo es necesario modificarlas una sola vez.

En vez de corchetes, en este tipo de declaración se utiliza un sistema de referencia que permita localizar las declaraciones de marcado externas. La declaración del tipo de documento puede utilizar alguna de las siguientes formas:

```
<!DOCTYPE NombreXML SYSTEM "URL" >
<!DOCTYPE NombreXML PUBLIC "id_publico" "URI" >
```

En la primera, después de la palabra SYSTEM se especifica un URL donde se pueden encontrar las declaraciones. La segunda indica al procesador XML generar algún URL alternativo.

A continuación se presenta el ejemplo del AMP, utilizando declaraciones externas al documento. En el ejemplo se ha utilizado una referencia absoluta, pero puede utilizarse una relativa cambiando el URL para que incluya únicamente el archivo amp.dtd, situando los dos archivos en un mismo directorio.

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

<!DOCTYPE AMP SYSTEM "http://www.secver.gob.mx/SistemaTramites/amp.dtd">

<AMP>
  <Dependencia>Secretaría de Educación y Cultura</Dependencia>
  <Movimiento>Alta - Cambio con modificación de percepciones</Movimiento>
  <Encadenado>328</Encadenado>
</AMP>
```

El archivo amp.dtd contiene:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!ELEMENT AMP (Dependencia,Movimiento,Encadenado) >
<!ELEMENT Dependencia (#PCDATA) >
<!ELEMENT Movimiento (#PCDATA) >
<!ELEMENT Encadenado (#PCDATA) >
```

2.3.4 Declaraciones de Tipos de Elementos

A través de las "declaraciones de tipos de elementos", se pueden definir los tipos de elementos a emplearse en el documento, especificando los elementos permitidos y su contenido. Respecto a los elementos, para que un documento XML sea válido se debe cumplir que:

- Todos los elementos utilizados estén incluidos en "alguna declaración de tipos".
- El contenido de cada elemento esté dentro de lo declarado en su tipo.

Composición de las Declaraciones de Tipos de Elementos

Todas las declaraciones de tipos de elementos deben incluirse dentro de una definición de tipo de un documento (DTD), pudiendo ser ésta interna o externa.

Según las reglas de la recomendación XML, todas las declaraciones de Tipo de Elementos se componen de una cadena "<!ELEMENT" seguida de uno o más espacios en blanco, un nombre XML, espacios en blanco, la especificación del contenido del elemento, espacios en blanco opcionales y el carácter de cierre ">".

Las Declaraciones de Tipos de Elementos son individuales y el tipo de elemento sólo se declara una vez. El contenido de un elemento puede adoptar alguno de los tipos siguientes:

- EMPTY
- ANY
- Mixed (Mezcla)
- Elements

El tipo EMPTY (vacío)

Este tipo de declaración se utiliza para especificar que un elemento no puede tener contenido, el formato general para este tipo de declaración es:

```
<!ELEMENT nombreElemento EMPTY>
```

En este caso nombreElemento es el nombre del tipo de elemento declarado, EMPTY es la palabra reservada que define al tipo como vacío.

En este ejemplo se incluye una declaración de elemento vacío.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!--Aviso de Movimiento de Personal-->
<!DOCTYPE AMP [
                    <!ELEMENT AMP EMPTY>
              ]>
<AMP/>
```

El tipo ANY

En la siguiente declaración se crea un nuevo tipo de elemento "nombreElemento" en donde sus elementos pueden contener cualquier mezcla de datos carácter y elementos sin restricciones.

```
<!ELEMENT nombreElemento ANY>
```

Siguiendo con el ejemplo anterior, se puede declarar el elemento raíz como de tipo ANY. Para que el documento fuera de tipo válido se tendrían que declarar también los atributos que contienen los tipos de elementos "movimiento" y "plaza", los cuales se definirán más adelante.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!--Aviso de Movimiento de Personal-->
<!DOCTYPE AMP [
                    <!ELEMENT AMP ANY>
                    <!ELEMENT MOVIMIENTO EMPTY>
                    <!ELEMENT PLAZA EMPTY>
              ]>
<AMP>
      <MOVIMIENTO clave="010201" descripcion="Alta de Personal - Reingreso" />
      <PLAZA UP="2311" Depto="030" Tabulador="1040" plaza="00079" />
</AMP>
```

El tipo MIXED (Mezcla)

Son una mezcla de caracteres de datos y elementos

Para declarar elementos que contendrán sólo datos caracter, se usa la siguiente sintaxis:

```
<!ELEMENT nombreElemento (#PCDATA)>    ó    <!ELEMENT nombreElemento (#PCDATA)*>
```

Los elementos declarados como de tipo PCDATA no pueden contener caracteres tales como "<", "&" y "]]>"; para incluirlos es necesario utilizar secuencias de escape. "nombreElemento" es el nombre del tipo de elemento especificado.

Se puede declarar un tipo de elemento "observaciones" que incluya una descripción de los detalles del movimiento.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!--Aviso de Movimiento de Personal-->
<!DOCTYPE AMP [
                          <!ELEMENT AMP ANY>
                          <!ELEMENT MOVIMIENTO EMPTY>
                          <!ELEMENT PLAZA EMPTY>
                          <!ELEMENT OBSERVACIONES (#PCDATA)>
             ]>
<AMP>
      <MOVIMIENTO clave="010201" descripcion="Alta de Personal - Reingreso" />
      <PLAZA UP="2311" Depto="030" Tabulador="1040" plaza="00079" />
      <OBSERVACIONES>El personal pertenecía al S.E.T.S.E. y ahora es S.N.T.E.
      </OBSERVACIONES>
</AMP>
```

Se puede redefinir el elemento "Observaciones" del ejemplo anterior para que permita realizar búsquedas sobre los elementos que lo delimitan.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!--Aviso de Movimiento de Personal-->
<!DOCTYPE AMP [
                          <!ELEMENT AMP ANY>
                          <!ELEMENT movimiento EMPTY>
                          <!ELEMENT plaza EMPTY>
                          <!ELEMENT encadenado (#PCDATA)>
                          <!ELEMENT observaciones (#PCDATA | encadenado)*>
]>
<AMP>
      <movimiento clave="010201" descripcion="Alta de Personal - Reingreso" />
      <plaza UP="2311" Depto="030" Tabulador="1040" plaza="00079" />
      <observaciones>El personal pertenecía al S.E.T.S.E. y ahora es S.N.T.E.
      El movimiento anterior es el <encadenado>67/02</encadenado>
      </observaciones>
</AMP>
```

La declaración de tipo observaciones no fuerza a que contenga elementos o datos carácter, observaciones podría estar vacío.

El tipo Elements

Estos tipos de elementos solamente pueden incluir otros elementos y su definición no permite el manejo de datos carácter. A través de este tipo de elementos se pueden agrupar otros elementos y así formalizar estructuras. Existe un patrón que deben seguir los elementos del tipo, el cual está especificado en la declaración; este patrón puede tomar varias formas como se detalla a continuación:

Una secuencia de elementos.- Después del nombre del tipo de elemento declarado, se añade una lista de los elementos que puede contener, separados por comas.

```
<!ELEMENT nombreElemento (elemento1, ... , elemento N)>
```

Alternativa de elementos.- Para especificarla, en lugar de comas se utiliza la barra vertical "|".

```
<!ELEMENT nombreElemento (elemento1 | ... | elemento N)>
```

Lo anterior indica que sólo un elemento de la lista puede formar parte del contenido.

Combinación de modelos.- Los elementos se pueden agrupar mediante paréntesis para así utilizar de manera combinada secuencias y alternativas de modelos.

Especificación de frecuencia.- Se puede especificar una frecuencia de repetición añadiendo a un elemento o a un paréntesis de cierre los siguientes caracteres:

?: es opcional; el elemento o grupo se puede ocurrir cero o una vez

+: es obligatorio; indica una o más ocurrencias, debe presentarse al menos una.

*: representa cero o más ocurrencias.

No es posible especificar una frecuencia determinada y en caso de no aparecer ninguno de estos caracteres sólo se presentan una vez.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!--Aviso de Movimiento de Personal-->
<!DOCTYPE AMP [
      <!ELEMENT AMP (movimiento)*>
      <!ELEMENT movimiento (datos_generales, propios_movimiento, percepciones,
info_auxiliar,observaciones?,firmantes)>
      <!ELEMENT datos_generales (folio, fecha, relacion?, sector, dependencia,
tramite,motivo)>
      <!ELEMENT propios_movimiento (rfc, nombre, plaza, depto_actual, tabulador,
nombramiento,numero_personal,adscripcion, localidad,vigencia_ini,vigencia_fin?)>
                        <!ELEMENT folio (#PCDATA)>
                        <!ELEMENT fecha (#PCDATA)>
                        <!ELEMENT relacion (#PCDATA)>
                        <!ELEMENT sector (#PCDATA)>
                        <!ELEMENT dependencia (#PCDATA)>
                        <!ELEMENT tramite (#PCDATA)>
                        <!ELEMENT motivo (#PCDATA)>
                        <!ELEMENT rfc (#PCDATA)>
                        <!ELEMENT nombre (#PCDATA)>
                        <!ELEMENT plaza (#PCDATA)>
                        <!ELEMENT depto_actual (#PCDATA)>
                        <!ELEMENT tabulador (#PCDATA)>
                        <!ELEMENT nombramiento (#PCDATA)>
                        <!ELEMENT numero_personal (#PCDATA)>
                        <!ELEMENT adscripcion (#PCDATA)>
                        <!ELEMENT localidad (#PCDATA)>
                        <!ELEMENT vigencia_ini (#PCDATA)>
                        <!ELEMENT vigencia_fin (#PCDATA)>

                        <!ELEMENT percepciones (percepcion)+>
                        <!ELEMENT percepcion (tipo_perc,clave,importe)>
                        <!ELEMENT tipo_perc (#PCDATA)>
                        <!ELEMENT clave (#PCDATA)>
                        <!ELEMENT importe (#PCDATA)>

            <!ELEMENT info_auxiliar (zona_escolar, ofna_pago, programa,
subprograma,referencia?)>
                        <!ELEMENT zona_escolar (#PCDATA)>
```

```
                        <!ELEMENT ofna_pago (#PCDATA)>
                        <!ELEMENT programa (#PCDATA)>
                        <!ELEMENT subprograma (#PCDATA)>
                        <!ELEMENT referencia (#PCDATA | encadenado)*>

                        <!ELEMENT observaciones (#PCDATA | encadenado |
fundamento_legal)*>
                        <!ELEMENT encadenado (#PCDATA)>
                        <!ELEMENT fundamento_legal (#PCDATA)>

            <!ELEMENT firmantes (propone,visto_bueno,unidad_administrativa)>
                        <!ELEMENT propone (#PCDATA)>
                        <!ELEMENT visto_bueno (#PCDATA)>
                        <!ELEMENT unidad_administrativa (#PCDATA)>

]>
<AMP>
  <movimiento>
    <datos_generales>
        <folio>567</folio><fecha>22/01/2003</fecha>
        <sector>Secretaría de Educación y Cultura</sector>
        <dependencia>Dirección General de Educación Popular</dependencia>
        <tramite>Alta</tramite>
        <motivo>Reingreso</motivo>
    </datos_generales>
      <propios_movimiento>
        <rfc>ROGL710508</rfc>
        <nombre>Rodríguez Gómez Leticia</nombre>
        <plaza>2311028103500324</plaza>
        <depto_actual>028</depto_actual>
        <tabulador>1035</tabulador>
        <nombramiento>titular</nombramiento>
        <numero_personal>435634</numero_personal>
        <adscripcion>044863</adscripcion>
        <localidad>030560</localidad>
        <vigencia_ini>26/07/2003</vigencia_ini>
      </propios_movimiento>
      <percepciones>
        <percepcion>
          <tipo_perc>Sueldo</tipo_perc>
          <clave>4538</clave>
          <importe>2300</importe>
        </percepcion>
      </percepciones>
      <info_auxiliar>
        <zona_escolar>040</zona_escolar>
        <ofna_pago>6079</ofna_pago>
        <programa>02</programa>
        <subprograma>07</subprograma>
      </info_auxiliar>
      <observaciones>
        <encadenado>1037/03</encadenado>
        <fundamento_legal>Art. 34 ley Orgánica</fundamento_legal>
      </observaciones>
      <firmantes>
        <propone>Sergio Palacios G.</propone>
        <visto_bueno>Antonio López Gutiérrez</visto_bueno>
        <unidad_administrativa>Walterio Wild</unidad_administrativa>
      </firmantes>
  </movimiento>

</AMP>
```

Los documentos XML de tipo AMP se componen de cero o más elementos de tipo movimiento. Cada elemento de tipo movimiento puede estar formado a su vez de varios elementos de tipo datos_generales, propios_movimiento, percepciones, info_auxiliar, firmantes, todos obligatorios, seguidos de un elemento de tipo observaciones que es opcional.

2.3.5 Declaraciones de Listas de Atributos

Así como se trató acerca de las declaraciones de tipos de elementos, ahora se definirán los atributos asociados a estos tipos de elementos mediante Declaraciones de Listas de Atributos. Estas declaraciones tienen las siguientes características:

- Los atributos definidos están asociados solamente a un tipo de elemento.
- Establecen restricciones en el contenido de estos atributos. Los atributos deben sujetarse a los tipos predefinidos especificados.
- Nos dan a conocer la naturaleza del atributo, es decir, si es voluntario u obligatorio.

Los atributos deben especificarse completamente dentro de las DTDs mediante las declaraciones de listas de atributos.

Inclusión de las Declaraciones de Listas de Atributos

Todas las declaraciones de listas de atributos comienzan con la cadena "<!ATTLIST", seguido del nombre del elemento al que pertenecen y separados por uno o más espacios. A continuación se coloca el nombre del atributo que se está declarando, espacios, el tipo de atributo y su declaración por defecto. Se pueden tener una o varias declaraciones de atributos por elemento, por lo que se puede tener cualquiera de las siguientes formas:

```
<!ATTLIST nombre_elemento nombre_atributo tipo declaración_por_defecto>
```

ó

```
<!ATTLIST nombre_elemento     nombre_atributo tipo declaración_por_defecto
                              nombre_atributo tipo declaración_por_defecto
>
```

Un ejemplo de uso de atributos aplicado al AMP (Aviso de Movimientos de Personal), es el siguiente:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!--Aviso de Movimiento de Personal-->
<!DOCTYPE movimiento [

                        <!ELEMENT movimiento (ligados_movimiento)*>

                        <!ELEMENT ligados_movimiento
(rfc,nombre,plaza,depto_actual,tabulador,nombramiento,numero_personal,adsc
ripcion, localidad,vigencia_ini,vigencia_fin?)>
```

```
                        <!ELEMENT rfc (#PCDATA)>
                        <!ELEMENT nombre (#PCDATA)>
                        <!ELEMENT plaza (#PCDATA)>
                        <!ELEMENT depto_actual (#PCDATA)>
                        <!ELEMENT tabulador (#PCDATA)>
                        <!ELEMENT nombramiento (#PCDATA)>
                        <!ELEMENT numero_personal (#PCDATA)>
                        <!ELEMENT adscripcion (#PCDATA)>
                        <!ELEMENT localidad (#PCDATA)>
                        <!ELEMENT vigencia_ini (#PCDATA)>
                        <!ELEMENT vigencia_fin (#PCDATA)>

                        <!ATTLIST plaza unidad_presupuestal CDATA "">
                        <!ATTLIST plaza departamento CDATA "">
                        <!ATTLIST plaza tabulador CDATA "">
                        <!ATTLIST plaza numero_plaza CDATA "">
]>

<movimiento>
            <plaza unidad_presupuestal="2311" departamento="030"
tabulador="1043" numero_plaza="00165"> </plaza>
</movimiento>
```

En el ejemplo, podemos observar los siguiente:

- El orden en el que están especificadas las declaraciones debe respetarse, o sea, primero va el nombre del elemento definido después el nombre del atributo, etc.
- Los valores de los atributos estarán siempre delimitados por comillas dobles o simples.
- Los nombres de los atributos deben escribirse de igual forma, es decir, con sus mayúsculas y minúsculas.
- Se puede definir cualquier cantidad de elementos y atributos; XML no tiene ninguna restricción al respecto.
- Pueden mezclarse las declaraciones simples y las múltiples como se quiera, dependiendo del diseñador de la DTD; da el mismo resultado si se incluyen en una misma declaración todos los atributos de un mismo elemento.
- Los valores de los atributos para un elemento pueden declararse en cualquier orden.

Las partes que conforman una lista de atributos son:

- nombre_de_elemento
- nombre_del_atributo
- tipo
- declaración_por_defecto

Nombre del elemento

El nombre del tipo de elemento al que se asocia el atributo debe cumplir con las restricciones impuestas a los nombres XML. En la siguiente declaración de lista de atributos,

plaza es el nombre del tipo de elemento al que se asocian los atributos unidad presupuestal y departamento.

```
...
                <!ATTLIST plaza unidad_presupuestal CDATA "">
                <!ATTLIST plaza departamento CDATA "">
...
```

La siguiente es una declaración de atributos múltiple en la que el nombre del tipo de elemento sólo se menciona una vez.

```
...
            <!ATTLIST   plaza unidad_presupuestal (2310|2311) #REQUIRED
                              departamento        CDATA "" #REQUIRED
            >
...
```

Las declaraciones de listas de atributos para un tipo de elemento pueden especificarse en cualquier lugar dentro de la DTD; depende del analizador al leer todo el documento si detecta el error o si localiza todas las validaciones, comprobando que no falte ninguna declaración.

Nombre del atributo

El nombre del atributo debe ser un nombre XML y cumplir con las restricciones asociadas.

En el ejemplo siguiente, unidad_presupuestal y departamento son atributos de plaza

```
            <!ATTLIST plaza unidad_presupuestal (2310|2311) #REQUIRED>
            <!ATTLIST plaza departamento CDATA "" #REQUIRED>
```

Tipos de atributos

El tipo especifica cómo puede ser el contenido del atributo, es decir, define los tipos de valores que puede tomar el atributo. Los tipos están ya predefinidos y los enlistamos a continuación:

- Tipos cadena
- Tipos Enumerados
- NOTATION
- Enumeration
- Tipos tokeniced
- ENTITY, ENTITIES
- ID
- IDREF, IDREFS
- NMTOKEN, NMTOKENS

Tipos cadena

El tipo CDATA es el único dentro de esta categoría y quiere decir "Carácter DATA", es decir, datos carácter. Se puede introducir cualquier cadena que esté formada únicamente por caracteres válidos; este tipo de atributo es más permisivo que el resto de los tipos en cuanto a la cantidad de caracteres que admiten.

Los caracteres que no están permitidos para usarse dentro del valor del atributo CDATA son:

"<" Se interpreta como el comienzo de una marca.
"&" Se interpreta como referencia a una entidad.
" Se utiliza para delimitar el valor de un atributo

Si es necesario incluirlos en el valor del elemento, se hará conforme a la siguiente tabla:

Carácter	Cadena de escape
<	&lt;
&	&amp;
"	&quot;
'	&apos;

Si se quiere declarar un atributo del tipo cadena, se tiene que añadir la palabra CDATA a continuación de su nombre en la declaración de una lista de atributos.

La siguiente es una declaración de este tipo:

```
<!ATTLIST plaza departamento CDATA "">
```

Tipos numerados

En esta categoría existen dos clases:

Tipo enumerado: este tipo de atributos sirven para conjuntar una lista de posibles valores que puede tomar y que puede aceptar un posible atributo. Todos estos valores son tokens.

Tipo Notación: este tipo de atributo orienta a la herramienta XML para que el contenido del elemento XML sea procesado mediante la herramienta que indica la notación.

Para declarar un atributo de tipo enumerado hay que añadir a continuación del nombre del atributo una lista de valores válidos para el atributo, separados por un carácter "|".

```
<!ATTLIST nom_elemento atributo (valor1 | valor2 |...) valor_por_defecto>
```

Si se quiere definir un atributo como de tipo NOTATION[1], se añade la sentencia "NOTATION" seguida de una lista de notaciones encerradas entre paréntesis. Los nombres de las notaciones posibles se deben separar por caracteres de barra vertical "|".

```
<!ATTLIST nom_elemento NOTATION (notación1 | notación2 |...)
valor_por_defecto>
```

Mediante el ejemplo del movimiento, se podría definir un elemento foto que pudiera contener una foto en un formato determinado del personal, codificada usando una llave criptográfica de 128 bits.

```
<!ATTLIST nombre numero_personal CDATA "">
<!ATTLIST foto formato (gif | jpg)  'gif'>
<!ATTLIST codificación NOTATION (ssl128) 'ssl128'>
      <!--SSL a 128 bits-->
...
<foto formato="gif" codificación='ssl128'>
@ASSD|·%&GFF/JF)=KDK?KD </foto>
```

El formato de las fotos sólo será admitido ya sea en gif o en jpg, al declarar el tipo numerado nos aseguramos de que del valor dado está dentro del rango.

Los tokens tienen algunas limitaciones respecto a los caracteres que pueden incluir, entre ellas están: espacios en blanco, ",", "!", ";", "/" o "\".

Atributos token (nombre)

Este tipo de atributos proveen nombres, ya sea identificadores únicos o nombres que se relacionan con identificadores, tienen parecido con los atributos CDATA pero permiten sólo caracteres "name token".

ID. Se usa para darle un nombre a los elementos. Pueden tomar un nombre de un campo de una base de datos Ej. "Descripcion001". Los atributos de este tipo deben cumplir con las siguientes restricciones:

- Los valores que pueden tomar son nombres XML, no podrían tomar por ejemplo, "10".
- El valor del atributo ID debe ser único dentro del documento XML.
- Cada elemento puede tener como máximo un atributo ID.

IDREF, IDREFS: Estos atributos referencian elementos identificados con un atributo ID, es decir, en el valor del atributo IDREF/S se encuentra el valor de otro atributo ID. Este tipo de atributos se delimitan por lo siguiente:

- Las referencias de un tipo IDREF deben estar contenidas en un atributo de tipo ID, si las referencias no se encuentran en ningún ID el documento no será de tipo válido.

[1] Para que una notación pueda utilizarse, ésta debe definirse previamente.

- Los atributos IDREFS tienen las mismas características que los IDREF, sólo que pueden tomar varios valores separados por espacios.

ENTITY, ENTITIES: Permite manejar objetos (imágenes, sonido, etc.) a través de atributos entidad. Estas entidades contienen en su declaración el nombre de archivo que contiene los datos seguido de una notación que señala la aplicación capaz de interpretarlos. Las entidades deben declararse en una DTD o Esquema. Deben cumplir con las restricciones de nombres XML.

NMTOKEN, NMTOKENS: tienen parecido con los del tipo CDATA, sólo que sus valores válidos son nombres token. Los de tipo NMTOKENS son los mismos que los de tipo NMTOKEN, sólo que aceptan varios valores al mismo tiempo y están separados entre sí mediante espacios.

Utilizando el ejemplo de movimientos de personal:

```
...
                        <!ATTLIST AMP ID ID #REQUIRED>
                        <!ATTLIST AMP encadenado IDREF>
...
```

Declaraciones por defecto

Este tipo de declaración es la última parte de la declaración de un atributo. Los valores posibles para esta declaración son:

- #REQUIRED: si se declara un atributo como requerido, en el ejemplar del documento se le debe dar un valor a todos los atributos de este tipo.
- #IMPLIED: estos atributos pueden especificarse opcionalmente en los elementos del tipo en donde se declara el atributo.
- "valor del atributo": son valores por defecto que pueden tomar los atributos, de forma que el atributo quede inicializado a ese valor.
- #FIXED "valor del atributo": este tipo de atributos no cambia su valor y por tanto no es posible modificarlo.

Los atributos ID admiten sólo los valores por defecto #IMPLIED y #REQUIRED y no se permiten otros.

A continuación se muestra un ejemplo completo de declaraciones de atributos aplicadas a los ejemplos anteriores.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!--Aviso de Movimiento de Personal-->
<!DOCTYPE movimiento [

                    <!NOTATION jpg  SYSTEM  "PaintShopPro.exe">
                    <!ENTITY NP02 SYSTEM "archivo/fotos/100002.jpg" NDATA jpg>
```

```
                    <!ELEMENT movimiento (ligados_movimiento)*>

                    <!ELEMENT                                    ligados_movimiento
(clave,rfc,nombre,plaza,depto_actual,tabulador,nombramiento,numero_personal,adscrip
cion, localidad,vigencia_ini,vigencia_fin?)>

                    <!ELEMENT clave (#PCDATA)>
                    <!ELEMENT rfc (#PCDATA)>
                    <!ELEMENT nombre (#PCDATA)>
                    <!ELEMENT plaza (#PCDATA)>
                    <!ELEMENT depto_actual (#PCDATA)>
                    <!ELEMENT tabulador (#PCDATA)>
                    <!ELEMENT nombramiento (#PCDATA)>
                    <!ELEMENT numero_personal (#PCDATA)>
                    <!ELEMENT adscripcion (#PCDATA)>
                    <!ELEMENT localidad (#PCDATA)>
                    <!ELEMENT vigencia_ini (#PCDATA)>
                    <!ELEMENT vigencia_fin (#PCDATA)>

                    <!ATTLIST clave Folio ID #REQUIRED>
                    <!ATTLIST clave motivo CDATA "">
                    <!ATTLIST              plaza              unidad_presupuestal
(2310|2311|2312|2313|2314|2321)  #REQUIRED
                                           departamento CDATA ""
                                           tabulador CDATA ""
                                           numero_plaza CDATA ""
                          >
                          <!ATTLIST nombre foto ENTITY "">

]>

<movimiento>
  <ligados_movimiento>
             <clave Folio="A5003" motivo="010201">Alta de Personal</clave>
             <rfc>ORSA750206JUY</rfc>
             <nombre foto="NP02">Orlando Sánchez Antonio</nombre>
             <plaza unidad_presupuestal="2311" departamento="030" tabulador="1043"
numero_plaza="00165"> </plaza>
             <depto_actual>030</depto_actual>
             <tabulador>1043</tabulador>
             <nombramiento>Interino Ilimitado</nombramiento>
             <numero_personal>435123</numero_personal>
             <adscripcion>Xalapa-Centro</adscripcion>
             <localidad>001342</localidad>
             <vigencia_ini>01/07/2003</vigencia_ini>
  </ligados_movimiento>
</movimiento>
```

2.4 Esquemas XML como sucesores de los DTD

Las Definiciones de Tipo de Documentos (DTDs) son un mecanismo para definir reglas en un documento XML. A pesar de que este es un buen método, existen algunos problemas con los DTDs. El problema más obvio es el hecho de que los DTDs están escritos en su muy particular formato de texto, no en XML. Podría tener un mayor sentido crear un documento escrito en XML para definir las reglas de un documento XML.

En los ejemplos XML que hemos mencionado, no se le ha dado importancia alguna a los tipos de datos (todos los ejemplos han utilizado tipo de datos carácter). Sin embargo por lo general se tienen documentos que contienen diferentes tipos de datos y uno desearía poder

validar todos los tipos de datos. Desafortunadamente los DTDs no se diseñaron para validar los tipos de datos o para verificar rangos de valores. Los DTDs tampoco entienden los espacios de nombres (namespaces).

Para resolver todos estos problemas, se inventaron los *esquemas*. A diferencia de los DTDs, los cuales tienen su sintaxis particular, los esquemas XML se escriben en XML.

Además de proveer la información que ofrecen los DTDs, los esquemas permiten:

- Especificar los tipos de datos.
- Usar espacios de nombres.
- Usar herramientas XML tales como editores y navegadores para procesar esquemas.
- Permite especificar el número de veces que puede aparecer un elemento (En los DTDs sólo se pueden especificar 0,1 o infinito).
- Definir rangos de valores para los elementos y atributos.

Como se ha mencionado anteriormente, un documento que se ajusta a un DTD se denomina de **tipo válido**. Análogamente, un documento que se pueda ajustar a un esquema XML se llama **esquema válido**.

Los propósitos de los esquemas son:

- Utilizar la misma sintaxis de los documentos XML.
- Mejorar el tipo de datos XML para poder utilizar tipos como números, fechas, etc. y no solo cadenas.
- Introducir conceptos orientados a objetos tales como herencia.

Se puede encontrar información actualizada de los nuevos esquemas XML en el sitio Web del W3C en `www.w3.org/XML`.

Los proveedores están desarrollando un analizador capaz de validar documentos XML en función del esquema XML; Microsoft y Oracle son algunas compañías que están incluidas en este esfuerzo.

Si bien los esquemas proponen un mayor número de prestaciones, las herramientas requeridas para su validación y edición son limitadas y se encuentran en fase de desarrollo. Por esta razón se optó por apegarse a la definición tradicional por DTDs la cual hereda sus características del estándar SGML.

2.5 Entidades

Las **entidades** en XML son las unidades físicas que componen un documento, piezas de texto, archivos o recursos accesibles a través de URL. Los ejemplos mostrados hacen uso de al menos una entidad que comprende el archivo XML de partida y a la que le damos el nombre de entidad documento.

Las declaraciones de marcado incluidas fuera del documento XML, textos o imágenes incluidos en el documento que no están en el documento XML sobre el que se trabaja forman parte de unidades de almacenamiento y también son entidades.

Las entidades se han utilizado anteriormente para secuencias de escape de caracteres y gracias a ellas es posible incluir caracteres de marcado tales como >,<,&,",'.

Las entidades permiten dar un nombre a un texto o una cadena y así permitir reemplazar el nombre por el texto evitando tener que escribirlo cada vez. Esto nos permite también realizar las actualizaciones más eficientemente ya que cambiando la cadena en un sitio, se actualizan automáticamente todas sus ocurrencias.

Las entidades permiten manejar archivos que no son nativos de XML.

Tipos de Entidades

La recomendación XML define que existen tres tipos de entidades:

Las ***entidades generales*** nombran cadenas de texto para incluirlas dentro del documento. Estas cadenas de texto pueden englobar valores de atributos, el contenido de algún elemento, etc. Otro uso es la inclusión de datos que en principio no son XML, como imágenes o sonidos como parte del documento XML actual.

Las ***entidades parámetro*** también dan un nombre a una cadena de texto, pero se utilizan exclusivamente **dentro de la DTD** del documento XML, el texto que contienen es válido sólo en este contexto y contienen básicamente declaraciones de marcado.

Las ***entidades predefinidas*** se utilizan para escapar caracteres; estas entidades no es necesario declararlas dentro de la DTD.

2.5.1 CARACTERÍSTICAS COMUNES DE ENTIDADES

Las entidades generales y parámetro tienen las siguientes características similares:

- Los nombres de las entidades son nombres XML, deben cumplir las restricciones asociadas a nombres. Ambos nombres de las entidades generales y parámetro no se contraponen, ya que cada una tiene su manera de diferenciarlas.
- No se pueden hacer referencias recursivas.
- Los dos tipos de entidades deben declararse en la DTD.
- Si una entidad es declarada varias veces, el procesador XML toma como válida la primera e ignora las demás.
- Los valores incluidos no deben contener los caracteres: &, % y el carácter comilla ".

2.5.2 ENTIDADES GENERALES

Una entidad general puede ser interna o externa, analizable y no analizable.

- Las entidades analizables (parsed) pueden contener datos de marcado o datos carácter. Este tipo de información es capaz de ser tratada por el procesador XML. Estas entidades al declararse pueden utilizarse dentro del documento.
- Las entidades no analizables (unparsed) contienen información que no es de tipo XML, el procesador XML no la interpreta. La mayoría de las veces esta información está referida a archivos que están en un formato no nativo de XML como imágenes, sonidos, etc.
- Una entidad es interna cuando el contenido de la misma se ha definido completamente dentro del documento XML de partida.
- Una entidad es externa si su contenido está definido total o parcialmente fuera de la entidad documento XML de partida.

Los tipos de entidades generales de acuerdo con los adjetivos aplicados son:

- Entidades generales internas analizables: Su función es de tipo comodín. Al hacer referencia a una entidad de este tipo, se puede emplear una cadena de texto definida en su totalidad dentro del documento XML de inicio. Por lo que haciendo uso de referencias, el procesador las sustituye por texto.
- Entidades generales internas no analizables: este tipo no es posible, ya que en un documento XML no puede incluirse código binario que es no analizable.
- Entidades generales externas analizables: se parecen a las entidades generales internas, sólo que la diferencia es que están definidas (total o parcialmente) fuera de la entidad documento. Se utilizan también referencias.
- Entidades generales externas no analizables: este tipo abarca archivos de sonido, imágenes, documentos Word, imágenes de disco, etc. En un documento XML no es posible incluir ciertos elementos de información, por lo que necesariamente se tienen que incluir de manera externa. Se requiere de un marcado adicional aparte de la entidad para utilizarlas.

La ***declaracion de entidades generales*** comienzan con la cadena "<ENTITY" + espacios + nombre XML(entidad) + espacios + definición del valor de entidad + cero o más espacios + ">".

2.5.2.1 Entidades Generales Internas

Este tipo de entidades, tal como se menciona en la primera definición, sólo tienen sentido dentro del ejemplar del documento o en valores de atributos, el texto que contienen es de tipo XML puede ser de tipo marcado o de tipo datos carácter.

El procesador XML sustituirá cada una de estas entidades por su respectiva cadena, al momento de querer interpretar el contenido de un elemento o el valor de un atributo.

Declaración y Referencias

En una DTD, la forma que toman las declaraciones generales internas es:

```
<!ENTITY Nombre_asociado_a_entidad  "valor de la entidad">
```

Nombre_asociado_a_entidad se refiere al nombre de la entidad declarada, aquí queda asociado con el valor especificado entre comillas (dobles o simples).

Las referencias a estas entidades tienen la forma:

```
&nombre_asociado_a_entidad;
```

Nombre_asociado_a_entidad es el nombre de la entidad declarada, el proceso de lectura interpretará esta referencia como una sustitución por el texto entre comillas en la declaración.

Sitios de colocación de estas referencias

Estas referencias sólo pueden incluirse:

- Delimitadas por el contenido de un elemento; en una primera pasada la entidad es reemplazada y el texto se procesa de forma plana.
- Dentro del valor de un atributo, la referencia es sustituida y su valor puesto en su lugar. Si se llegaran a encontrar caracteres como la comilla simple o doble dentro de la entidad de reemplazo, el literal del atributo las incluye.
- Las referencias mencionadas no se permiten como el valor de un atributo.
- Si se incluyen dentro del valor de una entidad, la referencia no se sustituirá hasta que se necesite.
- Particularizando, no se pueden utilizar en otros lugares que no sean el valor de un atributo, una instrucción de proceso, un comentario, o el literal de un identificador público o de sistema; se pueden usar dentro de una DTD sólo dentro del valor de una entidad.

Ejemplos

Los ejemplos mostrados a continuación definen completamente la entidad utilizada dentro de la entidad documento y su contenido es el especificado por los estándares XML y toma valores de atributos o contenido de elementos.

En este primer ejemplo, la entidad utilizada contiene sólo texto.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE entidades_ej1[
      <!ELEMENT  entidades_ej1 (#PCDATA)>
      <!ENTITY Entidad_DatosCaracter "Está formada por texto contenido en
elementos">
]>
<entidades_ej1> &Entidad_DatosCaracter; </entidades_ej1>
```

En este otro ejemplo las entidades están compuestas únicamente de elementos, podemos observar el anidamiento de entidades generales internas:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE entidades_ej1[
      <!ELEMENT  entidades_ej1 (elemento_vacio)*>
      <!ELEMENT elemento_vacio EMPTY>
      <!ENTITY elementos  "<elemento_vacio></elemento_vacio>">
      <!ENTITY elementos_anidados "&elementos;" >

]>
<entidades_ej1> &elementos_anidados; </entidades_ej1>
```

Podemos realizar cualquier combinación de elementos y datos carácter.

Las entidades generales internas son las únicas admitidas dentro del valor de los atributos y solamente pueden contener texto que esté bien formado desde fuera, no debiendo contener trozos de marcado.

Terminada la sustitución, el texto final debe estar bien formado y además debe estar acorde con lo que está declarado, por ejemplo, un atributo de tipo ID no admitiría una cadena de texto cualquiera.

2.5.2.2 Entidades Generales Externas Analizables

Las entidades generales externas analizables se utilizan para hacer referencias a texto XML no incluido en el documento XML de partida. Contienen la lógica del texto junto con las entidades internas que abarca: marcado y datos carácter, por lo que tampoco permiten declaraciones de marcado y deben cumplir las mismas restricciones de buena formación y deben cumplir las restricciones de las DTD en cuanto al contenido de elementos y atributos. Una diferencia con especto a las internas es que éstas no se admiten dentro de atributos.

Forma de declararlas

Podemos declarar a las entidades generales mediante dos formas, siempre dentro de la DTD:

A través de un identificador de sistema:

```
<!ENTITY Nombre_de_Entidad SYSTEM "URL" ?>
```

O a través de un identificador público alternativo, junto con el identificador del sistema:

```
<!ENTITY Nombre_de_Entidad PUBLIC "id_publica" "URL" ?>
```

Nombre_de_Entidad es, como su nombre lo indica, el nombre de la entidad que se está declarando y "URL" es la localización del recurso. Como parte de la aportación de SGML se tienen los identificadores únicos y su objetivo es referenciar indirectamente recursos externos, es entonces cuando el procesador XML utiliza el identificador público para encontrar el recurso buscado.

El procesador XML tiene identificado dónde está el contenido de la entidad, si está de forma interna o externa, por lo que las referencias a las entidades generales externas son iguales a las referencias a entidades generales internas:

```
&Nombre_de_Entidad;
```

En este caso, Nombre_de_Entidad es el nombre de una entidad declarada, el procesador sustituirá la referencia por su respectivo valor descrito entre comillas en la declaración.

Sitios de colocación de estas referencias

Estas referencias se ubican correctamente:

- Dentro del contenido de un elemento, es decir, entre sus etiquetas de inicio y fin.
- Su uso no se permite dentro del valor de un atributo, ni como nombre de atributo.
- No se toman en cuenta las referencias a entidades generales analizables y se ignoran dentro del valor de otra entidad, no sufren cambios.
- No se permiten dentro de la DTD.

Ejemplos de aplicación

La siguiente es una entidad general externa que contiene texto y algunos elementos.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!--Aviso de Movimiento de Personal-->
<!DOCTYPE movimiento [

                    <!ELEMENT movimiento (ligados_movimiento)*>

                    <!ELEMENT      ligados_movimiento      (clave,rfc,nombre,plaza,
depto_actual,tabulador,nombramiento,numero_personal,adscripcion,localidad,vigencia_
ini,vigencia_fin?)>

                    <!ELEMENT clave (#PCDATA)>
                    <!ELEMENT rfc (#PCDATA)>
                    <!ELEMENT nombre (#PCDATA)>
                    <!ELEMENT plaza (#PCDATA)>
                    <!ELEMENT depto_actual (#PCDATA)>
                    <!ELEMENT tabulador (#PCDATA)>
                    <!ELEMENT nombramiento (#PCDATA)>
                    <!ELEMENT numero_personal (#PCDATA)>
                    <!ELEMENT adscripcion (#PCDATA)>
```

```
                    <!ELEMENT localidad (#PCDATA)>
                    <!ELEMENT vigencia_ini (#PCDATA)>
                    <!ELEMENT vigencia_fin (#PCDATA)>

                    <!ATTLIST clave Folio ID #REQUIRED>
                    <!ATTLIST clave motivo CDATA "">


                    <!ATTLIST  plaza  unidad_presupuestal  (2310|2311|2312|2313|2314
|2321)  #REQUIRED
                                                departamento CDATA ""
                                                tabulador CDATA ""
                                                numero_plaza CDATA ""
                          >
                    <!ENTITY Adscripción "Xalapa-Centro" >
                    <!ENTITY Datos_Movimiento SYSTEM "movim010201_435123.txt">
]>

<ligados_movimiento>
  &Datos_movimiento;
</ligados_movimiento>
```

El contenido del archivo `movim010201_435123.txt` contendría lo siguiente:

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

             <clave Folio="A5003" motivo="010201">Alta de Personal</clave>
             <rfc>ORSA750206JUY</rfc>
             <nombre foto="NP02">Orlando Sánchez Antonio</nombre>
             <plaza unidad_presupuestal="2311" departamento="030" tabulador="1043"
numero_plaza="00165"> </plaza>
             <depto_actual>030</depto_actual>
             <tabulador>1043</tabulador>
             <nombramiento>Interino Ilimitado</nombramiento>
             <numero_personal>435123</numero_personal>
             <adscripcion>&Adscripción;</adscripcion>
             <localidad>001342</localidad>
             <vigencia_ini>01/07/2003</vigencia_ini>
```

Puede observarse que existe un elemento de tipo adscripción que contiene una referencia a una entidad general interna definida en la entidad documento de inicio.

En el ejemplo se muestra claramente que es posible incluir referencias a entidades dentro del contenido de un elemento y dentro del contenido de otra entidad.

2.5.2.3 Entidades Generales Externas No Analizables

Las entidades generales externas no analizables se utilizan con el objeto de relacionar información binaria (archivos de imágenes, de sonido) y en general, cualquier archivo que no sea XML, con el archivo primario XML.

Forma de Declararlas

La forma de declarar estas entidades es la siguiente:

```
<!ENTITY Nombre_de_Entidad SYSTEM "url" NDATA Nombre_de_Notación >
```

Nombre_de_Entidad es el nombre de la Entidad que se está declarando, después de SYSTEM se especifica un "URL" al recurso designando para su incorporación a la entidad. La cadena NDATA le indica al procesador XML que después se especifica una notación (Nombre_de_Notación); a través de la notación el procesador XML puede relacionar la entidad con una aplicación capaz de manejarla.

Utilización de estas referencias

Estas entidades se pueden referenciar únicamente a través de atributos especiales llamados ENTITY o ENTITIES de la siguiente forma:

```
<!Elemento Atributo="Nombre_de_Entidad"> ... </Elemento>
```

Junto con estas entidades, se da lo siguiente:

- La declaración de la entidad: aquí se especifica en dónde se localiza la entidad y al declararla se le proporciona un nombre.
- Un atributo de tipo ENTITY o ENTITIES con el nombre de la entidad.
- Una notación declarada: a través de una notación se especifica la aplicación que es capaz de trabajar la entidad general externa a incluir.
- Un elemento que contiene el atributo

Ejemplo

En este ejemplo se quiere incluir la foto del personal como parte de sus datos generales

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!--Aviso de Movimiento de Personal-->
<!DOCTYPE datos_generales [

                    <!ELEMENT datos_generales (datos_personal)*>
                    <!ELEMENT datos_personal (nombre,rfc,numero_personal) >

                    <!NOTATION jpg  SYSTEM  "PaintShopPro.exe">
                    <!ENTITY NP02 SYSTEM "archivo/fotos/100002.jpg" NDATA jpg>
                    <!ELEMENT nombre (#PCDATA)>
                    <!ELEMENT rfc (#PCDATA)>
                    <!ELEMENT numero_personal (#PCDATA)>
                     <!ATTLIST nombre foto ENTITY "">
]>
<datos_generales>
      <datos_personal>
            <nombre foto="NP02">Orlando Sánchez Antonio</nombre>
            <rfc>ORSA750206JUY</rfc>
            <numero_personal>435123</numero_personal>
      </datos_personal>
</datos_generales>
```

2.5.3 ENTIDADES PARÁMETRO

Las entidades parámetro se utilizan dentro de la DTD en forma de comodines; a las entidades parámetro se les aplican modificadores.

Las entidades parámetro pueden ser internas o externas, todas las entidades parámetro son analizables; hablar de entidades parámetro no analizables no tendría sentido dado que su objetivo es ayudar a la definición de las DTD.

Forma de Declararlas

Las entidades parámetro se declaran de forma diferente, su estructura es la siguiente: Cadena "<!ENTITY" + espacios + un carácter de porcentaje "%" + espacios + un nombre XML + el nombre de entidad + espacios + tipo de entidad declarada (interna o externa) + el carácter ">".

Para las referencias a este tipo de entidades se utiliza también el carácter de "%", por lo que la declaración queda "%nombre_de_entidad".

Características comunes:

- Cada entidad parámetro debe declararse antes de poder ser utilizada.
- Si una entidad parámetro interna o externa se utiliza dentro de la definición interna de un documento XML, sólo puede declararse en su totalidad y no se pueden utilizar referencias a entidades parámetro dentro de una declaración de marcado, excepto si se define otra entidad parámetro.
- El contenido de las entidades parámetro debe estar bien formado.

Sitios de colocación de estas referencias

- Se ignoran todas las referencias a estas entidades que estén dentro del contenido de un elemento, dentro del valor de un atributo o como nombre de un atributo.
- Las referencias a estas entidades que estén dentro de otra entidad se incluyen dentro del literal. Si en el texto de reemplazo existen comillas, no truncan el valor de la entidad que se sustituye en la referencia.
- Se sustituyen las referencias a estas entidades si se encuentran dentro de una DTD o si no se encuentran dentro de una entidad, de un atributo, de una instrucción de proceso, de un comentario o fuera de los literales de identificadores públicos o de sistema. Este tipo de entidades no pueden contener declaraciones sin un sentido de unidad, como es el caso de un nombre XML (no se pueden unir dos entidades parámetro para formar un nombre).

2.5.3.1 Entidades Parámetro Internas (Analizables).

Las entidades parámetro internas se utilizan para nombrar cadenas dentro de una DTD. Estas entidades no pueden utilizarse en las declaraciones de marcado.

Declaración

Las entidades parámetro internas se declaran:

```
<!ENTITY % Nombre_de_Entidad "valor_de_entidad">
```

Esta declaración es muy similar a la de las entidades generales internas, excepto por el carácter %.

Referencias

La estructura de las referencias a las entidades parámetro es:

```
%Nombre_de_entidad;
```

Estas referencias se reconocen únicamente dentro de la DTD y deben declararse previamente.

c) Ejemplos

En este ejemplo se utilizan tres entidades parámetro internas.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE datos_generales [
            <!ELEMENT datos_generales (datos_personal)*>
            <!ELEMENT datos_personal (nombre,rfc,numero_personal) >
            <!ENTITY % nombre "<!ELEMENT nombre (#PCDATA)>">
            <!ENTITY % rfc "<!ELEMENT rfc (#PCDATA)>">
            <!ENTITY % numero_personal "<!ELEMENT numero_personal (#PCDATA)>">
            %nombre;
            %rfc;
            %numero_personal;
]>
<datos_generales>
      <datos_personal>
            <nombre>Orlando Sánchez Antonio</nombre>
            <rfc>ORSA750206JUY</rfc>
            <numero_personal>435123</numero_personal>
      </datos_personal>
</datos_generales>
```

En este otro ejemplo, la entidad parámetro interna contiene una declaración de un atributo con una entidad general que define su valor.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE datos_generales [
            <!ELEMENT datos_generales (datos_personal)*>
            <!ELEMENT datos_personal (estado,nombre,rfc,numero_personal)>
            <!ENTITY % estado "<!ELEMENT estado (#PCDATA)>">
            <!ENTITY % nombre "<!ELEMENT nombre (#PCDATA)>">
            <!ENTITY % rfc "<!ELEMENT rfc (#PCDATA)>">
```

```
            <!ENTITY % numero_personal "<!ELEMENT numero_personal (#PCDATA)>">
            <!ENTITY número_maestros "40,000">
            <!ENTITY % Atributo_estado "<!ATTLIST estado número_maestros CDATA
'&número_maestros;'>">
            %estado;
            %nombre;
            %rfc;
            %numero_personal;
            %Atributo_estado;
]>
<datos_generales>
      <datos_personal>
            <estado>Veracruz</estado>
            <nombre>Orlando Sánchez Antonio</nombre>
            <rfc>ORSA750206JUY</rfc>
            <numero_personal>435123</numero_personal>
      </datos_personal>
</datos_generales>
```

2.5.3.2 Entidades Parámetro Externas (Analizables)

Permiten una declaración más fácil de las DTDs y permiten que se compartan declaraciones de marcado entre varios documentos.

Las declaraciones de marcado indicadas a través de entidades parámetro externas pertenecen al subconjunto externo porque están definidas fuera de la entidad documento de origen. El procesador XML debe procesar también las entidades parámetro en el caso que se quiera validar este tipo de entidades.

Declaración

Las entidades parámetro externas pueden procesarse de dos maneras:

```
<!ENTITY % Nombre_de_entidad SYSTEM "URL" ?>
```

O con identificador público:

```
<!ENTITY % Nombre_de_entidad PUBLIC "id_pública" SYSTEM "URL" ?>
```

Referencias

Las referencias a las entidades parámetro internas son de la siguiente forma:

```
%Nombre_de_Entidad
```

Formas de Aplicación

A diferencia de las declaraciones anteriores, estas entidades si pueden contener partes de declaraciones de marcado, pero la declaración mínima debe ser una unidad lógica completa, un nombre, un tipo, un valor por defecto, etc., añadiendo el procesador XML un espacio en blanco para asegurar la compatibilidad.

2.6 NOTACIONES

Las notaciones relacionan un nombre XML con una aplicación externa capaz de procesar los datos proporcionados.

Las declaraciones pueden tomar cualquiera de las formas siguientes:

```
<!NOTATION Nombre_Notación SYSTEM "Url_Prog_aplicación">
```

ó

```
<!NOTATION Nombre_Notación PUBLIC "id_pública" "Url_Prog_aplicación">
```

"Nombre_Notación" corresponde al nombre de la notación y debe cumplir con las restricciones de nombres XML, "Url_Prog_aplicación" especifica un URL que indica la localización de la aplicación que puede procesar los datos. En el segundo caso, "id_pública" es un identificador público para el recurso que puede utilizarse para encontrar la aplicación. Estas declaraciones se colocan dentro de la DTD, en el subconjunto interno o externo, debiéndose declarar todas las notaciones.

Usos de las notaciones:

- Proporcionar instrucciones para procesar correctamente el contenido de un elemento.
- Para declarar entidades generales externas no analizables relacionadas con del documento XML a través de atributos ENTITY o ENTITIES.
- En las instrucciones de proceso, declarar la aplicación destino de la instrucción.

En un ejemplo anterior donde se hacía referencia a las fotos del personal, se pueden observar las notaciones.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!--Aviso de Movimiento de Personal-->
<!DOCTYPE movimiento [
                    <!NOTATION jpg  SYSTEM  "PaintShopPro.exe">
                    <!ENTITY NP02 SYSTEM "archivo/fotos/100002.jpg" NDATA jpg>
]>
<movimiento>
  <ligados_movimiento>
             <nombre foto="NP02">Orlando Sánchez Antonio</nombre>
  </ligados_movimiento>
</movimiento>
```

2.6.1 INSTRUCCIONES DE PROCESO

Un ejemplo de instrucción de proceso es la declaración XML que se ha utilizado en todos los ejemplos mostrados hasta el momento; su función es dar a conocer que el contenido del documento es XML, la versión, la codificación utilizada y si es autónomo o no.

La sintaxis general de una instrucción de proceso es la siguiente:

```
<?aplicacion_destino cadena ?>
```

Las instrucciones pueden hacer uso de notaciones.

A través de las notaciones, las instrucciones pueden declarar cuál es la aplicación a la que envían la información. En la línea anterior "aplicacion_destino" se refiere a un nombre de notación que pudiera aparecer en el código. La siguiente parte simbolizada en este caso por *cadena*, se refiere al contenido de la instrucción o el comando que se mandará a la aplicación declarada en la notación, con la restricción de que *cadena* no puede contener los caracteres que cierran la instrucción "?>".

Las instrucciones de proceso pueden colocarse dentro del prólogo de la DTD, entre el prólogo y el ejemplar del documento, dentro del ejemplar del documento, y después del ejemplar del documento, es decir, después de la etiqueta de cierre del elemento raíz.

El siguiente es un ejemplo de una instrucción de proceso:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<?xml-stylesheet type="text/css" ?>

<!DOCTYPE FormatoAMP [
      <!ELEMENT FormatoAMP (Persona, Movimiento)>
      <!ELEMENT Persona (#PCDATA)>
      <!ELEMENT Movimiento (#PCDATA)>
]>

<FormatoAMP xmlns:HTML="http://www.w3.org/Profiles/XHTML-transitional">
      <HTML:STYLE>
            Persona {
                    display:block;
                    color: #000F00;
                    font-style: italic;
                    text-align: center;
            }
              Movimiento {
                    display:block;
               }
      </HTML:STYLE>
      <Persona>JOSE RAMON PEREZ</Persona>
      <Movimiento>Licencia sin goce de sueldo</Movimiento>
</FormatoAMP>
```

La instrucción de proceso que se encuentra en la segunda línea, indica al navegador que el formato del documento se define mediante una hoja de estilos en cascada (text/css).

Secciones CDATA

Una sección CDATA es una manera de especificar a un procesador XML que un determinado texto no debe ser interpretado como marcado. La única secuencia de caracteres permitidos son los que cierran el CDATA "]]>", haciendo imposible anidar secciones CDATA.

Las secciones CDATA son tomadas en XML como información que debe transferirse a las aplicaciones y una diferencia fundamental es que los comentarios no se consideran datos carácter.

Las secciones CDATA comienzan por la cadena "<![CDATA[" y terminan con "]]>", como se puede ver a continuación.

```
<![CDATA[ Aquí se incorpora un texto <HTML> que puede ser <B>HTML</B>
</HTML> ]]>
```

Un texto más completo de una sección CDATA lo mostramos en las siguientes líneas:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<formato_HTML>
      Formato de CEDULA DE PERSONAL, incorpora los datos necesarios para
un movimiento de personal.
     <![CDATA[
      <HTML>
        <HEAD>
          <TITLE>
            Cedula de Personal
          </TITLE>
            <META HTTP-EQUIV='Content-Type' CONTENT='text/html;
charset=iso-8859-1'>
        </HEAD><BODY background='imagenes/formato1.jpg'>
            <center><H1>C&eacute;dula de Personal</H1></center>
            <br>
            <font face="Arial">Nombre:
            <br>Número de Personal:
      ]]>
...
</formato_HTML>
```

2.7 ESPACIOS DE NOMBRES

Como lo indica su nombre, XML es un lenguaje extensible. La implantación de esta característica requiere considerar el manejo de la extensibilidad en un ambiente distribuido en el que no haya conflictos.

Los espacios de nombres son la solución prouesta para ayudar a manejar la extensibilidad de XML; su principal utilidad es evitar la confusión de elementos con el mismo nombre al momento de combinar documentos distintos.

Ejemplo. Se requiere integrar en uno solo los siguientes documentos XML de niveles_educativos y personal_docente:

```
<niveles_educativos>
  <nombre id="1">Educación Básica</nombre>
  <nombre id="2">Bachillerato</nombre>
</niveles_educativos>


<personal_docente>
<nombre id="1">Juan Castañeda</nombre>
<nombre id="2">Ma. del Carmen López</nombre>
</personal_docente>
```

Se podría realizar mediante la inclusión de un espacio de nombres:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<personal_docente xmlns:personal_docente="http://sec_intranet/personal">
<personal_docente:nombre personal_docente:id="1">Juan Castañeda</nombre>
<personal_docente:nombre personal_docente:id="2">Ma. del Carmen
López</nombre>
</personal_docente>
```

Antes de incluir un espacio de nombres es necesario declararlo, hay que asociar un índice con un URL asignado al espacio, mediante un atributo especial `xmlns`.

En el ejemplo anterior se conjuntaron dos documentos, pero puede darse el caso de que se tenga que trabajar sobre un documento ya estructurado, por la naturaleza de trabajo colaborativo basado en el Web.

En este otro ejemplo, al archivo de personas hay que agregarles un grado de eficiencia (del 1 al 10), quedando como sigue:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<personal_docente>
<nombre id="1">Juan Castañeda</nombre>
<eficiencia>76</eficiencia>
<nombre id="2">Ma. del Carmen López</nombre>
<eficiencia>90</eficiencia>
</personal_docente>
```

Si otro jefe llegara a calificar la eficiencia lo podría hacer de otro modo y el mismo sería incompatible con el primero:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<personal_docente>
<nombre id="1">Juan Castañeda</nombre>
<eficiencia>76</eficiencia>
<eficiencia>Regular</eficiencia>
<nombre id="2">Ma. del Carmen López</nombre>
<eficiencia>90</eficiencia>
<eficiencia>Muy buena</eficiencia>
</personal_docente>
```

El problema anterior podría solucionarse mediante el cambio del elemento `<eficiencia>` a `<a-eficiencia>` y `<b-eficiencia>`. Esta solución nos lleva a pensar en que el documento XML podría extenderse de una manera incompatible para otros trabajos ya que nadie puede crear etiquetas por sí solo. Podría entonces establecerse un registro global de etiquetas válidas y su definición pero esto afectaría notablemente a la flexibilidad de XML.

Con el objeto de no limitar el lenguaje XML sacrificando su flexibilidad y extensibilidad surgen los **espacios de nombres**.

El ejemplo previo queda como sigue utilizando espacios de nombres:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
  <personal_docente xmlns:a="http://www.dinasoft.com.mx/doc1.0"
                    xmlns:b="http://www.dinasoft.com.mx/doc2.0"
                    xmlns="http://www.dinasoft.com.mx/doc3.0">
      <nombre id="1">Juan Castañeda</nombre>
      <a:eficiencia>76</a:eficiencia>
      <b:eficiencia>Regular</b:eficiencia>
      <nombre id="2">Ma. del Carmen López</nombre>
      <a:eficiencia>90</a:eficiencia>
      <b:eficiencia>Muy buena</b:eficiencia>
  </personal_docente>
```

En este ejemplo, se ha añadido la estructura de prefijo a la estructura de nombres de elementos, por lo que en el caso de `<a:eficiencia>` , al elemento "eficiencia" se le antepone el prefijo "a".

Bajo este esquema de prefijos, cualquiera puede crear prefijos que sean incompatibles, regresando de esta manera al problema mencionado anteriormente. Para evitar los conflictos entre prefijos, éstos deben ser declarados:

```
  <personal_docente xmlns:a="http://www.dinasoft.com.mx/doc1.0"
                    xmlns:b="http://www.dinasoft.com.mx/doc2.0"
                    xmlns="http://www.dinasoft.com.mx/doc3.0">
```

La declaración asocia un URI con un prefijo, y es el objetivo central de los espacios de nombres, ya que los URIs, a diferencia de los nombres, son únicos.

Alcance

El espacio de nombres es válido para el elemento sobre el que se declara y todos los elementos que contiene, tal y como se enlista el siguiente ejemplo:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
  <ru:personal_docente  xmlns:ru="http://www.dinasoft.com.mx/doc3.0">
      <ru:nombre id="1">Juan Castañeda</ru:nombre>
      <a:eficiencia xmlns:a="http://www.dinasoft.com.mx/doc1.0">
      76</a:eficiencia>
      <ru:nombre id="2">Ma. del Carmen López</ru:nombre>
```

```
      <b:eficiencia xmlns:b="http://www.dinasoft.com.mx/doc2.0">Muy
buena</b:eficiencia>
  </ru:personal_docente>
```

Aquí se declaran tres espacios de nombres. Se declara `ru` en el elemento raíz y por lo tanto es válido para todos los elementos. Se declara `a` como el primer elemento asociado a `eficiencia`. Por último se declara `b` como segundo elemento de `eficiencia`.

3.1 Herramientas de creación de documentos XML

3.1.1 Microsoft XML Notepad

XML Notepad es una aplicación sencilla que permite la construcción de prototipos rápidos de aplicaciones XML. Los autores pueden construir y editar rápidamente pequeños trozos de datos XML durante el desarrollo de aplicaciones basadas en XML.

XML Notepad ofrece una interfaz de usuario simple e intuitiva que representa gráficamente la estructura de árbol de la información XML. Se puede trabajar con bloques de construcción XML estándar (elementos, texto y atributos), los autores de los documentos son capaces de crear estructuras de datos reproducibles que pueden llenarse fácilmente.

La interfaz de usuario se muestra a continuación:

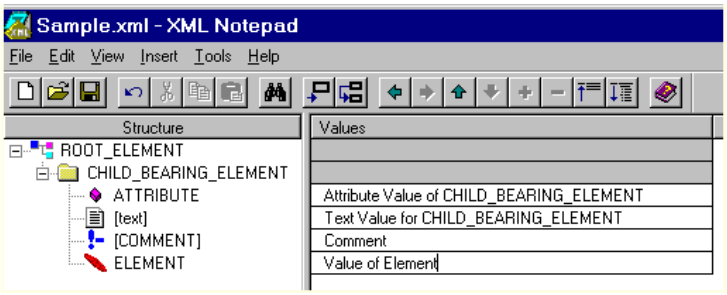


Fig. 3.1 *Interfaz de usuario de XML Notepad.*

La última versión de este software utilizada al momento de la realización del presente trabajo, no es capaz de crear los DTDs. Sin embargo soporta la creación de esquemas XML, los cuales se representan mediante la sintaxis XML.

Al ser éste un editor, no tiene la capacidad de validar documentos bien formados ni de realizar validaciones de DTD o de esquemas XML. Las validaciones tienen que realizarse externamente ya sea mediante Internet Explorer 5.0 o superior o cualquier otro producto de los mencionados en este capítulo.

3.2 Herramientas de validación de sintaxis XML y de validación de semántica DTD

3.2.1 Macromedia Dreamweaver MX

Macromedia Dreamweaver MX es una herramienta de creación, edición y validación de código aplicable a la Web; incluye la edición de:

- Páginas dinámicas como ASP (JavaScript y VBScript), ASP.NET (C#, VB), JSP, ColdFusion y PHP.
- Páginas estáticas de tipos HTML, CSS y XML (.xml, .dtd, .xsl y .xslt)

Dreamweaver MX incluye, además de las herramientas de edición e inclusión de eventos de la versión anterior *Dreamweaver Ultradev 4*, mejoras en las plantillas para auxiliar a los diseñadores visuales y nuevas capacidades de edición de código. Para el desarrollo de aplicaciones, MX ofrece un nuevo espacio de trabajo (workspace) proveniente de Macromedia ColdFusion Studio, el código de ejecución (runtime) es mejor, y soporta las últimas tecnologías en aplicaciones Web.

Para nuestros fines, Macromedia MX permite la validación de código bien formado y de documentos XML válidos (con su DTD) mediante una herramienta de validación conocida como *Dreamweaver Validator*, la cual permite verificar si el código tiene errores de etiquetas o de sintaxis.

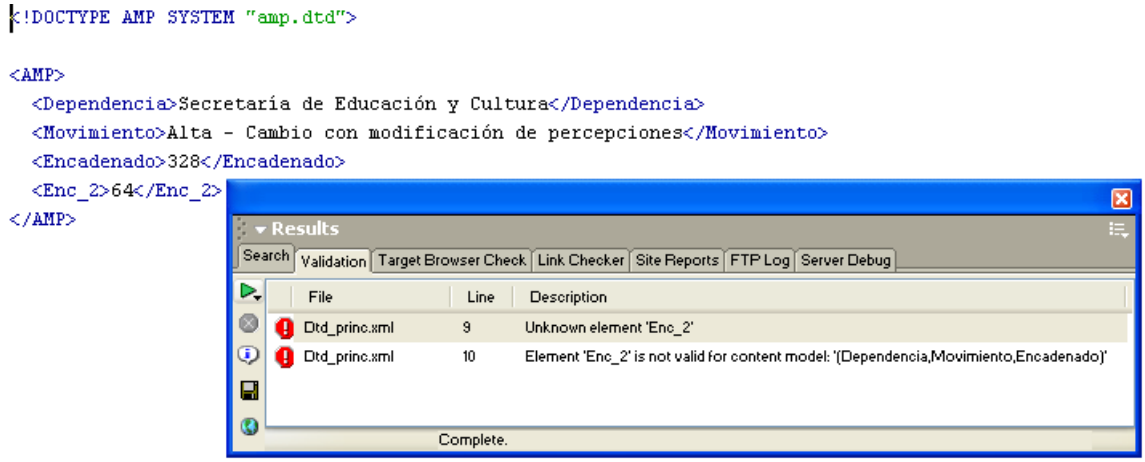


Fig. 3.2 *Validación de un documento XML en Dreamweaver MX.*

Por tanto, esta herramienta permite analizar documentos bien formados y también aquellos que deban cumplir con una DTD (válidos).

3.2.2 Parsers XML

Los documentos pueden ser ya sea bien formados o válidos. Los documentos que son bien formados respetan las reglas sintácticas. Los documentos válidos no sólo respetan las reglas sintácticas sino que también se adecuan a la estructura descrita en un DTD.

Existen parsers[1] que validan y otros que no lo hacen (sólo analizan sintácticamente). Ambos parsers aseguran las reglas sintácticas pero sólo los parsers que validan están programados para confrontar documentos con sus DTDs.

Los parsers que no hacen validación pueden leer documentos válidos pero nunca validarlos. Para un parser que no valida, cualquier documento, ya sea que utilice una DTD o no, es un documento bien (o mal) formado.

Un parser que valida acepta documentos bien formados. Por supuesto, cuando se trabaja con documentos bien (o mal) formados, se comporta como un parser que no valida.

3.2.2.1 Xerces

Xerces provee de herramientas para el análisis (parsing) y la generación de código XML. Los parsers de validación están disponibles tanto para Java como para C++. Implementa los estándares para XML del W3C tales como DOM (nivel 1 y 2) y SAX (versión 2).

Sus características principales son las siguientes:

- Los parsers son altamente configurables y modulares.
- Provee de un soporte inicial para esquemas XML.
- Provee de una envolvente de Perl para las versiones de C++ de Xerces, la cual da acceso a un completo parser para validar XML DOM desde Perl.
- Provee también de acceso completo a cadenas Unicode, dado que Unicode es una pieza principal del estándar XML.
- Una envolvente de COM (Component Object Model) proporciona compatibilidad con el parser de Microsoft MSXML.

La siguiente es una lista de componentes utilizados en la configuración estándar de Xerces2:

- Tabla de símbolos
- Reporteador de Errores
- Escáner del Documento
- Escáner de la DTD
- Gestor de entidades
- Validador de la DTD
- Enlazador con Namespaces
- Validador de Esquemas

En general, existen niveles de dependencia entre los componentes en la configuración estándar. Algunos componentes son requeridos por otros componentes. El siguiente diagrama ilustra estos niveles básicos de dependencia.

[1] El término *parser* proviene de los compiladores, los cuales se conforman de un módulo que lee e interpreta el lenguaje de programación.

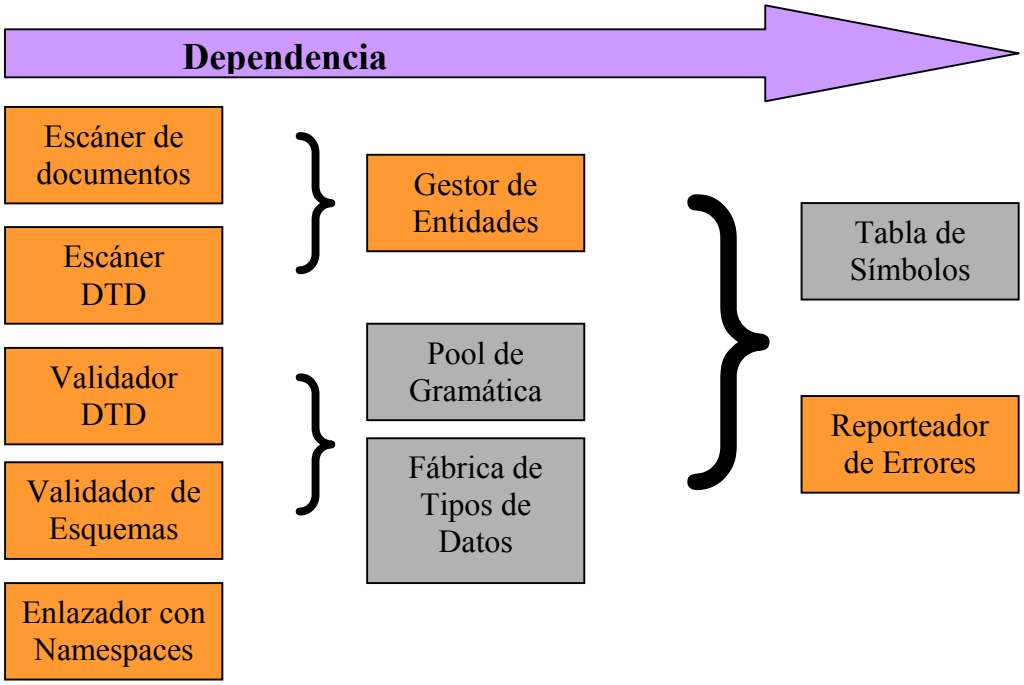


Fig. 3.3 *Niveles básicos de dependencia entre componentes de Xerces.*

Las dependencias básicas, tal y como se muestra en el diagrama, se enlistan a continuación:

- Todos los componentes configurables de Xerces2 en la configuración estándar dependen de la "Tabla de Símbolos" y del "Reporteador de Errores"
- Ambos, el "Escáner de documentos" y el "Escáner DTD" dependen del componente "Gestor de Entidades".
- Adicionalmente a las otras dependencias, el "Escáner de Documentos" también depende del "Escáner DTD" para analizar los subconjuntos internos y externos de la DTD.

Xerces es parte del proyecto XML de Apache, es altamente configurable y con un rico conjunto de características. Está diseñado para alcanzar los estándares de rendimiento y compatibilidad al momento de analizar documentos XML.

Podemos observar entonces que esta herramienta es capaz de realizar la comprobación de documentos bien formados sintácticamente y válidos (analizando las DTD).

3.2.2.2 Stylus Studio 5 XML Professional Edition

. La revisión 3 de este producto incluye una herramienta de validación de XML y un parser XML, y soporta componentes XML usados comúnmente por desarrolladores tales como: MSXML 4.0 SAX, MSXML 4.0 DOM, Microsoft .NET XML Parser (System.XML), Xerces-J 2.5.1, XSV 2.6 y otros.

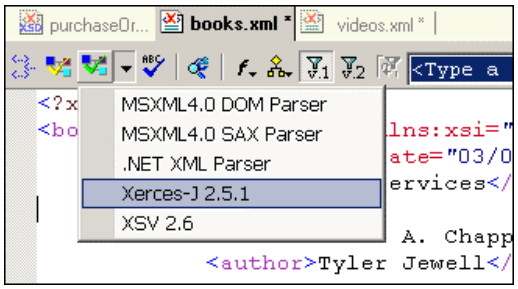


Fig. 3.4 *Herramientas de validación en Stylus Studio 5 Professional Edition.*

Éste es un editor que cuenta con herramientas para crear documentos XML. Su característica principal consiste en tener una arquitectura abierta de validación y de parsing XML lo cual garantiza un soporte de desarrollo que incluye el mismo procesador y validador XML que se utiliza en el ambiente de desarrollo.

Tipo de herramienta: Análisis de documentos bien formados y válidos (DTD).

Para más información visitar: ***http://www.stylusstudio.com/xml_parsers.html***

3.3 XML y las Hojas de estilo

Una diferencia de XML con respecto a HTML es que no tiene propiedades de visualización predefinidas para elementos específicos; para compensar tal situación, es necesario incluir un método que nos permita modificar la forma en que se visualizan los documentos. Las hojas de estilo contienen las descripciones de cómo se deben mostrar los elementos del documento XML. El objetivo principal al usar esta técnica es permitir, mediante la separación del contenido y de la presentación, modificar la visualización de un documento sin necesidad de modificar los contenidos del mismo.

3.3.1 Tipos de Hojas de Estilo

En la actualidad existen dos tipos de hojas de estilo disponibles para poder visualizar documentos: las CSS (Cascading Style Sheets), hojas de estilo en cascada y el XSL (eXtensible Style Language), lenguaje de estilos extensible. Ambos se derivan del DSSSL, el cual es un estándar internacional sobre lenguajes para hojas de estilo. Especificamente se derivan del DSSSL-O que es un perfil de DSSSL sin algunas funcionalidades y al que se le han incorporado nuevas posibilidades para hacerlo más propicio para trabajar con documentación en línea.

Diferencias entre XSL y CSS:

XSL	**CSS**
Utiliza notación XML	Utiliza su propia notación
El árbol objeto del programa puede ser totalmente diferente al árbol fuente.	El árbol objeto del formato es casi siempre el árbol fuente
La herencia de propiedades de formato se da sobre el árbol destino del formato.	La herencia de las propiedades de formato se da sobre el árbol fuente.
Independiza el contenido de un documento permitiendo transformaciones muy sofisticadas, todo ello sin modificar el documento XML original, que podrá ser utilizado en otros destinos simplemente cambiando sus hojas de estilo.	La estructura de un documento XML es idéntica a la estructura de su presentación, es decir, todos sus componentes de aparecer en la presentación (pueden no aparecer) lo harán en el orden que se definen dentro del documento XML.
Los cambios en la presentación no afectan a la estructura del documento XML original.	La mayoría de los cambios necesarios en la estructura de la presentación llevarán consigo asociados cambios del contenido XML original.
Está dirigido a operaciones de formato complejas, que suelen tener fases de transformación del documento XML.	Está más orientado a operaciones de formatos sencillas para el tratamiento de documentos *online*.

XSL posee las siguientes características que lo complementan:

- Generación de etiquetas para datos.
- Composiciones complejas utilizando elementos HTML.
- Aparición de datos en más de una ocasión a lo largo de la presentación.
- Acceso a la información almacenada en los atributos de los elementos.
- Reordenación de datos.
- Comportamientos dinámicos.

Al momento de elegir un lenguaje a utilizar para visualizar un documento debemos conocer si la estructura del documento se puede mostrar, el navegador a utilizar (no todos los navegadores admiten los dos tipos de hojas de estilo).

El Microsoft Internet Explorer 5 y posterior y el Netscape Navigator soportan hojas de estilo de tipos XSL y CSS.

3.3.2 UTILIZACION DE CSS

Las hojas de estilo CSS se han utilizado en conjunto con HTML, por lo que a la fecha existen una gran diversidad de desarrollos basados en esta técnica; con XSL es distinto ya que aún no ha alcanzado el mismo nivel de madurez.

CSS y XSL se pueden utilizar con XML. CSS es más adecuado utilizarlo en documentos con una estructura lineal para cuya presentación no sea necesario mucho manejo de elementos. XSL es el adecuado para trabajar con secuencias XML altamente estructuradas y que requieren una amplia manipulación de su estructura para ser mostradas.

La tendencia es que poco a poco CSS y XSL empiecen a converger en la implementación de objetos de formato y características comunes. Cada lenguaje está bien definido y por ende XSL no sustituirá a CSS ya que ambos tienen objetivos distintos. El punto de convergencia está precisamente en la utilización de aplicaciones XSL en el servidor para transformar documentos XML complejos en otros mucho más simples y CSS del lado del cliente para mostrarlos (este método sería compatible con todos los navegadores con capacidad para procesar CSS).

3.3.2.1 Concepto de Hojas de Estilo en Cascada.

Las hojas de estilo en cascada (CSS Cascade Style Sheets) provienen de un estándar desarrollado por el World Wide Web Consortium (W3C) con el objeto de lograr una mayor flexibilidad en las páginas Web al momento de visualizarse. Actualmente existen dos versiones de estándres CSS reconocidas CCS1 y CCS2. Se trabaja sobre la versión CCS3, aún no finalizada.

La razón principal por la que el W3C nombra a estos documentos como "hojas de estilo en cascada" es porque permite utilizar múltiples estilos para particularizar en la apariencia que deseemos utilizar en ese momento y el navegador sigue las reglas en "cascada" para determinar la prioridad y resolver posibles conflictos de especificación.

Algunas de las ventajas del uso de hojas de estilo en cascada CCS son:

- mayores posibilidades para el formato y presentación
- un mayor control sobre el documento
- mayor facilidad en la personalización de los documentos.
- Utilización de márgenes, sangrías, diferentes tamaños, colores de fondo, características de formato que utilizan los diseñadores y que el conjunto de etiquetas estándar no soporta directamente.

- Algunas alternativas poco prácticas para aplicaciones de formato como el uso de <BLOCKQUOTE> se dejan de utilizar.

- Permite la modificación del formato de cada una de las páginas Web que comprenden un sitio determinado sin necesidad de realizar los cambios a cada una de las páginas de manera individual.

- Reduce la cantidad de etiquetas utilizadas en el documento y por tanto lo hace ver más ordenado.

- Permite el cambio de diseño a través del uso de clases.

3.3.2.2 Utilización de una hoja de estilos CSS para visualizar XML

Las reglas especificadas dentro de la hoja de estilos CSS se aplican a diferentes elementos que componen el documento HTML a visualizarse.

Sintaxis Básica de una hoja de estilos

Un estilo o regla de estilo consta de dos partes principales:

- ***El selector***. Es el elemento al cual se va a aplicar el estilo

- ***La declaración***. Son las propiedades que conforman el estilo.

La forma más simple de selector dentro de un documento XML es cualquiera de los elementos definidos por el usuario. A este selector le siguen una lista de propiedades que afectarán al elemento/etiqueta y que se enlistan dentro de un par de llaves "{" y "}".

Cada propiedad se define por un par del tipo:

nombre_propiedad : valor

Para aplicar diferentes propiedades a un mismo elemento/etiqueta se hace mediante la separación de las mismas mediante puntos y coma (";"). La estructura básica de una regla de estilo se asemejaría a la siguiente:

```
Etiqueta {
Nombre_propiedad_1: Valor_1;
Nombre_propiedad_2: Valor_2;
Nombre_propiedad_3: Valor_3;
...
Nombre_propiedad_n: Valor_n;
}
```

Opciones de sintaxis

Agrupamiento de definiciones

Cuando más de una etiqueta/elemento comparten la misma definición de estilo se enlistan dichos elementos/etiquetas separados por comas en la cabecera de la definición del estilo común, sin importar el orden en que se coloquen. La estructura es de la siguiente forma:

Etiqueta1, Etiqueta2,...,Etiqueta N {...}

En el siguiente ejemplo, los elementos *percepcion*, *deduccion* y *plaza* deben mostrarse de forma resaltada (fuente negrita de 15 pts), ya que son importantes para el magisterio:

```
percepcion, deduccion, plaza {
    font-size: 15pt;
    font-weight: bold;
}
```

Definición del contexto de los selectores

Esta sintaxis se aplica cada vez que queramos aplicar el estilo cuando un elemento se encuentre dentro de otro, de la forma siguiente:

elemento_superior elemento_inferior {...}

Si queremos forzar la aplicación de un estilo a un elemento que sea descendiente directo de otro se indicará de la siguiente forma:

elemento padre > elemento hijo {...}

Comentarios

La forma de utilizar los comentarios en las hojas CSS es muy parecida a la utilizada en el lenguaje de programación C. Podemos ver su utilización en el siguiente ejemplo:

```
percepcion, deduccion, plaza {
    font-size: 15pt;        /* Tamaño de letra de 15 puntos */
    font-weight: bold;      /* estilo negrita */
}
```

Como nota podemos mencionar que los comentarios no pueden anidarse y los parser CSS1 toman los comentarios como espacios en blanco.

Clases como selectores

Para lograr un mayor nivel de especialización sobre los elementos, las hojas de estilo CSS contemplan *clases* de elementos. Esto nos permite diferenciar la presentación de

elementos del mismo tipo. Para definir clases, se añade a un selector dado el nombre de la clase:

selector.clase {...}

3.3.3 XSL

XSL y CSS tienen rasgos distintos y sin embargo los dos comparten algunas propiedades de formato específicas. La forma en la que trabajan sus procesadores marca la diferencia.

El motor de CSS lee el documento XML original y le coloca los arreglos de formato con las reglas que definen la hoja de estilos CSS asociadas. El procesador XSL analiza el documento XML fuente, lo examina y va ejecutando las instrucciones descritas en un documento XSL asociado, transformándola en un nuevo documento. Las transformaciones que se producen resultan en la creación de un nuevo árbol XML. Esta separación que se crea entre el documento fuente y los árboles de resultado respeta las reglas de XML al separar el contenido del documento de su presentación, permitiendo que tanto la gramática como la estructura del documento XML fuente permanezcan independientes de la estructura y el lenguaje de presentación.

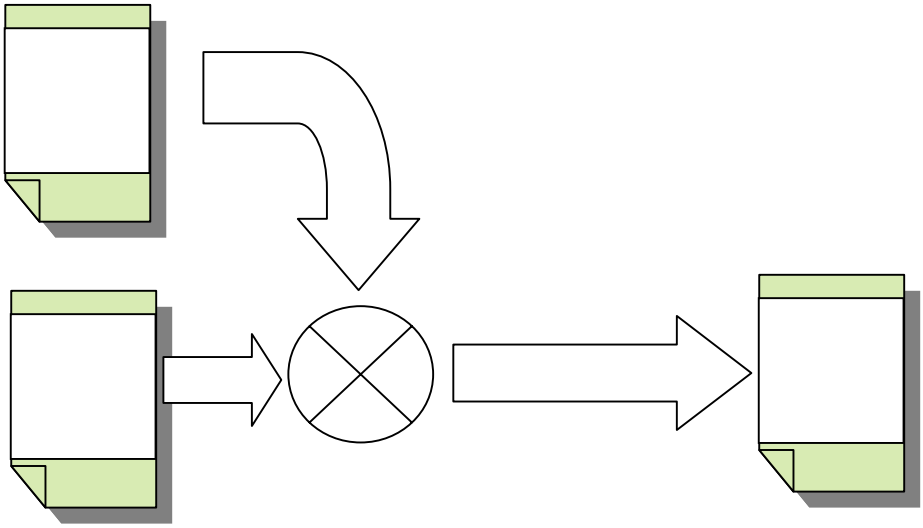


Fig. 3.5 *Uso de XSL para transformar un documento XML.*

3.3.3.1 Componentes del XSL.

Como parte del concepto general de XSL está la definición de tres lenguajes relacionados con el estándar, tales como:

- El lenguaje XSL. Se conforma de los objetos de formato descritos con vocabulario XML.
- El lenguaje XSLT o XSL Transformations. Es un lenguaje de marcas que describe la forma en que XSL permite la transformación del documento XML original. XSLT

analiza y procesa un documento XML (fuente), transformándolo en una versión distinta del documento (ó árbol resultado) basándose en los filtros y patrones definidos en la hoja de estilos.

- El lenguaje Xpath o XML Path Languaje. La partición y la definición del curso que tomarán los "trozos" de código están descritos a través de este lenguaje. Este lenguaje es utilizado por XSLT para definir expresiones y rutas locales que nos lleven a la creación de transformaciones XSL avanzadas.

3.3.3.2 Conceptos Previos

Tal y como se ha descrito en el capítulo 1, la estructura básica de un documento XML se compone de un elemento principal que define el documento llamado elemento raíz, a partir del cual se estructuran el resto de los elementos, los cuales serán hijos del raíz o hijos de los hijos y así sucesivamente hasta llegar a elementos que no tienen hijos (que se conocen como nodos hoja). Por consiguiente, forman una estructura de árbol en la que se tiene un nodo raíz, nodos intermedios y nodos hoja.

Los conceptos que hay que definir antes de explicar los componentes de XSL son:

- **Regla o plantilla.** Es el bloque básico de construcción del XSL; indica sobre qué secuencias de marcas/etiquetas se aplica la regla y la forma en que las marcas/etiquetas deben formatearse o modificarse. Una regla específica se aplica al elemento raíz del documento, otras reglas se aplican para las distintas marcas/etiquetas y existen reglas por defecto que se aplican a las marcas/etiquetas que no tienen alguna regla que las defnina.
- **Patrón.** Es la parte que conforma una regla XML que especifica la marca/etiqueta la cual se incluye dentro de un documento XML y que será formateada o modificada.
- **Acción.** Es otra parte de una regla XML la cual especifica el modo en el que el patrón puede modificarse o formatearse.

A continuación se describen las consideraciones a tomar y los ejemplos de cada uno de los conceptos:

3.3.3.3 Reglas o Plantillas XSL

La hoja de estilos XSL más básica está formada por una plantilla con la estructura que se desea obtener e identifica qué datos del documento XML fuente deben ser insertados a esta plantilla.

XML puede trabajar fusionando datos regulares y repetitivos de Documentos (*template-driven*) con platillas específicas o también con datos irregulares y recursivos de múltiples plantillas (*data-driven*) en el que la estructura de datos define las estructuras de salida.

Básicamente la estructura de las plantillas XSL está formada por un elemento de inicio, un cuerpo que compone la plantilla y el cual se utilizará en caso de que se dé la coincidencia o selección sobre el nodo del documento fuente y un elemento de cierre. Por lo tanto, la estructura general tendrá la siguiente forma:

```
<apertura del elemento XSL>
      Cuerpo de la plantilla
</cierre del elemento XSL>
```

3.3.3.4 Patrones XSL

Los patrones XSL se definen mediante una sintaxis simple y concisa para identificar nodos en un documento XML, los cuales se basan en los tipos de los nodos, su nombre, su contenido y el contexto en el que se dan (en relación con otros nodos). Cada elemento del documento XML fuente es contrastado con los patrones definidos en el documento XSL, los cuales en el caso de coincidir se incorporan al árbol de resultado que generan como salida.

Los patrones XSL utilizan una sintaxis muy similar a la empleada en los sistemas de archivos basados en jerarquías de directorios, o también la que define un URL dentro de un navegador Web. Por tanto la estructura básica de un patrón XSL define una ruta a través de el documento XML fuente y está formada por una lista de elementos separados por los caracteres de barras inclinadas ("/"). Para hacer referencia a un elemento se podrá hacer de la manera siguiente:

```
/raiz/predecesor/elemento/sucesor/..
```

Ejemplo:

`empleado/nombre` encajaría en cualquier nodo nombre que son nodos directos de un nodo empleado.

Se pueden utilizar distintos operadores para definir el tipo de filtro que se va a seguir.

- El operador *, se denomina también operador *comodín*. Se puede utilizar este operador en XSL para describir que el elemento es desconocido. P.Ej. `/BD_Docentes/*/*/estado` puede referirse tanto al estado de ocupación de una plaza como al estado de la plaza.

- Los operadores [y], se utilizan para operaciones de filtrado. El patrón que sigue tendría coincidencias con los elementos `Ocupacion` hijos de `Ocupacion_plazas` y que a su vez tienen un hijo denominado `folio`. `/BD_docentes/Ocupacion_plazas/Ocupacion[folio]`

Estos operadores son los principales, para una mayor referencia consúltese la Bibliografía indicada al final de esta Tesis.

3.3.3.5 Elementos de Selección

Los patrones de selección hacen uso de los elementos XSL: **<xsl:apply-templates>**, **<xsl:value-of>** y **<xsl:for-each>.**

Elemento <xsl:apply-templates>

Sintaxis:

```
<xsl:apply-templates
  select = patrón
  mode = nombre cualificado>
</xsl:apply-templates>
```

Atributos:

- **select:** está formado por un patrón XSL.
- **mode:** permite al elemento procesarse varias veces con resultados distintos.

Funcionamiento:

Realizan la selección de varios nodos utilizando para ello la consulta definida en el atributo select. En caso de no existir el atributo se seleccionan todos los hijos del nodo actual. El elemento <xsl:apply-templates> le indica que aplique los elementos <xsl-template> para todos los nodos seleccionados en los que la expresión **match** coincide con el nodo en cuestión.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
<HTML>
      <HEAD><TITLE>EjemploXSLa.xsl</TITLE></HEAD>
      <BODY>
         <CENTER>
      <H2>Patrón asociado con el <U>nodo Raíz</U></H2>
      <xsl:apply-templates />
         </CENTER>
      </BODY>
</HTML>
</xsl:template>

<xsl:template match="Empleados">
      <H3>Patrón coincidente con el <U>nodo de Empleados</U></H3>
</xsl:template>
</xsl:stylesheet>
```

El patrón de ajuste cuenta con un atributo **match** que coincide con el elemento `Empleados` del documento XML. Al incorporar el elemento `<xsl:apply-templates>` dentro del cuerpo del elemento <xsl:template>, le estamos indicando al procesador XSL que debe procesar el resto de los patrones que queden en la hoja.

Elemento `<xsl:value-of>`

Sintaxis del elemento:

```
<xsl:value-of
      select = patrón
      disable output-escaping = "yes" | "no" >
</xsl:value-of>
```

Atributos:

- **select:** incluye un patrón XSL. Por defecto su valor es "." que hace referencia al nodo actual.

- **disable-output-escaping:** nos ayuda a decidir si el texto obtenido debe contener caracteres escapados.

Funcionamiento:

No maneja nodos, sin embargo permite insertar los contenidos de los mismos. Tiene la función de devolver la primera cadena encontrada que se obtuvo como resultado de la consulta sobre el contexto actual y que coincide con el valor del atributo **select**.

Ejemplo:

El ejemplo que sigue enseña el funcionamiento del elemento <xsl:value-of> de XML

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
<HTML>
      <HEAD><TITLE>EjemploXSLb.xsl</TITLE></HEAD>
      <BODY>
         <CENTER>
      <H2>Patrón asociado con el <U>nodo Raíz</U></H2>
      <xsl:apply-templates />
         </CENTER>
      </BODY>
</HTML>
<xsl:template>

<xsl:template match="Empleados">
      <H3>Patrón coincidente con el <U>nodo de Empleados</U></H3>
</xsl:template>

<xsl:template match="alumno">
      <H3>Patrón coincidente con el <U>elemento de Empleado</U></H3>
      <xsl:value-of/>
</xsl:template>
</xsl:stylesheet>
```

Al no utilizar el atributo **select** del elemento **<xsl:value-of>** trabajamos con el nodo seleccionado actualmente.

Elemento `<xsl:for-each>`

Sintaxis del elemento **<xsl:for-each>**:

```
<xsl:for-each
  select = patrón
</xsl:for-each>
```

Atributos:

select: contiene un patrón XSL. Por defecto se seleccionan todos los hijos del nodo actual.

Funcionamiento:

Permite especificar los patrones de selección, de esta manera se pueden trabajar con todas las instancias (nodos) resultado de la consulta contenida en su atributo **select**. Indica en qué contexto se va a realizar la iteración, sobre los nodos resultado de una consulta.
Ejemplo:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
<HTML>
      <HEAD><TITLE>EjemploXSLc.xsl</TITLE></HEAD>
      <BODY>
         <CENTER>
      <H1>Listado de empleados</H1>
         </CENTER>
      <TABLE BORDER="1" ALIGN="CENTER">
      <TR ALIGN="CENTER">
        <TD COLSPAN="3">Empleado</TD><TD>SEXO</TD><TD>FECHA
NACIMIENTO</TD>
      </TR>
      <TR ALIGN="CENTER">
        <TD>Nombre</TD>
        <TD>Ap. Paterno</TD>
        <TD>Ap. Materno</TD>
        <TD> </TD>
        <TD> </TD>
      </TR>
      <xsl:for-each select="Empleados/Empleado">
        <TR ALIGN="CENTER">
      <TD><xsl:value-of select="Nombre"></TD>
            <TD><xsl:value-of select="ApPaterno"></TD>
      <TD><xsl:value-of select="ApMaterno"></TD>
      <TD><xsl:value-of select="sexo"></TD>
      <TD><xsl:value-of select="fechanacimiento"></TD>
        </TR>
      </xsl:for-each>
      </TABLE>
```

```
      </BODY>
</HTML>
<xsl:template>

</xsl:stylesheet>
```

En este ejemplo se fusiona un documento XML regular y repetitivo con una plantilla HTML para su visualización.

Se rellenan todas y cada una de las filas de la tabla destino con los datos de los elementos empleado que contiene el XML fuente.

Referencias.
Xerces: http://xml.apache.org/xerces2-j/xni-xerces2.html
Otros Parsers XML: http://www.topxml.com/parsers/other_parsers.asp

Capítulo 4

Antecedentes del Subsistema de Manejo de Trámites Administrativos (SIMIP) del Sistema de Personal de la SEC Veracruz

Introducción

El Sistema Integral de Movimientos e Incidencias de Personal (SIMIP) fue concebido con el propósito de atender las necesidades de modernización y sistematización de la SEC (Secretaría de Educación y Cultura) del estado de Veracruz, el cual abarca el Área de Recursos Humanos (integración de los subsistemas satélite) y tiene comunicación con el Área de Nóminas mediante accesos a la Base de Datos.

El sistema cuenta con 7 módulos principales, que se describen a continuación:

Módulo de Administración General.- Controla y maneja los usuarios y parámetros específicos de trabajo para cada Área detallando la seguridad de cada uno de los módulos operativos.

Módulo de Apertura de Trámites (Ventanilla Única).- Al estar basado en un ambiente distribuido, agiliza el ingreso de trámites ya que éstos pueden realizarse desde las oficinas centrales o desde los niveles educativos.

Módulo de Evaluación de Trámites (Analistas).- Proporciona al trabajador la información suficiente para una validación segura de los trámites, que permite generar el aviso de movimiento de personal respectivo.

Módulo de Creación y Administración de Plazas.- Permite la modificación y consulta en línea de la base de Plazas registradas en el sistema.

Módulo de Auditoría.- Almacena y genera reportes de todas y cada una de las acciones de los usuarios con Acceso al Sistema Integral de Trámites.

Módulo de Control y Seguimiento a Trámites.- Este módulo permite llevar un control de cada una de las fases por las que pasa el Aviso de Movimientos de Personal (AMP) en su trayecto por cada una de las Áreas que requieren autorizarlo. Asimismo lleva el registro de los rechazos que pudieran suscitarse, permitiendo dar un informe preciso de la ubicación del AMP en la primera etapa de elaboración del trámite.

Módulo de Envío y Recepción de Mensajes y Tareas.- Permite el uso de mensajes en forma automática, en la sucesión de los procesos de cada trámite, registrando el tiempo de atención.

4.1 Descripción del proceso administrativo de Operación de Trámites.

4.1.1 Ventanilla

Recibe las propuestas o movimientos de personal por medio de cédulas de movimientos de personal que son generadas por parte de Supervisión escolar (Niveles), el interesado y los sindicatos. Estos últimos manejan propuestas sindicales antes de la elaboración de la cédula. Se anexa la documentación requerida a la cédula y se revisa para ser turnada a análisis.

En caso de estar completo se le asigna un número de control y se sella y se captura en una base de datos de propuestas y se imprime un reporte simple que no es el AMP (Aviso de Movimientos de Personal) tal como se muestra en la figura 4.1.

La documentación es revisada utilizando una guía de revisión simple que no llega a ser un manual de procedimientos. En este departamento la gente trabaja por la experiencia que tiene y no se guía por procedimientos.

Cuando se trata de nuevo ingreso, Ventanilla solicita el número de personal a la Secretaría de Finanzas y Planeación (SEFIPLAN) y ésta a su vez regresa el número de personal a la Ventanilla.

4.1.2 Análisis

Recibe el documento de control de Ventanilla, lo sellan de recibido e integran las propuestas por número de zonas escolares a cada analista; pueden ser 3, 5 o más zonas escolares por cada analista. Cada analista firma una copia de recibido del oficio o planteamientos de supervisión escolar por cada zona.

En el documento de control se menciona que es alta y baja como tramite motivo, pero en el planteamiento del reporte sí se refleja el motivo real del movimiento tal como cambio de plaza, licencia por enfermedad, etc. En esa fecha no existía un sistema que controlara estos movimientos.

Los errores mas frecuentes ocurren desde el inicio en donde no se menciona que tipo de movimiento es.

También ocasionan errores los trámites que entran por enlace administrativo y llegan directamente al Área de Análisis.

Los trámites llegan al Área de Análisis por la Ventanilla cuando proceden de forma normal, pero de forma extraordinaria pueden recibirse de Enlace Administrativo, pudiéndose originar errores de duplicidad al ingresar por dos vías distintas.

Una vez asignado al analista se analiza la documentación soporte según el tipo de movimiento y elaboran la cedula de movimientos en caso de no presentarse errores, si es

rechazado se regresa al nivel (enlace administrativo) siempre y cuando se haya realizado por Enlace, Supervisión Escolar o por el interesado. Se regresa al sindicato cuando llega por ese medio.

Cuando envían el rechazo a Enlace Administrativo o a otro Nivel elaboran un oficio y un formato de rechazo en donde anotan el motivo de rechazo, el número de control de Ventanilla y su número de folio de rechazo, con todos sus documentos.

Cuando es rechazado y presentado nuevamente para su trámite llega directamente al Área de Análisis ya que anteriormente Ventanilla le había asignado su Número de control.

Una vez que el trámite procedió se consulta en el archivo histórico para verificar que todos los datos coincidan y se turna a Informática para que sea capturado y se genere el AMP.

El sistema con el que cuentan les permite consultar los Kárdex con antigüedad máxima de 1995. Para consultar los históricos del 95 y anteriores, lo hacen a través del Núm. de personal. Cada analista maneja sus propios históricos (cajón de escritorio). En caso de que se pierda algún Kárdex, consultan el expediente que se encuentra en el IPE. El esquema completo se ilustra en la figura 4.2.

En caso de que el trámite involucre el cambio en alguna plaza, éste se turnará al Área de Registro y Control de Plazas. En caso contrario (trámite normal), el trámite será manejado por Análisis en coordinación con el Área de Informática tal y como se muestra en la figura 4.4.

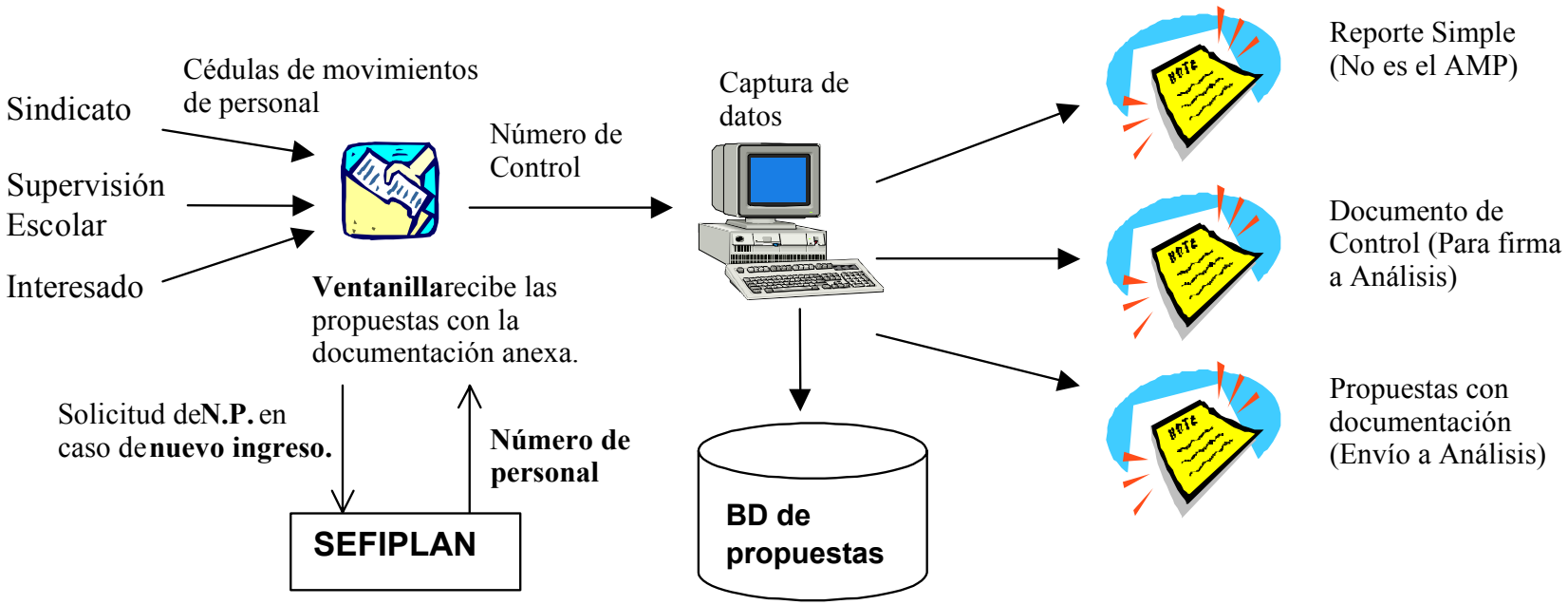


Figura 4.1 *Diagrama de operación de trámites en Ventanilla.*

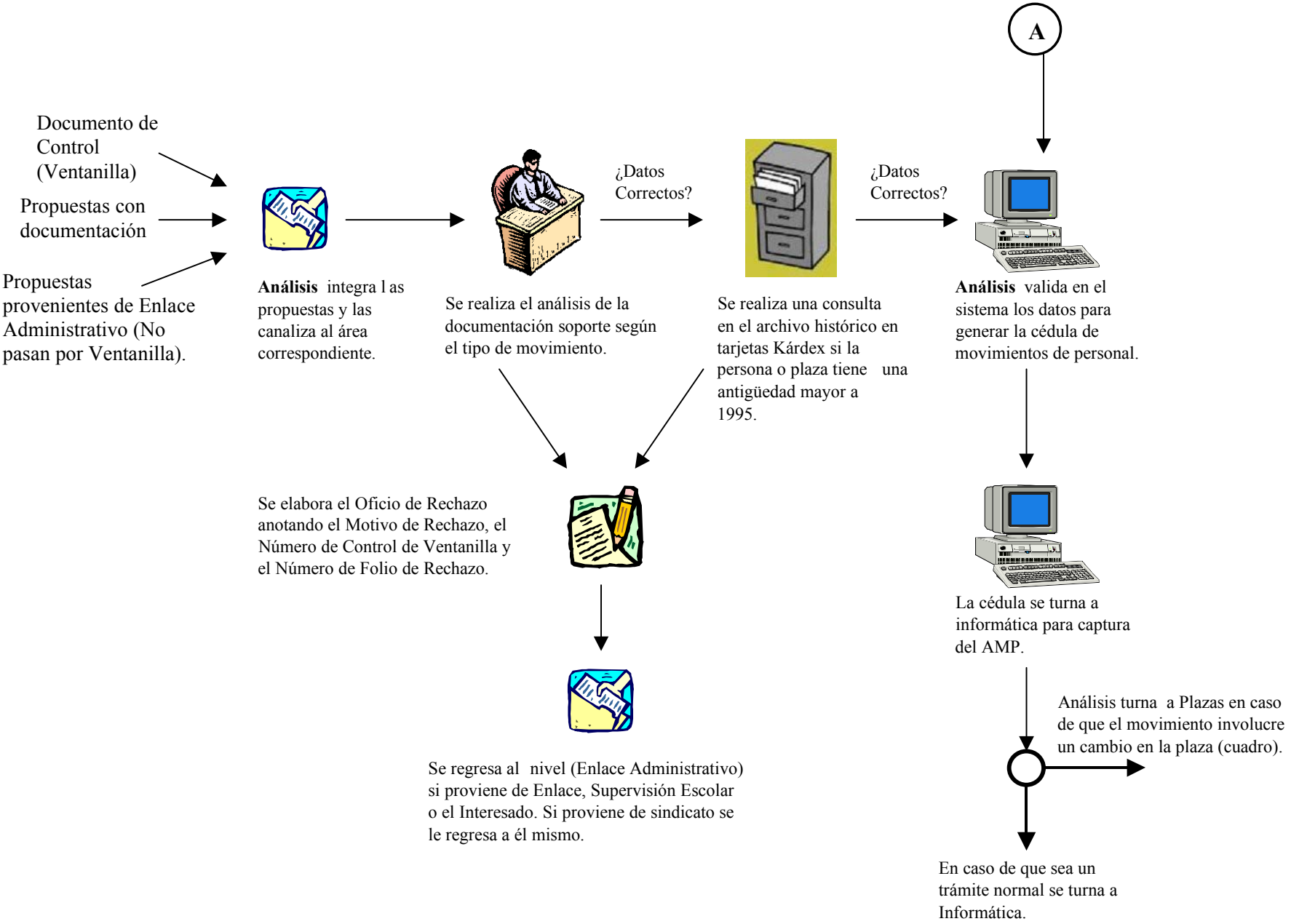


Figura 4.2 *Diagrama de operación de trámites en Análisis.*

4.1.3 Registro y Control de Plazas

El Área de Análisis, al determinar que el movimiento involucra algún cuadro, turna el trámite a esta Área. En general, las causas que pueden llevar a un caso de generación de cuadro son cuando el maestro cambia su tabulador (sueldo) y cuando entra un nuevo maestro a cubrir una plaza anterior, entre otras.

Existe un sistema de cuadros de cancelación creación en donde ingresan las propuestas. Es ahí en donde se imprime el cuadro, se valida y si es correcto se turna a Informática con las cédulas para generar los AMP.

El Área de Informática maquila y valida la información una vez que recibe el trámite de plazas. Realizan la impresión y devuelven el AMP impreso a plazas. Plazas vuelve a validar lo que recibe de Informática, ya que pueden presentarse errores como la no coincidencia del sindicato, la fecha, etc.

Al final, plazas elabora un oficio en donde se hace la petición a la Secretaría de Finanzas y Planeación (SEFIPLAN) para que les autoricen la cancelación y la creación de una nueva plaza (tomando en cuenta el control presupuestal que ellos manejan). El proceso administrativo para Plazas está representado en la figura 4.3.

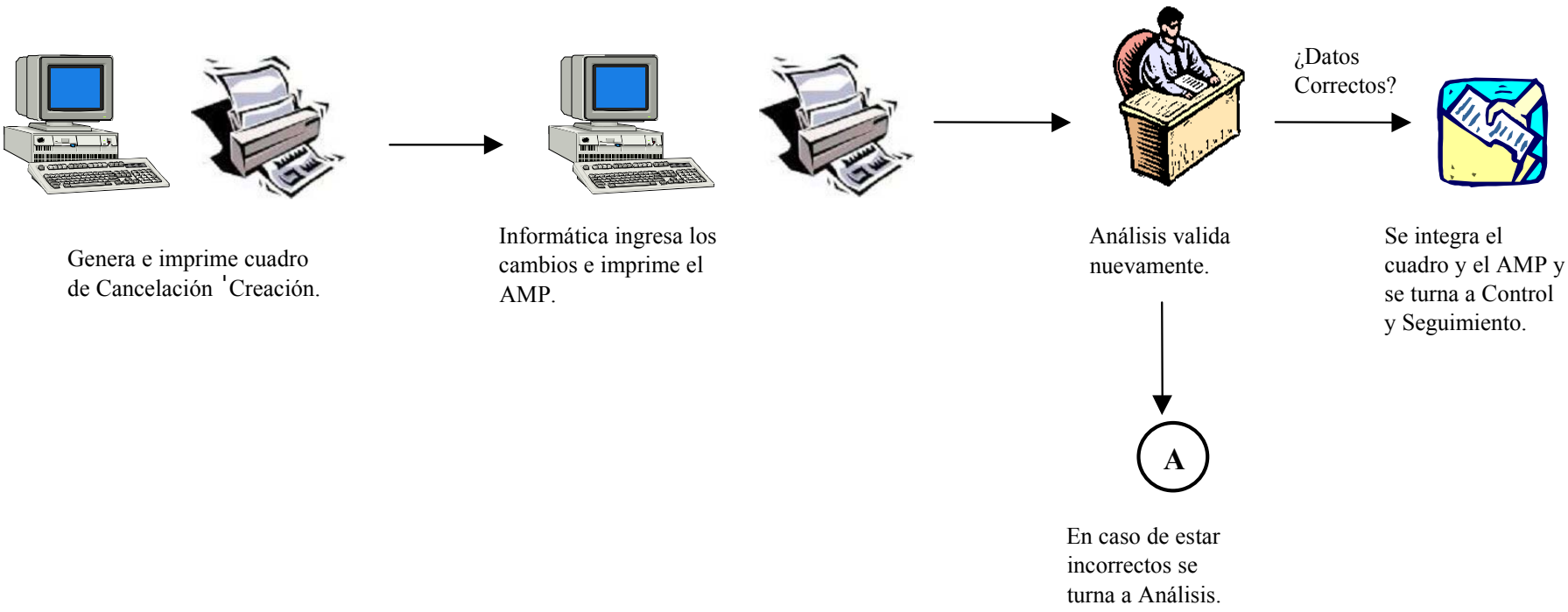


Figura 4.3 *Diagrama de operación de trámites en Plazas.*

Figura 4.4 *Diagrama de operación para trámite normal (Informática-Análisis).*

4.1.4 CONTROL Y SEGUIMIENTO

Esta Área envía la documentación a las distintas áreas y dependencias tales como Finanzas, sindicatos, el IPE y Supervisión Escolar, entre otras. El trámite pasa a su siguiente etapa cuando el área de Nóminas recibe tanto el original como el duplicado.

Reciben de plazas o análisis toda la documentación, especificando el total de movimientos.

Registran en el sistema el número de relación, No. de folio, fecha de envío y No. de oficio con el que se envía al Nivel.

Envía los movimientos a los diferentes Niveles firmando un acuse de recibido. Una vez que regresa vuelven a validar las firmas y registran en el sistema la fecha de devolución.

Separan las relaciones de AMP por las copias correspondientes para enviarlas a las diferentes áreas y dependencias. Esta Área tiene registro desde Abril del 2000 en adelante (del 99 y anteriores no tienen nada archivado). La ruta completa del trámite para Control y Seguimiento está representada en la figura 4.5.

El Área de Control y Seguimiento realiza aproximadamente 38,000 trámites al año.

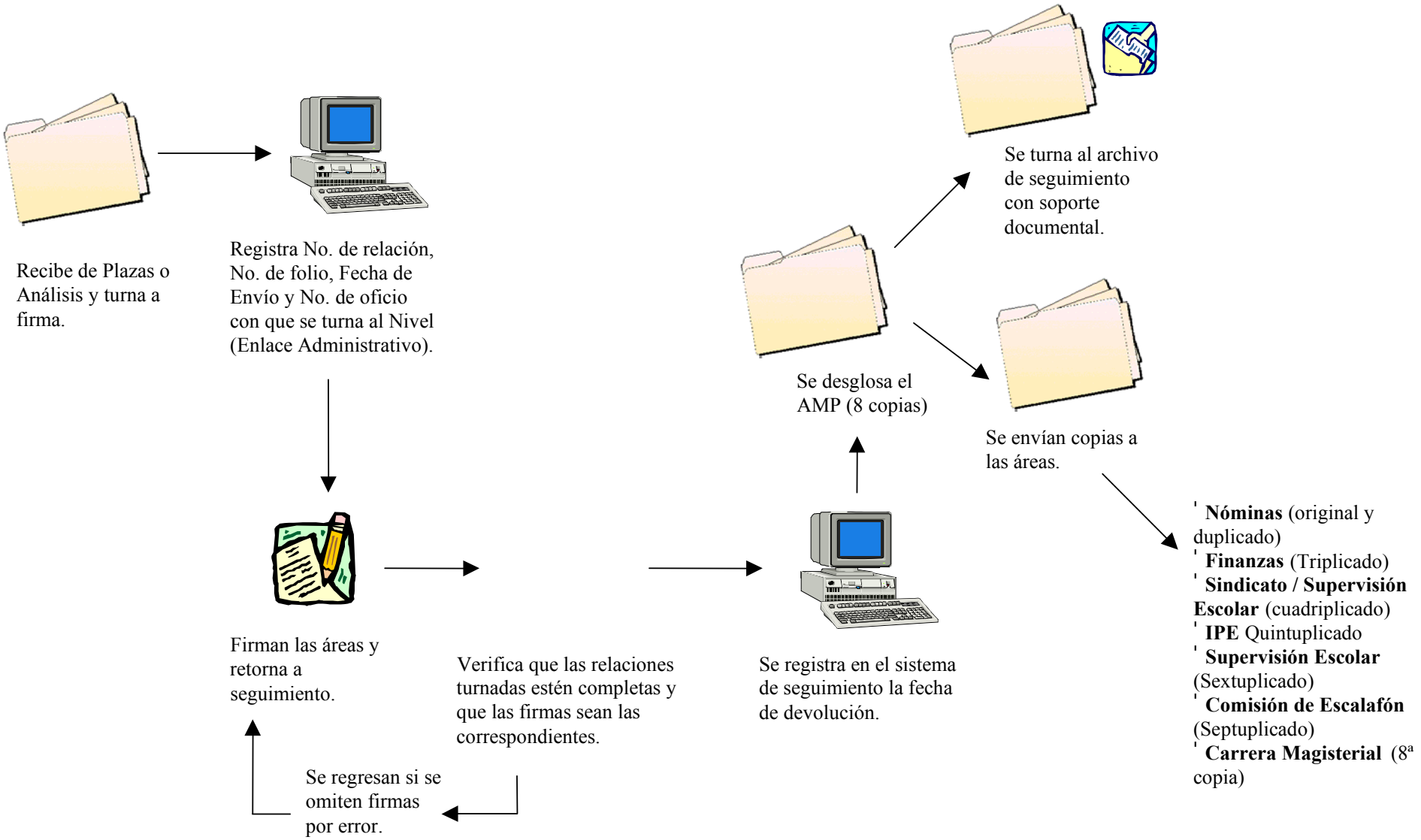


Figura 4.5 *Diagrama de operación para Control y Seguimiento.*

4.2 Automatización de la Operación de Trámites

El proyecto tecnológico emprendido con el objetivo de modernizar la administración de los Recursos Humanos pretende:

- Abatir el rezago de los movimientos de personal para su incorporación dinámica a la nómina.
- Simplificar y modernizar en forma integral la tramitación de los movimientos de personal con apoyo en los avances tecnológicos.
- Desconcentrar gradualmente trámites y movimientos de personal (que sean posibles) hacia las áreas educativas.
- Facilitar la aplicación de la normatividad justificando cada trámite con el motivo jurídico (norma o ley) correspondiente.

Los párrafos siguientes describen las deficiencias en el manejo administrativo y las razones por las cuales un nuevo sistema mejoraría el desempeño en la atención de trámites de la Subdirección de Recursos Humanos y en general, de la SEC.

De forma general:

- Se detecta la existencia de varios sistemas "satélite" creados por cada una de las áreas y que hacen redundante el flujo y la captura de información. Un sistema integral puede evitar esta redundancia ya que ayuda a compartir la información entre las áreas.
- Los trámites están concentrados en las oficinas de Recursos Humanos de la SEC, lo que complica al magisterio que tenga una residencia muy apartada la realización de trámites. Esto hace necesario la implementación de un sistema que permita capturar los movimientos desde los Niveles Educativos a través de la Intranet de la SEC.
- Los trámites se atrasan ya que desde la Ventanilla se permite la entrada de trámites claramente improcedentes, haciéndole perder tiempo al interesado a la vez que incrementan el tiempo de respuesta global de la Secretaría. Un sistema que traiga consigo un módulo de validación automática de trámites implantado desde la Ventanilla puede resolver que serán rechazados determinados trámites sin que éstos tengan que llegar al área de Análisis.
- Cada vez que el soporte documental es enviado de un área a otra, el área que recibe, firma una copia de la relación misma que se queda en el área emisora. Este procedimiento aunque puede automatizarse, tiene implicaciones de carácter normativo que impedirían que un sistema manejara vía electrónica los acuses de recibo. Mientras se tiene la confianza en el mismo se propone que al mismo tiempo que firman las áreas el acuse indiquen en el sistema que ya recibieron los documentos.
- Para consultar el estado de cualquier trámite es necesario recurrir a todos los sistemas existentes, ingresando a cada uno de ellos con interfases distintas dificultando el acceso a la información de la Secretaría. Y aún más, la información contenida en las bases de datos es difícil de homologarse es decir, no existe una forma práctica para convertirla en una sola, y sería este un proyecto muy complicado

de realizar. La opción viable es que un sistema uniforme permita ir construyendo con el tiempo los históricos necesarios para la operación de la Secretaría, a medida que transcurra el tiempo se irán realizando un mayor número de consultas sobre la nueva Base de Datos que sobre las BD anteriores.

- Los comunicados realizados entre las áreas son mediante oficios protocolizados que tardan un tiempo considerable en llegar al área correspondiente y que pueden traspapelarse. El sistema propone una comunicación mediante un pizarrón electrónico personalizado (messageboard) que facilita la comunicación entre las áreas.

A continuación se describen las mejoras propuestas para cada una de las áreas involucradas.

Ventanilla

Las propuestas sindicales y las cédulas de movimientos de personal se capturan en papel. Mediante un sistema automatizado se podrían capturar desde los niveles educativos evitando la recaptura en Ventanilla.

La documentación se revisa mediante una guía de revisión simple. Se puede integrar en el sistema una validación documental que permita al usuario verificar interactivamente la documentación requerida para cada trámite.

Análisis

El área carece de herramientas informáticas que faciliten mediante consultas la verificación de la validez o procedencia de un trámite. Lo hacen a través de archivos históricos y su información está guardada en tarjetas llamadas "kárdex". Mediante las nuevas herramientas se pueden consultar los históricos contenidos en las Bases de Datos de las distintas áreas. Esto se eliminaría gradualmente ya que la Base de Datos unificada reemplazaría a las BD de los sistemas anteriores.

Registro y Control de Plazas

Actualmente generan los cuadros de cancelación-creación mediante un sistema de cuadros, el cual tienen que alimentar nuevamente con las propuestas que llegan desde el área de Análisis. Mediante el sistema, Análisis enviaría la solicitud en forma de cuadro electrónico de cancelación-creación, por lo que Plazas sólo tendría que aplicarlo.

Control y Seguimiento

Esta área se apoya para dar seguimiento a los trámites de un sistema en donde registra las fechas de envío y recepción de documentos. El sistema integral mejoraría este procedimiento ya que no se tienen que volver a capturar los datos del trámite, solamente se elegiría el trámite por número de AMP para darle seguimiento.

4.3 Manejo actual en ASPs con ADO-DB

4.3.1 Conformación del Sistema Integral de Movimientos e Incidencias de Personal (SIMIP)

El Sistema Integral de Movimientos e Incidencias de Personal (SIMIP) actualmente consta de los siguientes módulos:

Administración General: Manejo de usuarios y parámetros específicos de trabajo para cada área con confiables medidas de seguridad como la administración de los accesos por página.

Módulo de Apertura de Trámites (Ventanilla Única): Agiliza el ingreso de trámites ya que tiene la capacidad de atender de manera simultánea a varios usuarios y se puede hacer desde cualquier lugar de la Intranet de la SEC. Evita gran parte de los rechazos ya que valida los trámites a través de su *Módulo de Validación*.

Las figuras que muestran el funcionamiento del módulo de forma gráfica son:

Figura 4.6, Recepción de trámites en Ventanilla. Es la vista desde arriba del modelo jerárquico que representa el flujo de información, desde la recepción y captura de propuestas hasta la impresión de facturas (comprobantes) y la emisión de propuestas corregidas para su ingreso a Análisis.

Figura 4.7, Captura de Propuestas. Captura todos trámites que vienen incluidos con el oficio y validando los datos según la zona escolar.

Figura 4.8, Módulo de Validación. Verifica cada uno de los trámites en búsqueda de otros movimientos (encadenados) que son necesarios para que éste sea considerado como procedente. En caso de no estar encadenado, aplica las funciones de validación que indican si un trámite procede o no con salvedades.

Figura 4.9, Rutina de Encadenamiento. Dentro del Módulo de Validación se encuentra esta función, la cual mediante consultas a la BD (Catálogo de encadenamiento y Datos de Ventanilla) se establecen criterios de comparación que permiten saber si el trámite está encadenado o no.

Figura 4.10, Validación para trámites que no están encadenados. Se genera un *query* con todas las validaciones requeridas para el movimiento en cuestión, de acuerdo con las claves previamente obtenidas de la Base de Datos. En un ciclo se van ejecutando todas y cada una de las validaciones y se van generando las salvedades de cada movimiento, mismas que se incluirán al momento de imprimir la factura (comprobante) con todos los trámites que acompañan al oficio.

Módulo de Evaluación de Trámite (Analistas): Proporciona al trabajador herramientas que proveen la información suficiente para una validación segura de los trámites; mismas que darán plena certeza al analista de que procede o no el trámite. Permiten la impresión posterior del Aviso de Movimiento de Personal (AMP) si es el caso.

Módulo de Creación y Administración de Plazas: Permite la impresión de los cuadros de cancelación-creación para realizar el trámite correspondiente.

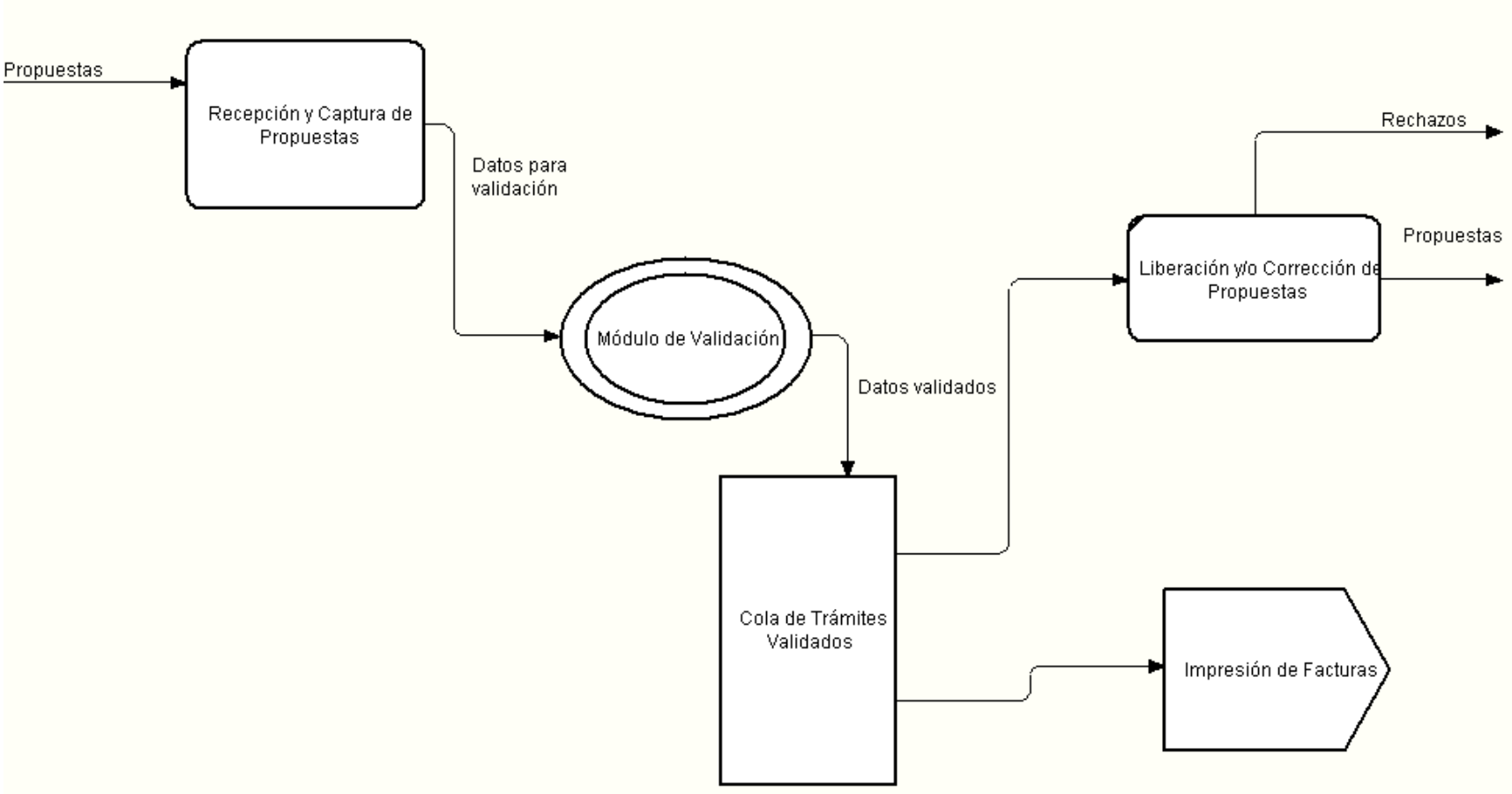


Figura 4.6 *Recepción de trámites en Ventanilla.*

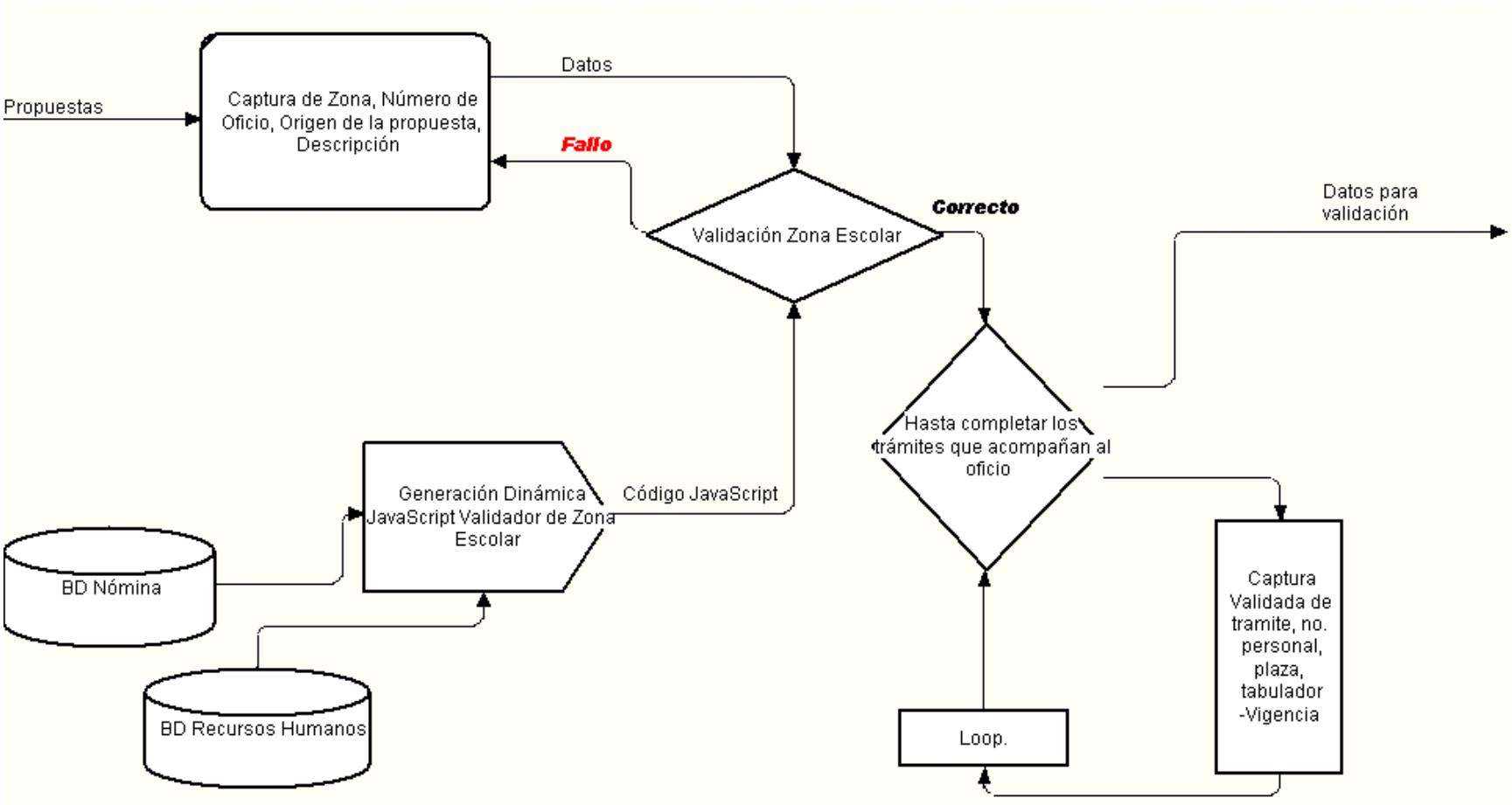


Figura 4.7 *Captura de propuestas.*

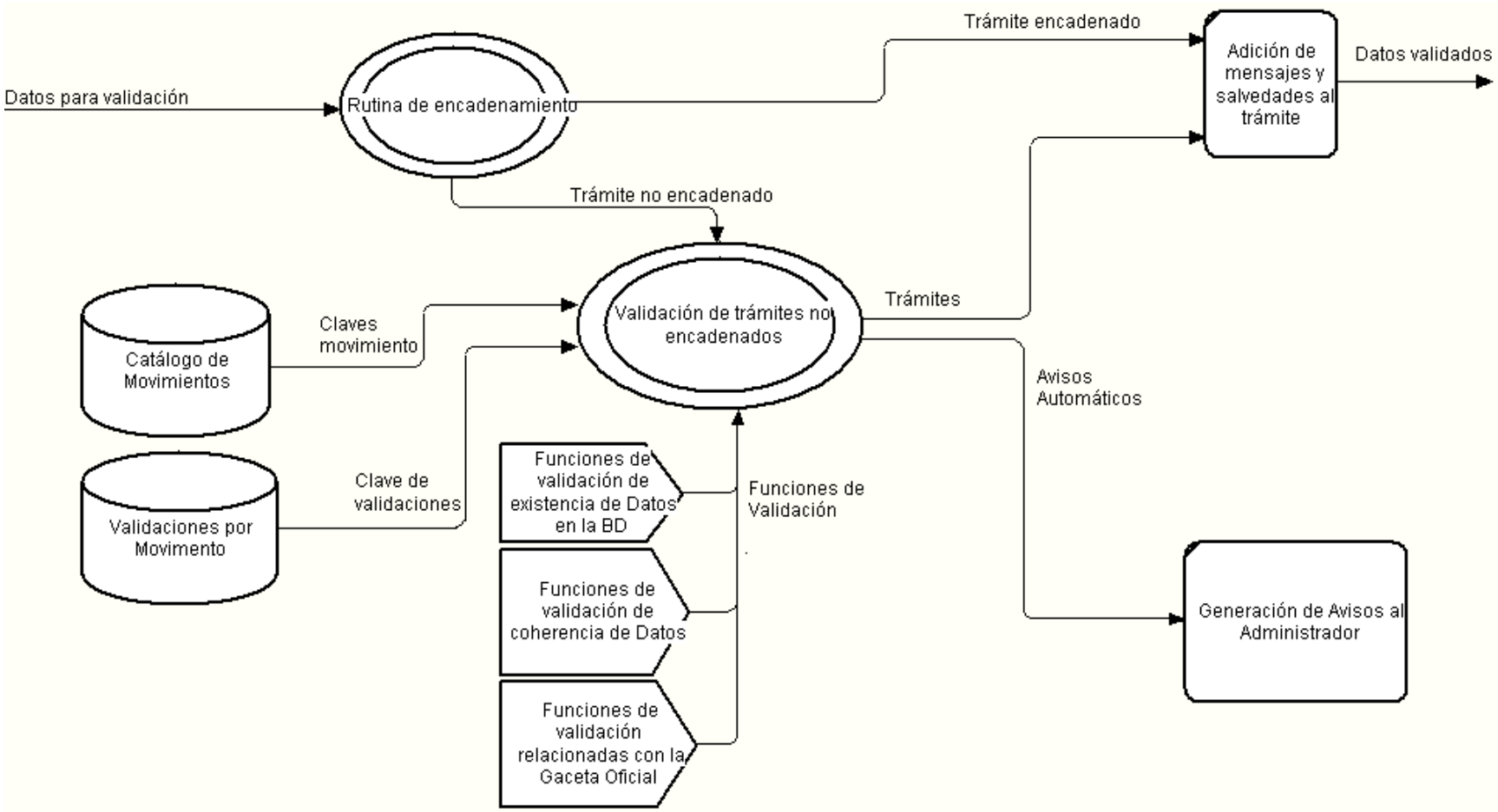


Figura 4.8 *Módulo de Validación.*

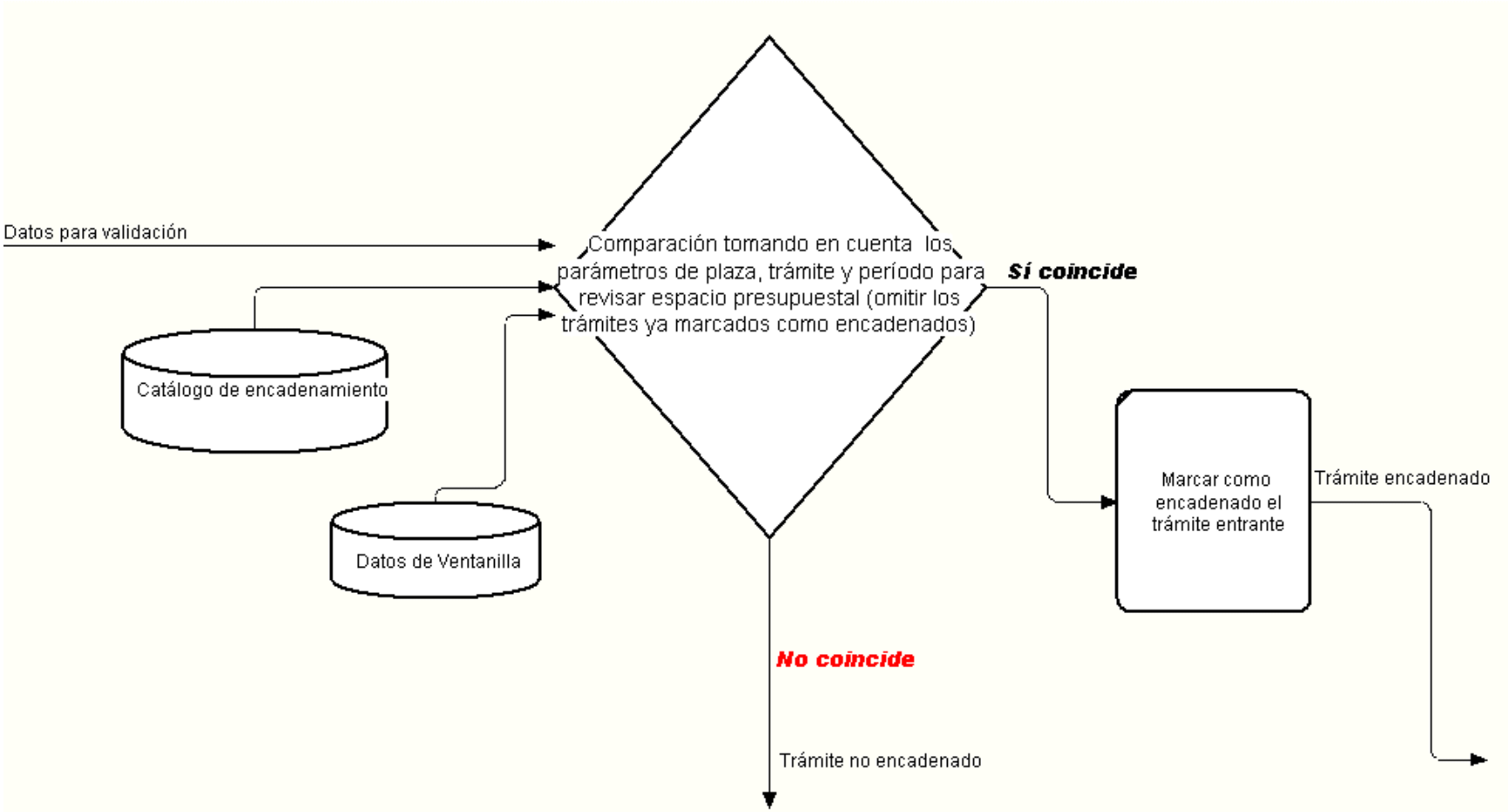


Figura 4.9 *Rutina de Encadenamiento.*

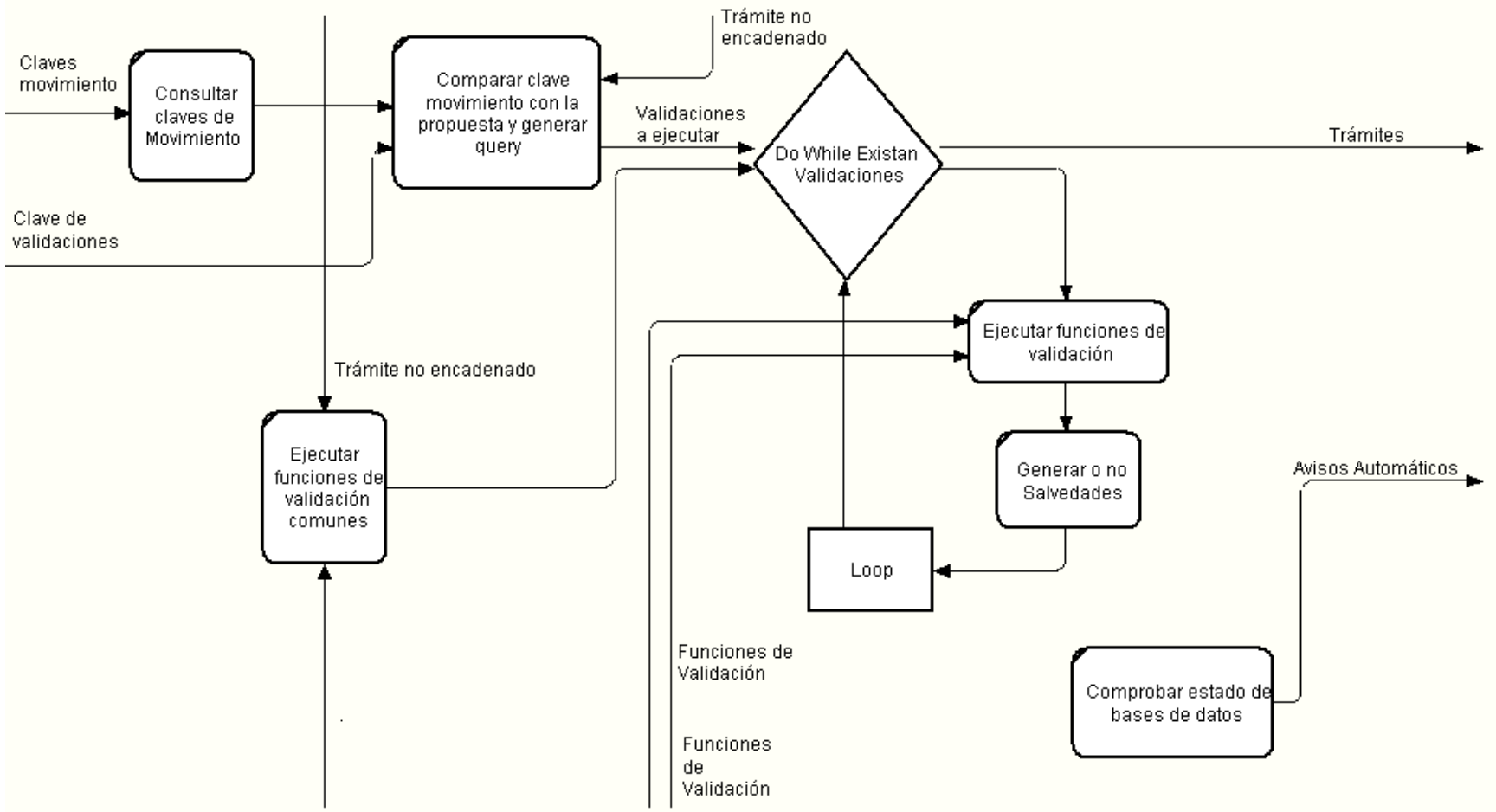


Figura 4.10 *Validación de trámites aue no están encadenados.*

Módulo de Control y Seguimiento: Permite, como su nombre lo indica, llevar el control y seguimiento de los documentos que se mandan a firmas a las áreas.

Módulo de Auditoría: Almacena todas y cada una de las acciones de los usuarios con acceso al Sistema Integral de Trámites.

Sistema de Envío y recepción de mensajes y tareas: Permite el uso de mensajes en forma automática, en la sucesión de los procesos de cada trámite, registrando el tiempo de atención.

4.3.2 Conceptos de ADO

ADO (ActiveX Data Objects) substituyó tanto a DAO (Data Access Object), como a RDO (Remote Data Object), que eran los sistemas previos que se usaban para acceder a las bases de datos, y bases de datos remotas, respectivamente. Tiene la mayor parte de la funcionalidad de ambos modelos, y sin embargo, es más sencillo de usar y de entender.

ADO es un intermediario entre el programa y la base de datos. El programa no ve la base de datos directamente, sino que hace todo el trabajo a través de ADO. Usando ADO, el programa se comunica con la base de datos, consulta, edita, inserta, borra, registros, añade tablas, etc. ADO a su vez se comunica con la base de datos a través de un "proveedor de datos".

Programa → ADO → Proveedor de datos → Base de datos

El proveedor de datos es un componente que se relaciona directamente con la base de datos. Hay un proveedor de datos por cada tipo de base de datos. Así, las bases de datos de tipo Access, SQL Server, Oracle, MySQL, tienen, cada una, un proveedor de datos específico.

La conexión ADO puede usar dos tipos de proveedores de datos, OLE DB y ODBC, siendo OLE DB el tipo nativo del proveedor.

Referencia: http://es.wikipedia.org/

4.3.3 Plataforma tecnológica

El sistema actual tiene como base de programación a las páginas Active Server Pages (ASP) utiliza Visual Basic Script en el lado del servidor y JavaScript en el lado del cliente. Como servidor Web utiliza MS IIS (Internet Information Server) V 5.0 corriendo sobre MS Windows 2000 Server.

El sistema tiene como plataforma de base de datos MS SQL Server 2000 corriendo con 4 diagramas de Integridad Referencial y alrededor de 110 tablas que almacenan los catálogos y archivos históricos correspondientes.

Se utilizó la tecnología ActiveX Data Objects (ADO) para tener acceso a la Base de Datos con la siguiente estructura:

- Definición del objeto de conexión del tipo "ADODB.Connection".
- Asignación de la cadena de conexión al objeto mediante el driver SQLOLEDB.
- Definición del objeto de consultas "ADODB.Recordset".
- Asignación de la cadena SQL al objeto de consultas (recordset) correspondiente.

Objeto Connection: Se utiliza para enlazar directamente la página Web y el servidor de bases de datos. Durante el establecimiento de la conexión se puede realizar cualquier operación sobre la base de datos.

Objeto RecordSet: Es una tabla de datos en donde se almacenan las consultas realizadas y de ahí se pueden obtener las filas (registros) y columnas (campos) que contienen los datos de nuestro interés.

Objeto Command: Es un comando SQL mediante el cual podemos ejecutar instrucciones SQL sobre la base de datos definida en el objeto Connection.

Estos objetos pueden existir por sí mismos. La figura siguiente muestra las relaciones existentes entre ellos.

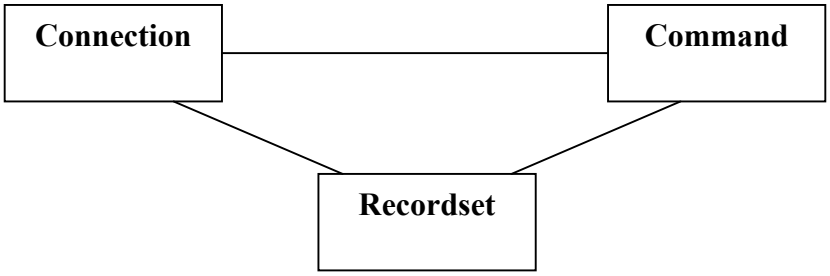
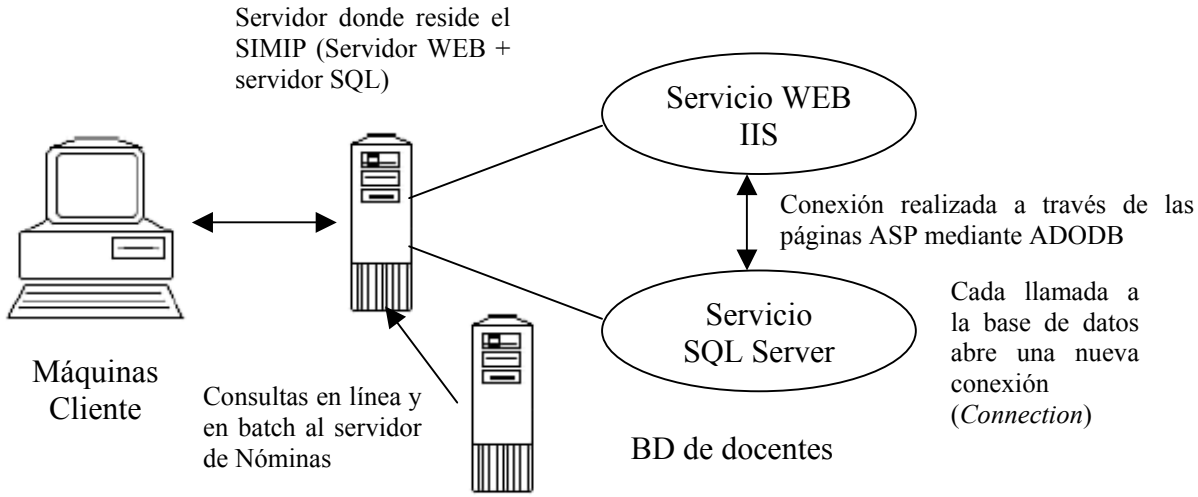


Figura 4.11 *Diagrama de Conexiones de Base de Datos del SIMIP.*



En este sistema no se explota toda la capacidad de SQL Server ya que además de que adolece de las ventajas de usar procedimientos almacenados (Stored Procedures), necesita estar en línea para funcionar cabalmente. Aunque la tecnología no limita a que otras aplicaciones accedan a la base de datos, es solamente mediante el sistema que dichos datos tienen sentido y preservan su integridad. Faltaría extender este concepto a aplicaciones que utilicen por ejemplo las ventajas de la movilidad para que la independencia sea más notoria.

No existe diccionario de datos con metainformación (información sobre la información)**.** Las cartas y documentos que imprime el sistema carecen de estructura y a lo más se pueden identificar los elementos básicos marcados exclusivamente por llamadas a campos de la base de datos (sólo parte de la semántica) en la parte correspondiente a la programación ASP; en lo referente a su impresión y almacenamiento éste se realiza en formato HTML plano. Un sistema informático no podría determinar, por ejemplo, en donde se localizan las advertencias y notas y qué prerrogativas legales avalan a todos los comprobantes que ha emitido la Ventanilla a lo largo de la utilización del sistema.

Los submódulos principales utilizan sus propias tablas que pocas veces se relacionan con las de los demás grupos; cada submódulo tiene su forma de almacenar los datos repitiendo continuamente información en sus diversas tablas de históricos.

4.3.4 Descripción Funcional del SIMIP

El SIMIP está basado en una arquitectura cliente-servidor el cual se conecta a una Base de Datos SQL Server a través de una conexión ADO y utiliza el proveedor de datos nativo SQLOLEDB. Está escrito usando tecnología Active Server Pages 3.0 (ASP) sobre lenguaje Vbscript. Del lado del cliente se utilizó JavaScript para la validación de los datos de entrada del usuario.

Un *script* ASP central (conectarBD.asp) se encarga de definir los parámetros de conexión que utilizarán todas las demás páginas .asp.

```
Set Conn = Server.CreateObject("ADODB.Connection")
Conn.Open("Provider=SQLOLEDB;driver=SQLOLEDB;server=172.30.74.141;database
=sec;uid=sa;pwd=PWD")
```

Cada una de las consultas o ingresos al sistema tiene el orden siguiente:

- Definición del Recordset.
  - `set RSDatos = server.CreateObject("ADODB.recordset")`
- Asignación de la consulta o *query* en una variable temporal.
  - `QueryDatos = "select Origen from tareaspendientes where num_control = 5"`
- Ejecución de la sentencia SQL.
  - `RSDatos.Open queryDatos,conn`

En muchas ocasiones se usa extensivamente el "select *" con un alto consumo de memoria y de procesador sobre todo si las tablas de la SEC son grandes debido a la gran cantidad de datos que almacenan.

Con este tipo de consultas, cada vez que se abre un objeto Recordset o un objeto Command se abre una nueva conexión. La creación implícita de objetos de conexión incrementa el tiempo de ejecución de programas y scripts en aproximadamente 3 veces.

En la programación se usaron indistintamente cursores dinámicos en toda la aplicación en vez de la utilización de cursores de "sólo avance", lo que disminuye el rendimiento de la misma.

En este sistema no se aplican las hojas de estilo (*.css) ni tampoco las hojas de estilo XSL por lo que el mantenimiento del diseño de las páginas sería una labor tediosa y tardada.

Dada la complejidad del sistema y con el objeto de centrarnos en el módulo de ejemplo de esta tesis, sólo se describirá el procedimiento de entrada que se tiene en la Ventanilla que a su vez contiene el Módulo de Validación de trámites.

4.4 Glosario de Términos

Autorización: Es la formalización o aceptación para poner en práctica alguna actividad o proyecto; o la aprobación final de una gestión o trámite con base a un marco normativo formalmente establecido.

Aviso de Movimiento de Personal (AMP): Documento que formaliza y define los términos de la contratación del personal así como los cambios o incidencias que ocurren durante la vida laboral de los trabajadores.

Bajas de personal: Movimientos que se generan por la separación del servicio de un trabajador en forma temporal o definitiva, de conformidad al marco normativo aplicable.

Cancelación-creación de plazas: Proceso de transferencias presupuestales que se realiza cuando a un trabajador se le asigna una nueva plaza a partir de otra formalmente autorizada; operándose de conformidad a los procedimientos y políticas formalmente establecidas. Esta operación se lleva a cabo en el área de Registro y Control de Plazas utilizando para ello los documentos de control internos conocidos como *cuadros de cancelación-creación.*

Enlace administrativo: Dirección encargada de elaborar y actualizar los manuales administrativos, modificar las estructuras orgánicas, reactivar plazas de mando, tramitar ante el ISSSTE los accidentes de trabajo, elaborar y actualizar la plantilla del personal, tramitar promociones del personal ante Subjefatura de Recursos Humanos de la SEC, coordinar las labores de seguridad e higiene y apoyar en actividades inherentes al área. En ocasiones ingresa directamente el trámite a análisis utilizando un memo.

Es también el medio que vincula con el Nivel Educativo para el propósito de aprobar un movimiento u otro asunto relacionado con el trámite del interesado.

Incidencia: En los avisos de movimiento de personal, es la especificación detallada de los motivos que puede tener un trámite para su mejor identificación.

Incompatibilidad: Incumplimiento de los criterios reguladores fijados para establecer la compatibilidad horaria de empleos y/o en distancia geográfica para que un trabajador ejerza simultáneamente dos o más cargos o funciones.

Matriz para la Atención de Movimientos de Personal: Documento normativo que contiene los tipos de movimientos de personal que puede generar un trabajador al brindar sus servicios dentro de la SEC; dicha matriz describe y define cada tipo trámite-motivo-incidencia, así como señala los requisitos para su autorización y la normatividad que los regula.

Movimiento de personal: Proceso que se genera al iniciar o modificarse la situación laboral de un trabajador, puede tratarse con base en una propuesta sindical, oficial o gestión personal.

Nivel Educativo: Coordina junto con Supervisión Escolar la realización de las cédulas de movimientos de personal. Se compone de varias áreas:

- Dirección General de Educación Popular (DGEP - 2311).- Compuesta a su vez por nivel preescolar, nivel primarias y nivel educación especial.
- Dirección General de Educación Media Superior y Superior (DGEMSyS - 2312).
- Dirección de Secundarias y Telesecundarias (2314).
- Universidad Pedagógica Veracruzana (UPV - 2355).

Plaza presupuestaria: Posición individual de trabajo que no puede ser ocupada por más de un empleado a la vez, además tiene una adscripción determinada. Integra un conjunto de labores y responsabilidades asignadas en forma permanente a un solo empleado.

Procedimiento: Sucesión cronológica de operaciones concatenadas entre sí, que se constituyen en una unidad de función cuando se realiza una actividad o tarea específica dentro de un ámbito predeterminado de aplicación. Todo procedimiento involucra actividades y tareas del personal, determinación de tiempos, de métodos de trabajo y de control para lograr el cabal, oportuno y eficiente desarrollo de las operaciones.

Supervisión escolar: Es una función ejercida por una persona llamada supervisor o inspector. Teóricamente la supervisión es una función del sistema educativo cuyo objetivo es cuidar, vigilar y apoyar el desarrollo de la organización escolar. Es el encargado de ingresar el trámite a Ventanilla utilizando para ello la cédula de movimientos de personal.

Trámite: Acción o gestión que se realiza, a petición del interesado o por decisión oficial, que modifica y/o documenta las percepciones o situaciones administrativas de un trabajador.

Unidad presupuestal: Clave presupuestaria que identifica y clasifica a las dependencias públicas.

Vacante: Cargo o plaza que se encuentra sin titular o sin quien lo(a) ejerza.

Zona escolar: Clasificación convencional administrativa con que se designa a una región donde se encuentran ubicados conjuntos de centros de trabajo o escuelas que son coordinados por un supervisor.

Fuentes:

- Documentos normativos del gobierno federal y estatal.
- Administración de personal y recursos humanos
- William B. Werther, Jr. – Keith Davis
- Información documental del Centro Latinoamericano de Administración para el Desarrollo (CLAD).
- Página en Internet de la Secretaría de Educación Pública: http://www.sep.gob.mx

SIGLAS Y ABREVIATURAS UTILIZADAS

DGEMSyS	Dirección General de Educación Media Superior y Superior.
DGEP	Dirección General de Educación Popular.
DGES	Dirección General de Educación Secundaria.
IPE	Instituto de Pensiones del Estado.
ISSSTE	Instituto de Seguridad Social al Servicio de los Trabajadores del Estado.
SEC	Secretaría de Educación y Cultura.
SEFIPLAN	Secretaría de Finanzas y Planeación.

5.1 Ventajas de XML sobre la técnica basada en ASPs-ADODB

5.1.1 Introducción

El futuro de la programación basada en XML está intrínsecamente relacionado por la funcionalidad que proveen los lenguajes estándares como C++, Java, Visual Basic (ASP), etc., los cuales a su vez se implementan sobre sistemas de manejo de datos (DBMS).

La información se almacena en un modelo relacional y por varias razones continuará almacenándose de esta forma, pero XML es una manera muy útil de comunicarse con otros sistemas. Los gestores de datos son el punto inicial y/o final de muchos flujos de datos y como tales deben tener el rol de procesar consultas y de transformar datos. SQL/XML es ahora una parte innovadora del estándar SQL que permite a los usuarios generar y consultar información en XML dentro del modelo relacional. Las formas no-relacionales de información están adquiriendo importancia creciente y las bases de datos pueden mejorarse mediante estructuras específicas de almacenamiento XML.

5.1.2 Estructura Jerárquica

El tipo de estructura jerárquica de XML permite una mejor representación de la información en diferentes aplicaciones Ej. temarios, planes de estudio; para nuestro caso, XML es una buena herramienta para modelar el encadenamiento requerido para programar el motor de validación de movimientos.

5.1.3 Facilidad de rastreo de acciones de un usuario (Analista de Trámites)

El SIMIP tiene incorporado un catálogo de páginas dentro de su base de datos para tener un mejor control sobre el acceso de un usuario a cada una de las páginas. Sin embargo, el mantenimiento a este catálogo era tedioso, ya que había que crear la página, agregarla al catálogo y después definir los permisos y alcances de la misma. Mediante XML, esto sería una labor sencilla ya que un procedimiento automatizado extraería la información en estructura XML de cada una de las páginas para incorporarla a la Base de Datos, teniéndola definida con todos los permisos casi al mismo tiempo de su creación.

Adicionalmente, se puede tener un programa central que sea el que despliegue cada una de las páginas con las siguientes ventajas:

- Cada página estaría plenamente identificada mediante un encabezado (header) en XML.

- Cada acción del usuario quedaría plenamente auditada al extraer los identificadores "id's" de las páginas accesadas del encabezado en XML y ser ingresadas a la base de datos, con su fecha y hora respectivas.
- Se tendría un mejor control sobre la sesión del usuario, número de acciones realizadas, etc.

Al almacenarse los documentos (comprobantes y cartas) generados por el sistema en formato HTML, éstos tienen los inconvenientes inherentes al formato empleado que limitan el alcance y funcionalidad del sistema. Mediante el uso de la nueva tecnología basada en XML, se pueden eliminar muchas de las carencias de HTML.

- XML separa la estructura y la presentación, cuidando siempre la estructura y la semántica, a la vez que no se limita a una cantidad determinada de estilos.

- XML dispone de medios para comprobar si la estructura de un documento es correcta y también tiene formas para validar que las etiquetas estén bien formadas.

- Las búsquedas con XML son concisas, directas, pudiéndose realizar búsquedas complejas de información sobre grandes volúmenes de datos, en comparación con la forma de manejo actual en la que se requeriría mucho tiempo para realizar una búsqueda utilizando pocos criterios.

- Los datos estructurados con XML son plenamente reutilizables, se pueden incorporar a una base de datos, se pueden procesar mediante SAX o DOM para crear otros documentos, etc.

En comparación con los lenguajes de programación tradicionales (entre ellos Vbscript-ASP), el lenguaje de aplicación de estilos XSLT incrementa enormemente la productividad de un programador: las tareas sencillas se programan rápidamente, y las más complicadas se resuelven mucho más rápido que si se hubiera utilizado Java, C/C++ o Visual Basic. Esto se comprueba por la cantidad de tiempo empleado en el diseño de comprobantes de movimientos (Facturas), hojas de antigüedad y hojas de servicios, entre otras.

5.2 Herramientas XML de explotación de la BD del personal

5.2.1 Introducción

Hasta el momento se han elaborado muchas herramientas las cuales se han incorporado como paquetes de clases en Java (como Xerces que contiene DOM y SAX), como componentes .DLL agregados al sistema Windows (MSXML) o como módulos de programación XML (PHP y Perl).

Las herramientas que se utilizarán en los ejemplos siguientes utilizan:

Transact-SQL de SQL Server 2000 para devolver los resultados de las consultas directamente a documentos con estructura XML.

MSXML y su módulo DOM para procesar documentos XML que según su jerarquía y estructura dentro del árbol va incorporando cada registro a la base de datos de docentes.

SAX y programación Java mediante los cuales se programa un ejemplo sencillo de motor de validación que muestra claramente cómo puede una aplicación trabajar directamente sobre documentos XML generados a partir de una consulta a una base de datos con la información estrictamente indispensable para la operación.

La tecnología actual y el tamaño de la información generada con las herramientas anteriores, hace posible manipularla en un dispositivo portátil PDA con muy pocos cambios, ya que gracias a la versión de Java para Palm J2ME (Java 2 Micro Edition) puede ejecutarse e implementarse de manera vertical.

5.3 Consultas automáticas en SQL Server usando XML

5.3.1 Introducción

La interoperabilidad es esencial en la transferencia de información en los diversos entornos operacionales, por tal motivo Microsoft reconoce que XML es clave en el proceso de intercambio de información.

Como adición a la nueva versión, SQL Server 2000 incorpora una nueva funcionalidad en algunas de sus sentencias Transact-SQL de tal forma que ahora devuelven los resultados directamente como documentos XML y facilitan también la inserción y manipulación de registros utilizando este lenguaje.

5.3.2 Exportación de datos de SQL Server a XML

Uso de SELECT ... FOR XML

El modo tradicional de utilizar la sentencia SELECT se modifica, ya que ahora puede devolver resultados en formato XML en lugar de un conjunto de registros. Esto se logra añadiendo la cláusula FOR XML en la instrucción SELECT. Haciendo referencia solamente a las adiciones que se han realizado en esta materia, tenemos la siguiente sintaxis:

SELECT [... FROM ... WHERE ...] [FOR { BROWSE | XML | RAW | AUTO | EXPLICIT} [, XMLDATA] [, ELEMENTS] [,BINARY base64]]

Como podemos observar, dentro de esta sentencia se encuentra una serie de especificaciones de modo: RAW, AUTO o EXPLICIT.

RAW

Esta es una de las formas más simples para obtener los resultados de tipo XML ya que cada columna del conjunto de registros de resultado (Recordset) es interpretada como un elemento XML identificado por ROW. Las columnas aparecen como atributos, siendo excluidas aquellas con valores nulos, con el nombre de la columna igual al atributo.

AUTO

Otra de las formas simples de obtener un resultado XML es mediante la utilización de la cláusula AUTO, la cual devuelve los datos como elementos XML. Cada elemento representa una tabla de la cláusula FROM y los atributos corresponden a la columna asociada a la sentencia SELECT. La diferencia con modo RAW es que no se utiliza el identificador común <row> sino que éste es sustituido por el nombre de la tabla.

EXPLICIT

Esta opción permite controlar la forma del documento XML que devuelve la consulta. Esto se realiza mediante la especificación de la estructura jerárquica en la consulta SELECT, indicando mediante una sintaxis predefinida la aparición de los componentes.

Para construir la estructura del árbol XML, se asocia la siguiente sintaxis a los campos de la consulta SQL:

```
ElementName!TagNumber!AttributeName!Directive
```

ElementName. Nombre del elemento al que se le aplican las directivas.
TagNumber. Etiqueta asociada a la consulta sobre la que se está trabajando.
AttributeName. Nombre del atributo o elemento, esto último si Directive es "element".
Directive. Determina cómo debe ser manejada la información. En nuestro caso lo utilizamos para especificar que sea tratada como elemento en vez de atributo.

Para desarrollar el ejemplo necesitamos de las siguientes instrucciones:

`sp_makewebtask` . Procedimiento almacenado que permite guardar los resultados de una consulta mediante la especificación de las siguientes variables:

`@outputfile`. Especifica el archivo XML de salida en donde se almacenará la información.

`@query`. Esta variable contiene la consulta fuente de donde se tomarán los datos de exportación.

`@templatefile`.- Esta variable contiene la plantilla con el esqueleto básico de un documento XML bien formado.

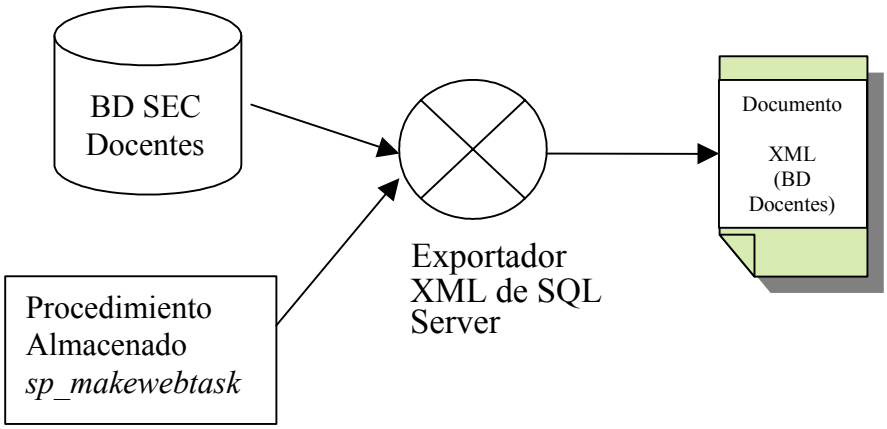


Fig. 5.1 *Procedimiento de exportación de datos usando el método EXPLICIT.*

Ejemplo de Aplicación: Exportación de datos de tablas de SQL Server a archivos XML.

El código se ejecuta desde el SQL Query Analyzer y consiste de tres secciones de código que servirán para la exportación de datos: la base de empleados, la base de ocupación de plazas y la base de plazas.

Exportación de la tabla empleados:

```
DBCC TRACEON(257)
EXEC sp_makewebtask
     @outputfile = 'c:\docs\tesis\Otros_doc\baseemp.xml',
     @query = 'SELECT 1 as tag,
NULL AS Parent,
E.num_per as [Empleado!1!num_per],
E.RFC AS [Empleado!1!RFC],
E.ApPaterno       as [Empleado!1!ApPaterno!element],
E.ApMaterno       as [Empleado!1!ApMaterno!element],
E.Nombre          as [Empleado!1!Nombre!element],
E.sexo            as [Empleado!1!sexo!element],
E.fechanacimiento as [Empleado!1!fechanacimiento!element],
E.NivelEstudios   as [Empleado!1!NivelEstudios!element],
E.nombreConyuge   as [Empleado!1!nombreConyuge!element],
E.notas           as [Empleado!1!notas!element]
From Empleados E
FOR XML EXPLICIT',
     @templatefile ='c:\docs\tesis\template.txt'
```

Exportación de la tabla ocupación:

```
DBCC TRACEON(257)
EXEC sp_makewebtask
      @outputfile = 'c:\docs\tesis\Otros_doc\baseocupacion.xml',
      @query = 'SELECT 1 as tag,
NULL AS Parent,
O.Unidad_presupuestal as [Ocupacion!1!Unidad_presupuestal],
O.Departamento        as [Ocupacion!1!Departamento],
O.Tabulador           as [Ocupacion!1!Tabulador],
O.Numero_plaza        as [Ocupacion!1!Numero_plaza],
O.empleado            as [Ocupacion!1!empleado!element],
O.tipoDeOcupacion     as [Ocupacion!1!tipoDeOcupacion!element],
O.estado              as [Ocupacion!1!estado!element],
O.licencia            as [Ocupacion!1!licencia!element],
O.fechaInicio         as [Ocupacion!1!fechaInicio!element],
O.fechaTermino        as [Ocupacion!1!fechaTermino!element],
O.fechaDeActualizacion as [Ocupacion!1!fechaDeActualizacion!element],
O.motivo              as [Ocupacion!1!motivo!element],
O.folio               as [Ocupacion!1!folio!element],
O.fechaMovimiento     as [Ocupacion!1!fechaMovimiento!element],
O.centroDeTrabajoEjercido as
[Ocupacion!1!centroDeTrabajoEjercido!element],
O.centroEstatal       as [Ocupacion!1!centroEstatal!element],
O.localidad           as [Ocupacion!1!localidad!element]
From Ocupacion_Plazas O
FOR XML EXPLICIT',
      @templatefile ='c:\docs\tesis\template.txt'
```

Exportación de la tabla de plazas:

```
DBCC TRACEON(257)
EXEC sp_makewebtask
      @outputfile = 'c:\docs\tesis\Otros_doc\baseplazas.xml',
      @query = 'SELECT 1 as tag,
NULL AS Parent,
P.Unidad_presupuestal as [Plaza!1!Unidad_presupuestal],
P.Departamento        as [Plaza!1!Departamento],
P.Tabulador           as [Plaza!1!Tabulador],
P.Numero_plaza        as [Plaza!1!Numero_plaza],
P.CentroDeTrabajo     as [Plaza!1!CentroDeTrabajo!element],
P.Tabulador1          as [Plaza!1!Tabulador1!element],
P.cve_ocupacion       as [Plaza!1!cve_ocupacion!element],
P.tipo                as [Plaza!1!tipo!element],
P.OrigenDeLaPlaza     as [Plaza!1!OrigenDeLaPlaza!element],
P.Status as [Plaza!1!Status!element],
P.FechaDisponibilidad as [Plaza!1!FechaDisponibilidad!element]
From Plazas P
FOR XML EXPLICIT',
      @templatefile ='c:\docs\tesis\template.txt'
```

5.3.3 Importación de datos existentes en archivos de datos XML a BD SQL Server.

Este método utiliza Script ActiveX DTS junto con MSXML 4.0 (la función DOM), el cual permite ejecutar el programa de importación dentro de SQL Server que realiza las funciones indicadas en la figura.

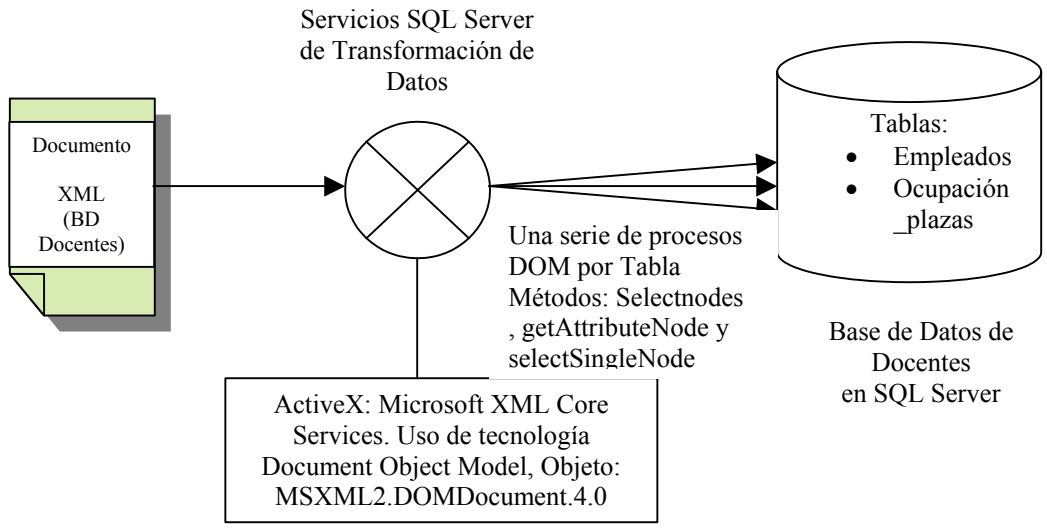


Figura 5.2 *Procedimiento de importación de datos utilizando DTS - MSXML.*

El programa utiliza como entrada el documento XML que contiene las tablas de la BD de docentes. Mediante un ciclo se van recorriendo los nodos del documento XML a la vez que se van insertando en la BD. Los métodos y procedimientos más importantes se explican a continuación.

Es necesario para utilizar el objeto crear una instancia del mismo (`objXMLDOM`).

```
Set objXMLDOM = CreateObject("MSXML2.DOMDocument.4.0")
```

La instancia se configura especificando la ubicación del archivo XML en strCurFileName.

```
objXMLDOM.load strCurFileName
```

Para manejar cada elemento que representa una tabla en la Base de Datos es necesario crear uno nuevo con esa especificación utilizando el método selectNodes.

```
Set objNodes = objXMLDOM.selectNodes("/BD_Docentes/Empleados/Empleado")
```

En este caso el nodo `Empleados/Empleado` se sustituye por el elemento que indica cada registro y que corresponde a su tabla: `Ocupacion_plazas/Ocupación` y `Plazas/Plaza`.

El objeto que contiene el conjunto de datos (recordset) a insertar en la BD de SQL Server se crea de la manera tradicional.

```
Set objADORS = CreateObject("ADODB.Recordset")
objADORS.Open "SELECT * FROM Empleados WHERE 1 = 2", objADOCnn,
adOpenKeyset, adLockOptimistic
```

Cada uno de los nodos del objeto es barrido para procesarse individualmente.

```
For Each objNodeItem In objNodes
```

En el caso de insertar algún atributo en un campo de la BD se realiza mediante el método `getAttributeNode`.

```
objADORS.fields("<campo BD>") = objNodeItem.getAttributeNode("<valor del
atributo>").Value
```

Para el caso de tener que insertar un elemento en la BD, se hace utilizando el método `selectSingleNode`.

```
objADORS.fields("ApPaterno") = objNodeItem.selectSingleNode("<valor del
elemento>" ).nodeTypedValue
```

Cerrando los objetos y actualizando el Recordset cada vez que se utiliza se puede garantizar la liberación de recursos (memoria) del sistema.

El ejemplo completo se enlista a continuación:

```
'************************************************************************
'  Visual Basic ActiveX Script
'************************************************************************
CONST strFilesPath = "C:\docs\tesis\Otros_doc\basetotalv2.xml"
CONST adOpenKeyset = 1
CONST adLockOptimistic = 3

Function Main()
      Dim objFSO
      Dim objFolder
      Dim objFilesColl
      Dim iFilesCount
      Dim objFile

      Dim objXMLDOM
      Dim objNodes
      Dim objNodeItem

      Dim objADORS
      Dim objADOCnn

      Dim strCurFileName

      '*** Crea e inicializa la conexión ADO
      Set objADOCnn = CreateObject("ADODB.Connection")
      objADOCnn.Open
"PROVIDER=SQLOLEDB;SERVER=.;UID=sa;PWD=dinasoft;DATABASE=tesis_xml;"

      '*** Crea el Objeto DOM MSXML 4.0 y lo inicializa
      Set objXMLDOM = CreateObject("MSXML2.DOMDocument.4.0")
      objXMLDOM.async = False
      objXMLDOM.validateOnParse = False
```

```
            strCurFileName = strFilesPath

            '*** Carga el archivo XML file
            'No se realiza manejo de errores
            objXMLDOM.load strCurFileName

'*** Comienza Tabla de Empleado
Set objNodes = objXMLDOM.selectNodes("/BD_Docentes/Empleados/Empleado")

'Crea y abre el Recordset
  Set objADORS = CreateObject("ADODB.Recordset")
  objADORS.Open  "SELECT  *  FROM  Empleados  WHERE  1  =  2",  objADOCnn,
adOpenKeyset, adLockOptimistic

'Añade registros
For Each objNodeItem In objNodes
  With objADORS
.AddNew

.fields("num_per")= objNodeItem.getAttributeNode("num_per").Value
.fields("RFC") = objNodeItem.getAttributeNode("RFC").Value
.fields("ApPaterno")=objNodeItem.selectSingleNode("ApPaterno").nodeTypedValue
.fields("ApMaterno")=objNodeItem.selectSingleNode("ApMaterno").nodeTypedValue
.fields("Nombre")=objNodeItem.selectSingleNode("Nombre").nodeTypedValue
.fields("sexo") = objNodeItem.selectSingleNode("sexo").nodeTypedValue
.fields("fechanacimiento")=objNodeItem.selectSingleNode
   ("fechanacimiento").nodeTypedValue
.fields("NivelEstudios")=objNodeItem.selectSingleNode
   ("NivelEstudios").nodeTypedValue
.fields("nombreConyuge")=objNodeItem.selectSingleNode
   ("nombreConyuge").nodeTypedValue
.fields("notas")=objNodeItem.selectSingleNode("notas").nodeTypedValue

.Update
  End With
Next

Set objNodes = nothing
objADORS.Close

'*** Comienza la Tabla de Plazas
  Set objNodes = objXMLDOM.selectNodes("/BD_Docentes/Plazas/Plaza")
'*** Abre el Recordset
  objADORS.Open "SELECT * FROM Plazas WHERE 1 = 2", objADOCnn, adOpenKeyset,
adLockOptimistic

'*** Añade registros
For Each objNodeItem In objNodes
  With objADORS
.AddNew
.fields("Unidad_presupuestal") = objNodeItem.getAttributeNode
  ("Unidad_presupuestal").Value
.fields("Departamento") = objNodeItem.getAttributeNode("Departamento").Value
.fields("Tabulador") = objNodeItem.getAttributeNode("Tabulador").Value
.fields("Numero_plaza") = objNodeItem.getAttributeNode("Numero_plaza").Value
```

```
.fields("CentroDeTrabajo") =objNodeItem.selectSingleNode
   ("CentroDeTrabajo").nodeTypedValue
.fields("Tabulador1") = objNodeItem.selectSingleNode
   ("Tabulador1").nodeTypedValue
.fields("cve_ocupacion") = objNodeItem.selectSingleNode
   ("cve_ocupacion").nodeTypedValue
.fields("Tipo") = objNodeItem.selectSingleNode("Tipo").nodeTypedValue
.fields("OrigenDeLaPlaza") = objNodeItem.selectSingleNode
   ("OrigenDeLaPlaza").nodeTypedValue
.fields("Status") = objNodeItem.selectSingleNode
   ("Status").nodeTypedValue
.fields("FechaDisponibilidad") = objNodeItem.selectSingleNode
   ("FechaDisponibilidad").nodeTypedValue
  End With
Next

Set objNodes = nothing
objADORS.Close
'*** Comienza la tabla de Ocupacion_plazas
Set                                objNodes                                    =
objXMLDOM.selectNodes("/BD_Docentes/Ocupacion_plazas/Ocupacion")
'*** Abre el Recordset
  objADORS.Open "SELECT * FROM Ocupacion_plazas WHERE 1 = 2", objADOCnn,
adOpenKeyset, adLockOptimistic

'*** Añade registros
For Each objNodeItem In objNodes
  With objADORS
.AddNew
.fields("Unidad_presupuestal") = objNodeItem.getAttributeNode
   ("Unidad_presupuestal").Value
.fields("Departamento") = objNodeItem.getAttributeNode("Departamento").Value
.fields("Tabulador") = objNodeItem.getAttributeNode("Tabulador").Value
.fields("Numero_plaza") = objNodeItem.getAttributeNode("Numero_plaza").Value
.fields("empleado") = objNodeItem.selectSingleNode("empleado").nodeTypedValue
.fields("tipoDeOcupacion") = objNodeItem.selectSingleNode
   ("tipoDeOcupacion").nodeTypedValue
.fields("estado") = objNodeItem.selectSingleNode
   ("estado").nodeTypedValue
.fields("licencia") = objNodeItem.selectSingleNode("licencia").nodeTypedValue
.fields("fechaInicio") = objNodeItem.selectSingleNode
   ("fechaInicio").nodeTypedValue
.fields("fechaTermino") = objNodeItem.selectSingleNode
   ("fechaTermino").nodeTypedValue
.fields("fechaDeActualizacion") = objNodeItem.selectSingleNode
   ("fechaDeActualizacion").nodeTypedValue
.fields("motivo") = objNodeItem.selectSingleNode
   ("motivo").nodeTypedValue
.fields("folio") = objNodeItem.selectSingleNode("folio").nodeTypedValue
.fields("fechaMovimiento") = objNodeItem.selectSingleNode
   ("fechaMovimiento").nodeTypedValue
.fields("centroDeTrabajoEjercido") = objNodeItem.selectSingleNode
   ("centroDeTrabajoEjercido").nodeTypedValue
.fields("centroEstatal") = objNodeItem.selectSingleNode
   ("centroEstatal").nodeTypedValue
.fields("localidad") = objNodeItem.selectSingleNode
   ("localidad").nodeTypedValue
```

```
  End With
Next

      objADORS.Close
      objADOCnn.Close

      '*** Parte final del script
      Set objADORS = Nothing
      Set objADOCnn = Nothing
      Set objXMLDOM = Nothing
      Set objFSO = Nothing

      Main = DTSTaskExecResult_Success
End Function
```

5.4 Explotación dinámica de contenidos XML mediante interfases API.

5.4.1 Aplicación basada en DOM

DOM es una API o interfaz de programación para desarrollo de aplicaciones que provee un modelo estándar de programación para trabajar con documentos XML bien formados y con documentos XML válidos.

El DOM fue diseñado para:

- Proveer un mecanismo para construir, navegar o actualizar y transformar los contenidos de los documentos XML de forma programática.
- Establecer un núcleo de interfases de programación neutrales al lenguaje, que puedan ser utilizados para satisfacer las necesidades más comunes para los desarrolladores que generan XML bien formado y flujos de datos HTML de entrada y salida.

La implementación DOM en Microsoft XML Core Services (MSXML) permite cargar o crear un documento, detectar errores si existen, acceder y manipular la información y estructuras dentro del documento; también permite guardar el documento de nuevo a un archivo XML, en caso de ser necesario.

5.4.1.1 DOM y MSXML

La implementación DOM es sólo una parte del parser MSXML. El siguiente diagrama muestra las tareas necesarias para procesar un documento XML ("parsing") y para presentar la información en una aplicación o script.

El enfoque DOM crea un árbol de objetos que es administrado por el analizador o "parser" de MSXML. Esto permite a los desarrolladores aprovechar la lógica incorporada a MSXML para el manejo de contenido XML, en lugar de crear una propia.

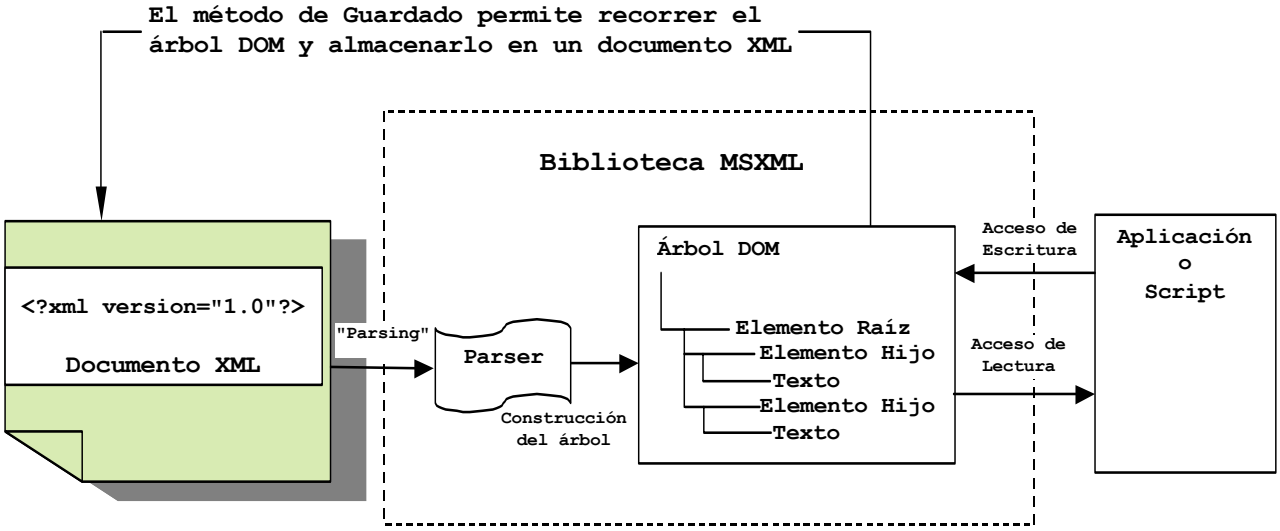


Fig. 5.3 *Funcionamiento de DOM con MSXML.*

5.4.1.2 Utilización de DOM

El DOM permite a las aplicaciones trabajar con estructuras de documentos XML en vez de flujos de texto. Las aplicaciones y los scripts pueden entonces manipular estas estructuras sin conocer los detalles de la sintaxis XML. De esta manera aprovechan las facilidades que incluye el API DOM de MSXML.

El DOM utiliza dos conceptos clave: Una jerarquía de tipo árbol y nodos que representan el contenido del documento y su estructura. La jerarquía se compone de estos nodos, los cuales pueden contener o están contenidos en otros nodos. Para los desarrolladores, esto significa que gran parte del trabajo en el procesamiento de XML requiere de la navegación de esta estructura de árbol para encontrar o modificar la información que contiene. El trabajo con XML requiere conceptualizar la información en términos de contenedores anidados, y garantizar que la información se pone en o se recupera desde el contenedor correcto.

El DOM trata a los nodos como objetos genéricos, haciendo posible crear un script que cargue un documento y luego recorra todos los nodos, indicando qué encuentra en cada nodo del árbol.

Con DOM es posible construir y modificar los nuevos documentos ya que la elaboración de los mismos puede realizarse sin tener nada, o a partir de otros documentos; DOM permite cortar y copiar partes de un documento y colocarlas en donde se requiera, incluso en otro documento.

NOTA: En el ejemplo anterior se utilizó MSXML DOM y SQL-Server para ingresar los datos desde el archivo XML a la Base de Datos.

5.4.2 Aplicación basada en SAX sobre Java y PERL

Además del ejemplo basado en MSXML DOM, presentamos un ejemplo que utiliza aplicaciones basadas en el concepto SAX y DOM; SAX incluido en Xerces implementa clases Java en su programación. También utiliza PERL como intermediario entre el programa Java y el servidor web que a su vez despliega la página HTML creada en Java.

SAX es una colección de interfases y clases, las cuales son herramientas muy útiles en el procesamiento de documentos XML. SAX está orientado a eventos; los eventos pueden ser el hecho de encontrar una etiqueta de inicio de un documento o el comienzo de un documento. Cuando el procesador XML encuentra un determinado evento se invoca a un método capaz de manejarlo.

El nombre SAX procede de la expresión en inglés "Simple API for XML", la cual quiere decir "API sencilla para XML". Este tipo de interfases facilitan el manejo de XML; por esto SAX junto con DOM son las API para XML más utilizadas.

SAX no genera estructuras internas, no almacena en memoria los eventos que ocurren y por tanto no se puede volver hacia atrás y es necesario asociar un código de programación que los maneje en el momento en que ocurren.

El SAX versión 2 se puede obtener de manera gratuita de http://www.megginson.com/SAX

5.4.2.1 Descripción de la Aplicación

La necesidad de movilidad, hace que sea muy útil la incorporación de extractos de las Bases de Datos en los dispositivos móviles tales como los PDAs, Palm, PocketPC, etc. Una aplicación muy importante para este tipo servicios administrativos es la de poder proporcionar y almacenar información de manera casi inmediata aún en las zonas más alejadas en las que se carece de servicios tales como el Internet.

Esta aplicación permite verificar movimientos fuera de línea sobre archivos XML, los cuales a su vez sirven de alimentación a la BD al momento de sincronizar la información.

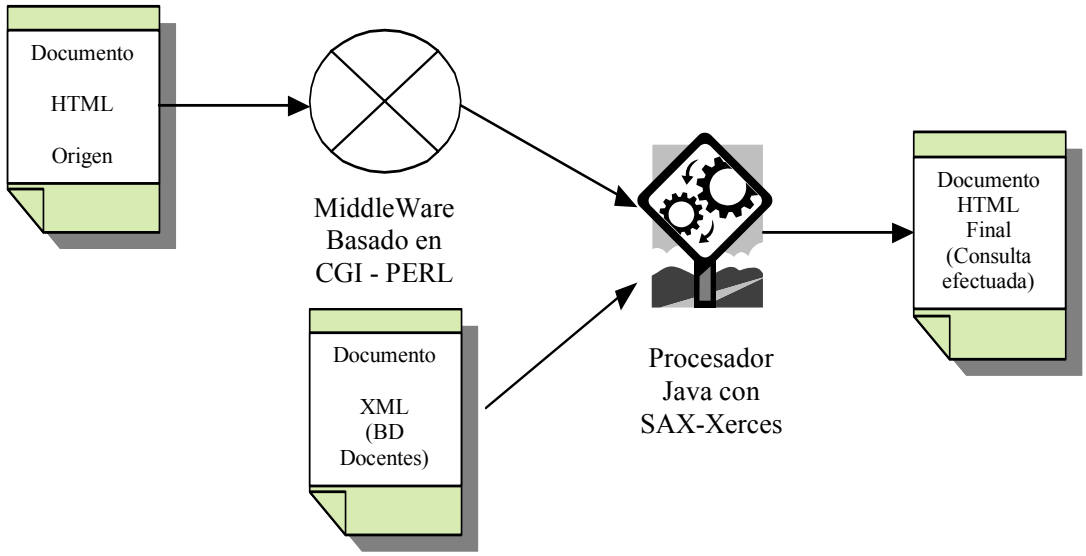


Figura 5.4 *Realización de movimientos sobre plazas y validación de las mismas.*

Dom vs SAX.

SAX es útil en aplicaciones en las que se cuenta las veces que ocurre un elemento o que deseen obtener el valor de un elemento o de un atributo en concreto, en búsquedas de información y en general en aquellas en las que no sea necesario conocer, navegar o modificar la estructura de un documento.

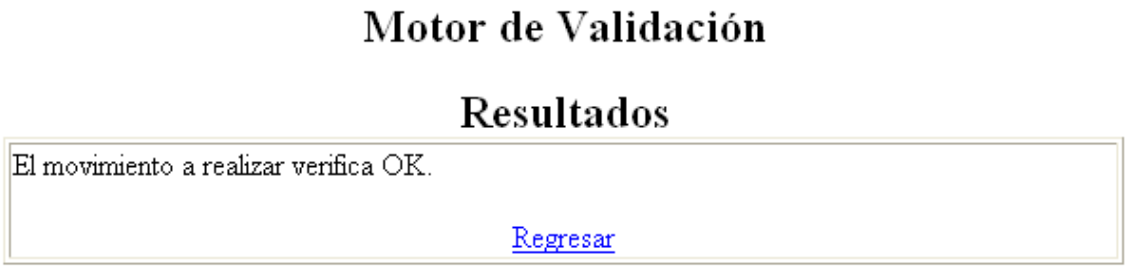
El ejemplo trabaja tomando como datos de entrada el nombre del empleado al que se va a aplicar el movimiento, el movimiento en sí y la plaza a afectar. Aunque ya en un caso real se tendría que tomar otra información como las vigencias, el catálogo de movimientos descrito en la gaceta, el nivel educativo al que pertenece, su número de personal, etc., esta información es suficiente para mostrar la funcionalidad de SAX sobre XML.

Figura 5.5 *Pantalla de ingreso de datos al sistema.*

El criterio sobre el cual se basa el sistema para decidir si el movimiento aplica o no es que una plaza registrada como cancelada no puede volverse a cancelar y por el contrario una plaza dada de alta no puede registrarse como activa nuevamente. En un caso real de operación se podrían incluir muchas más validaciones como la validación de vigencias que consiste en que la vigencia de la plaza coincida con la señalada en la entrada, la validación de horas del docente para que no rebase las horas marcadas en la normatividad, etc.

En este caso el empleado CRUZ EULOGIO GUADALUPE puede darse de alta porque la plaza 2311058109807218 se encuentra vacante, es decir, hubo una suspensión o cancelación anterior (Figura 5.6, arriba).

Por otro lado, si este empleado quisiera darse de baja por jubilación en esa misma plaza, no podría hacerlo porque los datos aún no se actualizan en la Base de Datos y el primer filtro marca que la plaza está registrada como cancelada (Figura 5.6, abajo).



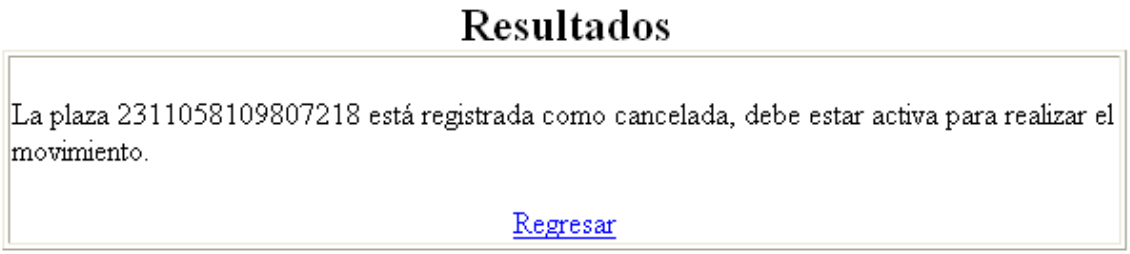


Figura 5.6 Verificación del movimiento cuando éste procede y cuando no lo hace, respectivamente.

5.4.2.2 Descripción del código.

El sistema se conforma de 5 clases principales que se conectan entre sí para lograr la funcionalidad requerida por la aplicación:

1. DespliegaEmpleado.class
2. MotorValidacion.class
3. ManejaEventos.class
4. cEventosXML.class
5. cEventosXMLBuscaPlazas.class

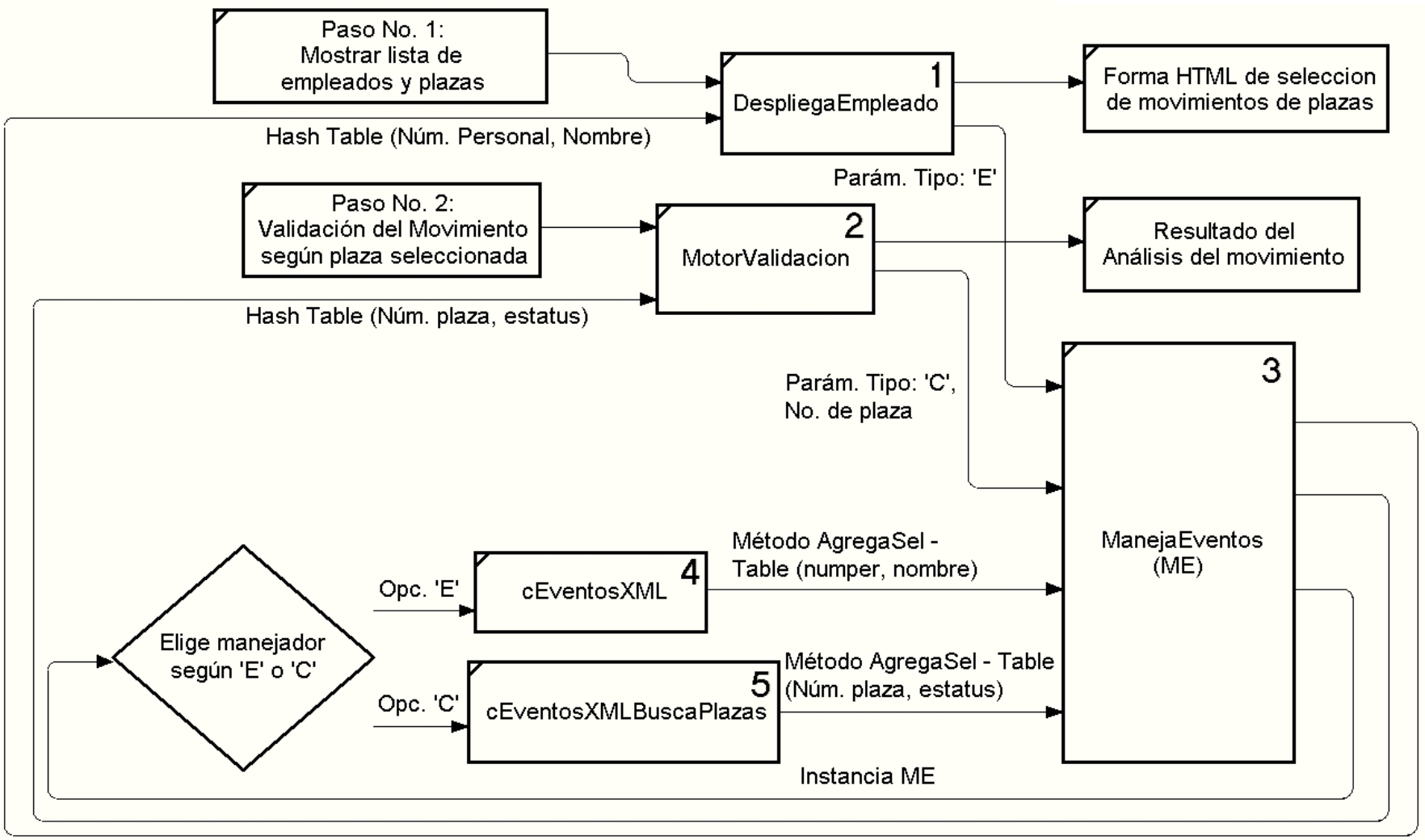


Figura 5.7 *Modelo funcional del motor de validación en Java.*

Clase DespliegaEmpleado (1)

La función principal de la clase es la de mostrar los elementos que nos permiten hacer una elección y así proporcionarle al motor de validación todos los elementos para decidir si un movimiento es válido o no.

Las estructuras más importantes en Java son:

```
public DespliegaEmpleado (){ ... }
```

Este constructor arma el esqueleto HTML para la inclusión de los elementos.

```
public void ConstruyeForma(){ … }
```

Este método manda llamar a la clase `ManejaEventos` y genera los elementos HTML de entrada `"<select>"` de la siguiente forma:

```
ManejaEventos ManejaSel = new ManejaEventos("E");
...
tablasel = ManejaSel.RegresaHash();
```

El parámetro mandado "E" especifica a la clase que debe buscar en la estructura XML como empleado usando como manejador de eventos a la clase `cEventosXML`. `Tablasel` es entonces recorrida para generar las etiquetas `"<option>"` dentro de la etiqueta `"<select>"`.

El manejo que se hace para las plazas es similar, con la diferencia de que se llama a `ManejaEventos` con "P" y se devuelve un vector `listaplazas`.

La clase completa se muestra a continuación:

DespliegaEmpleado.class

```
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;
import org.xml.sax.Attributes;
import org.xml.sax.ContentHandler;

import java.util.*;
import org.xml.sax.InputSource;

public class DespliegaEmpleado {

            String cdFolder = null;
            Hashtable tablasel;
            Vector listaplazas;

            static public void main (String [] argv){
                  DespliegaEmpleado        miempleado        =        new
DespliegaEmpleado();
            }

            public DespliegaEmpleado (){
                  System.out.println   ("<!DOCTYPE   HTML   PUBLIC   \"-
//W3C//DTD HTML 4.01 Transitional//EN\">");
                  System.out.println ("<html>");
```

```
                System.out.println ("<head>");
                System.out.println ("<title>Buscador Plazas</title>");
                System.out.println ("<meta http-equiv=\"Content-Type\"
content=\"text/html; charset=iso-8859-1\">");
                System.out.println ("</head>");
                System.out.println ("<body>");
                System.out.println ("<p align=\"center\"><strong><font
size=\"+2\">Motor de Validación</font></strong></p>");
                System.out.println      ("<form     name=\"formabusq\"
method=\"POST\" action=\"http://localhost/cgi-bin/Tesis1.pl\">");
                System.out.println ("<table width=\"60%\" border=\"1\"
align=\"center\"><tr><td>");
                ConstruyeForma();
                System.out.println ("</td></tr></table>");
                System.out.println ("</form>");
                System.out.println ("</body></html>");
                }

  //Construye la estructura de la forma
  public void ConstruyeForma(){
            ManejaEventos ManejaSel = new ManejaEventos("E");
            StringBuffer buf = new StringBuffer();
            tablasel = ManejaSel.RegresaHash();
            System.out.println("<table    width=\"100%\"    border=\"0\"
align=\"center\">");
            System.out.println("<tr><td>Nombre del empleado: </td>");
            System.out.println("<td><select name=\"empleado\">");
            Enumeration enumkey = tablasel.keys();
            for     (Enumeration     enum     =     tablasel.elements();
enum.hasMoreElements();) {
              buf.append( "<option value=\"" + enumkey.nextElement() +
"\">" + enum.nextElement() + "</option>").append( '\n' );
              }
            System.out.println( buf.toString() );
            System.out.println("</select>");
            System.out.println("</td></tr>");
            System.out.println("<tr><td>");
            System.out.println("Movimiento:");
            System.out.println("</td><td>");
            System.out.println("<select name=\"movimiento\">");
            System.out.println("<option    value=\"0101\">Alta    nuevo
ingreso</option>");
            System.out.println("<option     value=\"0201\">Baja     por
jubilación</option>");
            System.out.println("</select>");
            System.out.println("</td></tr>");

            ManejaEventos ManejaSelPlaza = new ManejaEventos("P");
            listaplazas = ManejaSelPlaza.RegresaVector();
            System.out.println("<tr><td>");
            System.out.println("Plaza:");
            System.out.println("</td><td>");
            System.out.println("<select name=\"plaza\">");
            for(int i=0; i < listaplazas.size(); ++i)
              System.out.println("<option>" + listaplazas.elementAt(i) +
"</option>");
            System.out.println("</select>");
```

```
            System.out.println("</td></tr>");
            System.out.println("<tr><td colspan=2 align=center>");
            System.out.println("<br><input                type=\"submit\"
value=\"Ingresar\">");
            System.out.println("</td></tr>");
            System.out.println("</table>");
            }
 }

 class cEventosXMLPlazas extends DefaultHandler{
            String Unidad_presupuestal = null;
            String Departamento = null;
            String Tabulador = null;
            String Numero_plaza = null;
            String cadenaleida = null;
        ManejaEventos MiDespliegue = null;
            int i = 0;

      public cEventosXMLPlazas (ManejaEventos Despliega){
                  super();
                  MiDespliegue = Despliega;
      }
        public void startDocument () throws SAXException {}
        public void endDocument   () throws SAXException {}
            public void startElement (String URL, String nombrelocal,
String nombrecal, Attributes atts) throws SAXException{
              if(nombrelocal.equals("NumPlaza")){
                  for (i = 0; i < atts.getLength(); i++){
                      String nomatrib = atts.getLocalName(i);
                        String valatrib = atts.getValue(i);
                        if(nomatrib.equals("Unidad_presupuestal")){
                          Unidad_presupuestal = valatrib;
                        }
                        if(nomatrib.equals("Departamento")){
                          Departamento = valatrib;
                        }
                        if(nomatrib.equals("Tabulador")){
                          Tabulador = valatrib;
                        }
                        if(nomatrib.equals("Numero_plaza")){
                          Numero_plaza = valatrib;
                          MiDespliegue.AgregaSelVec(Unidad_presupuestal
+ Departamento + Tabulador + Numero_plaza);
                        }
                    }
              } //Fin de if de Numero de Plaza
         }

         public void endElement (String URL, String nombrelocal, String
nombrecal) throws SAXException{
            }

        public void characters(char[] ch, int inicio, int longit) throws
SAXException{
          cadenaleida = new String(ch, inicio, longit);
        }
}
```

Clase MotorValidacion (2)

La función de esta otra clase es desplegar los resultados de la validación del movimiento, toma como parámetros el empleado, el tipo de movimiento y la plaza, tal y como lo indica la función main() de la clase:

```
MotorValidacion mimotor = new MotorValidacion(empleado, movimiento, plaza);
```

Las estructuras más importantes en Java son:

```
public MotorValidacion (String emp, String mov, String pza){ ... }
```

Este es el constructor de la clase y despliega el esqueleto HTML para mostrar los resultados. Asimismo manda llamar a la clase que mediante la búsqueda del parámetro del número de plaza devuelve su estatus:

```
ManejaEventos ManejaSel = new ManejaEventos("C", plaza);
tablasel = ManejaSel.RegresaHash();
```

Los parámetros mandados "C" y plaza especifican que se debe buscar en el árbol XML los estatus ligados a esa plaza utilizando el manejador de eventos `cEventosXMLBuscaPlazas`.

Tablasel se envía al método RealizaMotor:

```
public void RealizaMotor (Hashtable tabla, String mov){ ... }
```

Mediante la tabla y el tipo de asignación (alta o baja) el sistema determina si un movimiento es válido o no según el siguiente criterio:

- Si el movimiento es 0101 (Alta por nuevo Ingreso), la plaza debe estar suspendida o cancelada para poder ocuparse, por lo que el estatus debe ser alguno distinto de "Creada".
- Por el contrario, si el movimiento es 0201 (Baja), la plaza debe estar activa para poder ocuparse, por lo que el estatus debe ser alguno distinto de "Cancelada".

Es posible robustecer el motor de validación incluyendo en el mismo un mayor número de validaciones y así ponerlo en funcionamiento en un ambiente de trabajo real.

La clase completa se muestra a continuación:

MotorValidacion.class

```
import java.util.*;
import org.xml.sax.InputSource;
import java.lang.System;

import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;
import org.xml.sax.Attributes;
import org.xml.sax.ContentHandler;

public class MotorValidacion {
```

```
            String empleado = null;
            String movimiento = null;
            String plaza = null;

            Hashtable tablasel;
            Vector listaplazas;

              String Status = null;
              String Plaza = null;
              boolean error = false;

            static public void main (String [] argv){
            String empleado = argv[0].trim();
            String movimiento = argv[1].trim();
            String plaza = argv[2].trim();
            MotorValidacion mimotor = new MotorValidacion(empleado,
movimiento, plaza);
            }

            public MotorValidacion (String emp, String mov, String pza){
            empleado = emp;
            movimiento = mov;
            plaza = pza;
            ManejaEventos ManejaSel = new ManejaEventos("C", plaza);
            tablasel = ManejaSel.RegresaHash();
                  System.out.println   ("<!DOCTYPE   HTML   PUBLIC   \"-
//W3C//DTD HTML 4.01 Transitional//EN\">");
                  System.out.println ("<html>");
                  System.out.println ("<head>");
                  System.out.println ("<title>Buscador Plazas</title>");
                  System.out.println ("<meta http-equiv=\"Content-Type\"
content=\"text/html; charset=iso-8859-1\">");
                  System.out.println ("</head>");
                  System.out.println ("<body>");
                  System.out.println ("<p align=\"center\"><strong><font
size=\"+2\">Motor de Validación</font></strong></p>");
                  System.out.println                                  ("<div
align=\"center\"><strong><font
size=\"+2\">Resultados</font></strong></div>");
                  System.out.println ("<table width=\"60%\" border=\"1\"
align=\"center\"><tr><td>");
                  RealizaMotor(tablasel, movimiento);
                  System.out.println                ("<br><br><center><a
href=\"javascript:history.back(-1);\">Regresar</a></center>");
                  System.out.println ("</td></tr></table>");
                  System.out.println ("</body></html>");
                  }

            public void RealizaMotor (Hashtable tabla, String mov){
              Enumeration enumkey = tabla.keys();
              for (Enumeration enum = tabla.elements();
enum.hasMoreElements();) {

                Status = enum.nextElement() + "";
                  Plaza = enumkey.nextElement() + "";
                if(mov.equals("0101")){
```

```
                    if(Status.equals("Creada")){
                      System.out.println("<br>La plaza " + Plaza + "
está registrada como activa, debe estar cancelada o vacante para
realizar el movimiento.");
                        error |= true;
                    }
                  }else if(mov.equals("0201")){
                    if(Status.equals("Cancelada")){
                      System.out.println("<br>La plaza " + Plaza + "
está registrada como cancelada, debe estar activa para realizar el
movimiento.");
                        error |= true;
                    }
                  }
              } //Fin del for-tabla
              if(!error){
                System.out.println("El movimiento a realizar verifica
OK.");
              }
            }

}
```

Clase ManejaEventos (3)

Esta clase es la encargada de dirigir el gestor a los distintos manejadores de eventos, tomando en cuenta a los parámetros con los que se llamó la clase. Es también manejador de los objetos que almacenan los datos que se utilizan para consultar los resultados del "parsing" sobre la estructura de datos en XML. Las estructuras clave se muestran a continuación:

```
XMLReader Analizador = new org.apache.xerces.parsers.SAXParser();
```

Aquí se construye una instancia del analizador que es de tipo XMLReader el cual es el que va a correr el analizador en base al documento XML y al gestor especificados.

```
if (Tipo.equals("E")) {
  cEventosXML Gestor = new cEventosXML(this);
  Analizador.setContentHandler(Gestor);
}
```

Según el tipo se crea una nueva instancia del manejador que se encargará de procesar el documento XML, inmediatamente se le pasa la instancia "Gestor" como manejador al Analizador. En este caso utilizamos el tipo 'E', pero el programa decide también sobre el 'P' y el 'C'.

```
cdFicheroXML = "file:///C:/Inetpub/wwwroot/PrgTesis1/ejemp_1.xml";
InputSource ArchXML = new InputSource(cdFicheroXML);
Analizador.parse(ArchXML);
```

Se define el archivo (con la ruta completa) que contiene los datos exportados de SQL Server en estructura XML. El archivo es pasado a través de ArchXML, que es un objeto de tipo InputSource.

En este momento al Analizador sólo le falta activar la herramienta que procesa ArchXML. Es entonces cuando el parser de SAX marca los errores de mala formación del documento XML.

```
public void AgregaSel(String numpersonal, String nombcompleto){
  table.put(numpersonal, nombcompleto);
}

public void AgregaSelVec(String Nombre){
  listaemp.addElement(Nombre);
}

public Vector RegresaVector();
public Hashtable RegresaHash();
```

Estos elementos hacen la labor de ingresar datos a la tabla (tipo Hashtable) y al vector (tipo Vector). Asimismo, existen métodos encargados de recuperar estas estructuras llamadas RegresaHash() y RegresaVector() que regresan los valores de la tabla y el vector respectivamente.

El ejemplo completo se muestra a continuación:

ManejaEventos.class

```
import org.xml.sax.XMLReader;
import org.xml.sax.ContentHandler;
import org.xml.sax.helpers.ParserFactory;
import org.xml.sax.helpers.DefaultHandler;
import java.util.*;
import org.xml.sax.InputSource;
import org.xml.sax.SAXException;

public class ManejaEventos {
                  Hashtable table;
                  String cdFicheroXML;
                  String miplaza;
                  Vector listaemp = new Vector();
                  public ManejaEventos(String Tipo){
                  try{
                    table = new Hashtable();
                    XMLReader Analizador = new
org.apache.xerces.parsers.SAXParser();
                    if (Tipo.equals("E")){
                      cEventosXML Gestor = new cEventosXML(this);
                        Analizador.setContentHandler(Gestor);
                    }else if (Tipo.equals("P")){
                      cEventosXMLPlazas       Gestor       =       new
cEventosXMLPlazas(this);
                        Analizador.setContentHandler(Gestor);
                    }

                    cdFicheroXML =
"file:///C:/Inetpub/wwwroot/PrgTesis1/ejemp_1.xml";
                    InputSource ArchXML = new InputSource(cdFicheroXML);
                    Analizador.parse(ArchXML);
                  } catch (Exception e) {
                        System.out.println(e);
```

```
                        }
                    }

                    public ManejaEventos(String Tipo, String plaza){
                    try{
                    miplaza = plaza;
                      table = new Hashtable();
                      XMLReader Analizador = new
org.apache.xerces.parsers.SAXParser();
                      if (Tipo.equals("C")){
                        cEventosXMLBuscaPlazas Gestor = new
cEventosXMLBuscaPlazas(this,miplaza);
                          Analizador.setContentHandler(Gestor);
                      }

                      cdFicheroXML =
"file:///C:/Inetpub/wwwroot/PrgTesis1/ejemp_1.xml";
                      InputSource ArchXML = new InputSource(cdFicheroXML);
                      Analizador.parse(ArchXML);
                    } catch (Exception e) {
                          System.out.println(e);
                          }
                    }

                    public void AgregaSel(String numpersonal, String
nombcompleto){
                    //Coloca un nuevo elemento al HashTable
                      table.put(numpersonal, nombcompleto);
                    }

                    public Hashtable RegresaHash(){
                      return table;
                    }
                    //Maneja Vectores
                    public void AgregaSelVec(String Nombre){
                    //Coloca un nuevo elemento al Vector
                      listaemp.addElement(Nombre);
                    }
                    public Vector RegresaVector(){
                      return listaemp;
                    }
  } //Fin de clase ManejaEventos
```

Clase cEventosXML (4)

La función principal de esta clase es proveer el código para extraer la información de las etiquetas con el nombre de "empleado". La información que se extrae es específicamente la que necesitamos.

Las líneas de código más importantes son:

```
public cEventosXML (ManejaEventos Despliega){
            super();
            MiDespliegue = Despliega;
}
```

Se manda a llamar al constructor de la clase, en donde se importa el objeto, en este caso nombrado MiDespliegue.

Los dos métodos usados que invoca SAX son startElement y endElement.

El empleado, al tener como atributo el número de personal, el valor pasa a través de startElement y su valor almacenado en numper.

```
public void startElement (String URL, String nombrelocal, String nombrecal, Attributes atts)
```

En cambio, los apellidos están como elementos hijos y es necesario manejarlos a través del elemento de cierre o endElement.

```
public void endElement (String URL, String nombrelocal, String nombrecal)
```

Este método trabaja en conjunto con el método characters, el cual almacena una cadena siempre que la encuentra.

```
public void characters(char[] ch, int inicio, int longit) throws SAXException{
          cadenaleida = new String(ch, inicio, longit);
        }
```

La cadena es almacenada en la variable `cadenaleida`. Esta variable es utilizada en el método de cierre `endElement` para almacenar el apellido paterno, el apellido materno y el nombre, al final, cuando detecta la etiqueta de cierre "nombre", manda llamar al método que almacena los datos en la tabla Hash de nombre `AgregaSel`.

```
if(nombrelocal.equals("Nombre")){
             //Llamar al asignador externo
   MiDespliegue.AgregaSel(numper, ApPaterno + " " + ApMaterno + " " + cadenaleida);
             } //Fin de if EMPLEADO
```

El código completo se muestra a continuación:

cEventosXML.class

```
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;
import org.xml.sax.ContentHandler;
import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
```

```
public class cEventosXML extends DefaultHandler{
            String numper = null;
            String ApPaterno = null;
            String ApMaterno = null;
            String cadenaleida = null;
        ManejaEventos MiDespliegue = null;
            int i = 0;

      public cEventosXML (ManejaEventos Despliega){
                  super();
                  MiDespliegue = Despliega;
      }
        public void startDocument () throws SAXException {}
        public void endDocument   () throws SAXException {}
            public void startElement (String URL, String nombrelocal,
String nombrecal, Attributes atts) throws SAXException{
              if(nombrelocal.equals("Empleado")){
                  for (i = 0; i < atts.getLength(); i++){
                      //System.out.println("Pasa!");
                      String nomatrib = atts.getLocalName(i);
                        String valatrib = atts.getValue(i);
                        if(nomatrib.equals("num_per")){
                          //MiDespliegue.AsignaId(valatrib);
                          numper = valatrib;
                          //System.out.println("Pasa! " + numper);
                        }
                    }
              } //Fin de if de Empleado

              if(nombrelocal.equals("ApMaterno")){
              //LLamar al asignador externo

              }
         }
          public void endElement (String URL, String nombrelocal, String
nombrecal) throws SAXException{
              if(nombrelocal.equals("ApPaterno")){
                //Obtención del valor del contenido
                ApPaterno = cadenaleida;
              }else
              if(nombrelocal.equals("ApMaterno")){
                ApMaterno = cadenaleida;
              }else
              if(nombrelocal.equals("Nombre")){
              //LLamar al asignador externo
                 MiDespliegue.AgregaSel(numper,  ApPaterno  +  "  "  +
ApMaterno + " " + cadenaleida);
              } //Fin de if EMPLEADO
            }

        public void characters(char[] ch, int inicio, int longit) throws
SAXException{
          cadenaleida = new String(ch, inicio, longit);
        }
}
```

Clase cEventosXMLBuscaPlazas (5)

Esta clase procesa las etiquetas con el nombre de NumPlaza, para ello recibe como parámetros la instancia de la clase ManejaEventos ME y la plaza a buscar. Dado que la plaza se compone de Unidad presupuestal, Departamento, Tabulador y Numero_plaza, se asignan las variables correspondientes.

```
Unidad_presupuestal = Plaza.substring(0,4);
Departamento = Plaza.substring(4,7);
Tabulador = Plaza.substring(7,11);
Numero_plaza = Plaza.substring(11,16);

MiDespliegue = Despliega;
```

Asimismo se declara la referencia `MiDespliegue` para tener acceso a las funciones de `ManejaEventos` (ME).

```
public void startElement (String URL, String nombrelocal, String
nombrecal, Attributes atts) throws SAXException{
```

Al igual que en el ejemplo anterior, como se trata de atributos la verificación de existencia de una clase se realiza a través del método `startElement()`.

```
if(nombrelocal.equals("NumPlaza")){
if(nomatrib.equals("Unidad_presupuestal")){ ... }
if(nomatrib.equals("Departamento")){ ... }
if(nomatrib.equals("Tabulador")){ ... }
if(nomatrib.equals("Numero_plaza")){ ... }
```

En caso se tratarse de la etiqueta `NumPlaza` se verifica cada uno de los componentes de la plaza que pudiera coincidir con la plaza que se está buscando. En caso de hacerlo, `cumple` es verdadero.

```
public void endElement (String URL, String nombrelocal, String
nombrecal) throws SAXException{
```

Dado que necesitamos el estatus de la plaza que estamos buscando, le asociamos el método `AgregaSel` que agrega los datos a la tabla Hash, requiriendo para esto los parámetros `Plaza` y `Estatus`. Esto lo realiza en caso de que `cumple` sea verdadero.

```
MiDespliegue.AgregaSel(Plaza,Status);
```

La variable `Status` obtiene su valor de `cadenaleida` que a su vez se asigna en el método `characters`, al igual que en el caso anterior.

El código completo de la clase se muestra a continuación:

cEventosXMLBuscaPlazas.class

```
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;
import org.xml.sax.ContentHandler;
import org.xml.sax.Attributes;
import org.xml.sax.SAXException;

public class cEventosXMLBuscaPlazas extends DefaultHandler{
            String Unidad_presupuestal = null;
```

```
            String Departamento = null;
            String Tabulador = null;
            String Numero_plaza = null;
            String Status = null;
            String cadenaleida = null;
            String Plaza = null;
          ManejaEventos MiDespliegue = null;
            int i = 0;
            boolean cumple = true;

      public cEventosXMLBuscaPlazas (ManejaEventos Despliega, String
Plz){
                  super();
                  Plaza =     Plz;
                  MiDespliegue = Despliega;
                  Unidad_presupuestal = Plaza.substring(0,4);
                  Departamento = Plaza.substring(4,7);
                  Tabulador = Plaza.substring(7,11);
                  Numero_plaza = Plaza.substring(11,16);

      }
        public void startDocument () throws SAXException {}
        public void endDocument   () throws SAXException {}
            public void startElement (String URL, String nombrelocal,
String nombrecal, Attributes atts) throws SAXException{
              if(nombrelocal.equals("NumPlaza")){
                  for (i = 0; i < atts.getLength(); i++){
                      String nomatrib = atts.getLocalName(i);
                        String valatrib = atts.getValue(i);
                        if(nomatrib.equals("Unidad_presupuestal")){
                        cumple = true;

                           if(valatrib.equals(Unidad_presupuestal)){
                             cumple &= true;
                           }else{
                             cumple &= false;
                           }

                        }
                        if(nomatrib.equals("Departamento")){
                           if(valatrib.equals(Departamento)){
                             cumple &= true;
                           }else{
                             cumple &= false;
                           }


                        }
                        if(nomatrib.equals("Tabulador")){
                           if(valatrib.equals(Tabulador)){
                             cumple &= true;
                           }else{
                             cumple &= false;
                           }
                        }
                        if(nomatrib.equals("Numero_plaza")){
                           if(valatrib.equals(Numero_plaza)){
```

```
                              cumple &= true;
                            }else{
                              cumple &= false;
                            }

                           MiDespliegue.AgregaSelVec(Unidad_presupuestal
+ Departamento + Tabulador + Numero_plaza);
                         }

                    }
               } //Fin de if de Número de Plaza

          }

          public void endElement (String URL, String nombrelocal, String
nombrecal) throws SAXException{
                           if(nombrelocal.equals("Status")){
                           Status = cadenaleida;
                                if(cumple){
                                  MiDespliegue.AgregaSel(Plaza,Status);
                                }
                           }
             }

         public void characters(char[] ch, int inicio, int longit) throws
SAXException{
           cadenaleida = new String(ch, inicio, longit);
         }
}
```

6.1 Definición de Criterios de Evaluación

Beneficios vs. Costos de la estructura XML

Las ventajas y desventajas y desventajas del uso de XML descritas a continuación se han tratado de alguna manera en los anteriores capítulos; aquí se describen de una manera más concisa, pretendiendo enfatizar el por qué XML nos puede ayudar en desarrollo de nuevos proyectos de manejo de información, considerando de manera integral en la decisión los contrapesos tales como el rendimiento, complejidad de la estructura, etc.

6.1.1 El formato de texto estructurado de XML

El hecho de que XML sea un formato de documento estructurado permite ir más allá de la información a intercambiarse; la definición de *meta*-datos permite especificar el significado y estructura de la información que será intercambiada entre sistemas.

La gran diferencia respecto a la mayoría de los archivos de texto, es que éstos representan la información de manera simple, sin *meta*-datos, o llegan a incluirlos, se trata de una estructura plana de un solo nivel. Los formatos delimitados por comas o tabuladores, como los que se manejan en archivos de MS-Excel no tienen el nivel de estructura ni la legibilidad con la que cuenta un documento XML.

Otra de las ventajas al utilizar XML es que al utilizar DTDs o esquemas los editores XML proporcionan edición estructurada "gratuita". Como desarrollador esto ahorra muchos problemas ya que evita que nuestro programa genere una excepción en una línea determinada, ya que el editor solamente permite introducir una estructura XML válida.

Pueden utilizarse diversas herramientas estandar de manejo de texto disponibles sin costo adicional en sistemas UNIX, para búsqueda, ordenamiento, filtrado y reestructuración de la información contenida dentro de los documentos XML

6.1.1.1 Desventajas

A pesar de todo el valor agregado que conlleva el representar la información y la *meta*-información de una manera estructurada, algunos proyectos simple y sencillamente no requieren de la complejidad que involucra XML. En estos casos, los archivos de texto simples realizan el trabajo de una manera más eficiente. Por ejemplo, un archivo de configuración que requiere una pequeña lista de unos cuantos comandos y sus valores, no requiere de un formato de archivo multi-nivel e incorporado con *meta*-datos para que sea interpretado.

Otro ejemplo práctico y que tiene que ver más con la tecnología en Internet es el concerniente a la realización de un trabajo pequeño que involucra unas cuantas páginas Web. Sería poco práctico detenerse a pensar en los pasos para la creación de documentos XML tales como la especificación de requisitos y el diseño (definición de lo que hay que marcar, adición de organización y estructura, la decisión de definirlos como elementos o como atributos) para crear una página Web que contenga unas cuantas fotos familiares.

Otra desventaja de XML es que a pesar de que cuenta con herramientas de validación, carece de editores que permitan detectar todos los errores al realizar el *parsing* a lo largo del documento. Muchos de los editores XML son incapaces de seguir más allá del primer error de sintaxis.

6.1.2 Disponibilidad y costo de la tecnología de procesamiento de XML

XML está dispnible, muy difundida y es econónica. Uno de los aspectos principales de los formatos de archivo alternativos y de los lenguajes de bases de datos es que las herramientas de procesamiento son específicas, propietarias o muy costosas. Mientras una herramienta se difunde, ésta es usualmente muy específica para un formato propietario. Una de las más grandes fortalezas de XML es que las herramientas de procesamiento han llegado a ser relativamente difundidas y poco costosas e incluso gratuitas.

XML marca la estructura de un documento que comparte varias de las especificaciones con SGML y HTML; esto ha contribuido a que se hayan construido muchos *parsers* hasta la fecha. Varios de estos *parsers* se han interconstruido con los navegadores instalados en los clientes y en agentes instalados del lado del servidor.

Además, el DOM (Document Object Model) Modelo de Objeto de Documento, ha sido creado por el W3C como un modelo general de cómo los *parsers* y los procesadores deberían interactuar y procesar documentos XML para representarlos como un árbol de datos relacionados. DOM crea a partir de esto un método genérico y universal para procesar documentos XML. Las aplicaciones que requieren procesamiento XML pueden acceder a toda esta gama de herramientas y así añadir procesamiento simbólico o *parsing* sin complicaciones.

6.1.2.1 Desventajas

Aún cuando las herramientas de procesamiento de documentos XML son económicas, la implementación de todas estas herramientas no es un proceso fácil. El procesamiento de documentos XML no termina al momento de realizar un *parsing* sobre un archivo XML; si se quiere realmente tener una secuencia en el tratamiento de la información, es necesario añadir rutinas de procesamiento que le den utilidad propia a los datos recopilados.

El DOM tiene características que podrían causar lentitud en una aplicación de negocios, entre ellas están el alto costo de ejecución, manejo ineficiente de errores y otros problemas de implementación dado su nivel de desarrollo tecnológico aún no ha alcanzado un nivel de plena madurez.

6.1.3 Legibilidad de XML por humanos

Otro beneficio de XML es que es legible y puede ser escrito por humanos, muy diferente a los archivos generados por máquinas los cuales se encuentran en un formato que solamente ellas leen e interpretan. En un principio XML está pensado para una comunicación de máquina a máquina y puede crearse mediante herramientas visuales que no necesitan la edición sobre el código, sin embargo, la experiencia con el HTML nos ha mostrado que en numerosas ocasiones un desarrollador tiene que "meterse" al documento y hacer cambios. Por esta razón XML está escrito en texto plano y usa elementos que representan palabras o frases que contienen algún significado.

Un buen diseño de XML debe considerar la legibilidad. La legibilidad para humanos no sólo facilita las tareas de diagnóstico y depuración, sino que también acelera el tiempo de implantación. Sin la información bloqueada en un formato binario propietario, los desarrolladores pueden verificar fácilmente que sus procesos tienen un código preciso.

6.1.3.1 Desventajas

Se infiriere del argumento anterior que si el código XML está diseñado para un consumo entre máquinas solamente, la legibilidad entre humanos queda fuera de contexto. En este caso, dada la circunstancia de que los humanos nunca necesiten realmente meterle mano u observar el documento XML implica que el que esté formateada en texto y legible por humanos pueda resultar en un detrimento en el rendimiento dada la saturación y el tamaño del archivo por sí solo.

Existe código XML que tiene más características orientadas hacia la máquina. A esto se le puede sumar que los humanos pueden crear todavía código XML ilegible, por ejemplo, muchos desarrolladores codifican nombres que tienen significado para su aplicación pero que no son comprensibles para otros programadores. Un ejemplo es el elemento "<sdsiti44>" el cual no es nada significativo para nadie excepto el desarrollador y la aplicación. En principio habría que definir quién es el lector objetivo: los desarrolladores o la gente de negocios. ¡No sólo porque XML puede ser legible por humanos significa que en todos los casos lo será!

6.1.4 Flexibilidad de XML

XML es flexible y con él pueden definirse otros lenguajes La extensibilidad de XML, que es la capacidad del lenguaje de usarse para definir nuevos vocabularios y *meta*-información, puede aplicarse para describir cualquier cantidad de documentos que en conjunto forman un lenguaje; por supuesto que esto lo hace en colaboración con sus DTDs y esquemas.

. Los archivos de texto y esquemas de Bases de Datos relacionales están muy ligados a la información que representan y nada más. Si se requiere añadir más información a un archivo de texto o una RDBMS, es necesario pensar en el trabajo que conllevaría esto. En comparación, los archivos XML y especialmente aquellos que usan un "modelo abierto"

pueden extenderse fácilmente si se le agregan elementos o atributos adicionales. Las clases de documentos completas pueden definirse incorporando al documento el nuevo DTD o esquema.

En la primera versión de XML, el medio principal para especificar la validez del documento era a través de la Definición de Tipo de Documento (DTD), la cual es una especificación que utiliza una sintaxis basada en el formato utilizado para los documentos SGML. Los DTDs usan una sintaxis inadecuada que sólo permite una definición simplista de los contenidos de los documentos.

Recientemente ha habido una serie de propuestas para una especificación de validez XML que ha mejorado notablemente con respecto a la anterior. Estas propuestas forman lo que es la recomendación de W3C para *esquemas* y han permitido definir los esquemas de datos utilizando la sintaxis XML, especificar el tipo ya sea simple o complejo, soporte para espacios de nombres (*namespaces*) y capacidades de herencia.

XML puede entonces intercambiarse a través del Internet, y ser una base para una gran variedad de otros lenguajes, estándares y formatos de datos.

6.1.4.1 Desventajas

El hecho de que XML posea la habilidad para definir otros lenguajes puede significar desorden. Acordar una DTD o un esquema común no es suficiente aún dentro de una comunidad de usuarios pequeña y bien definida.

Con respecto a la extensibilidad, XML tiene algunos puntos desfavorables. En particular, los usuarios no pueden añadir nombres de etiquetas a un documento XML si se controla por un DTD en el que el usuario no tiene permiso o capacidad para modificarlo. Es ese sentido, se encuentra "amarrado" a archivo de texto o formato RDBMS que tiene un esquema fijo o predeterminado. El proceso para especificar el esquema que define el DTD puede estar controlado por una organización de estándares, ser controlado por personal de TI interno, o no estar disponible para modificación "casual".

Otro punto en contra con respecto a la extensibilidad es que los archivos XML no pueden "agregarse" como simples archivos de texto o bases de datos. Tomando el hecho que estructuralmente XML requiere un par de etiquetas de inicio y cierre, las adiciones o inserciones de XML deberían realizarse dentro del documento. Esto hace que XML no sea muy conveniente para aplicaciones que requieren la creación de archivos cuyos datos se agregan uno tras otro, como en las bitácoras de sistemas. A pesar de esto, sí son posibles los archivos de bitácoras que implementan el XML realizando ingeniería cuidadosa y un cierto "aplanamiento" de la estructura XML. Sin embargo, el hacer la estructura de XML más plana realmente borra algunos de los beneficios de la naturaleza estructural del lenguaje. Por supuesto, uno debe escoger siempre el lenguaje más apropiado para su propósito y como tal XML no es la mejor solución para aplicaciones de formateo de archivos.

6.1.5 Independencia de datos de XML

XML separa el contenido del procesamiento. Se ha definido como característica primordial de XML la representación de la información y los *meta*-datos de esa información, pero no se especifica una manera particular de cómo debe procesarse la información ni se proveen restricciones para los mecanismos que manejan la información. Esto contrasta con los archivos de texto y bases de datos que requieren explícitamente de acceder a los documentos de una manera específica. Más aún, los archivos por sí mismos definen cómo la información debe procesarse y qué requerimientos del sistema se deben considerar para que los documentos tengan sentido. En contraste, los documentos XML codifican la información y su *meta*-información sin especificar cómo se va a procesar y a desplegar dicha información.

Esta capacidad de XML para separar su proceso de su contenido es de tipo "prueba a futuro" o de tipo "unión holgada" dependiendo a cual sector del mercado y espectro de desarrollo se dirija.

La prueba a futuro significa que ningún cambio futuro en la capa de intercambio de datos debe afectar a la capa de programación y viceversa.

Los sistemas de unión holgada permiten un intercambio de información en donde una parte no necesita conocer los detalles de cómo la otra parte planea procesar la información. Esto permite cambios en las capas de proceso, presentación o datos sin que se afecten las otras capas.

Esta característica es muy benéfica para aquellos que buscan un lenguaje para comunicación entre plataformas y entre dispositivos. La información contenida en el XML por supuesto que debe procesarse, pero no se especifican ni el método ni los medios para procesar esta información en el documento XML.

El Lenguaje extensible de estilos (XSL) y su componente de transformación XSLT especifica cómo puede transformarse el documento XML para desplegarse en diferentes dispositivos o transformarse en formas alternativas de representación XML. Esto significa que un archivo XML en particular puede transformarse a través de múltiples archivos XSL para desplegarse en el Web, en un teléfono celular o en formato para archivar y almacenar o para impresión.

Los documentos XML, a pesar de las instrucciones de proceso pueden intercambiarse entre sistemas tan diferentes como los basados en Microsoft Windows NT y los mainframes de IBM. En nuestro caso la aplicación SIMIP puede incorporar funciones que "entiendan" los formatos XML y así poder subir la información a un servidor principal.

XML es un muy buen EAI (Enterprise Applications Integrator) integrador de aplicaciones empresariales, que se puede utilizar para permitir a los sistemas comunicarse entre ellos.

6.1.5.1 Desventajas

XML ofrece un formato de datos "puro" que permite a los usuarios separar la información de su forma de ser procesada y desplegada. Sin embargo no es lo único, pueden utilizarse cualquier otro tipo de representaciones puras de datos como los archivos delimitados por comas.

Mientras que XML provee de un mecanismo para separar la información del contenido a procesar, la experiencia nos indica que es responsabilidad del programador humano asegurarse de que esta separación ocurre realmente. Es posible agregar elementos a un documento XML que indiquen los requerimientos a procesar y las restricciones del documento, impidiendo de esta manera la capacidad de procesamiento entre plataformas. De hecho, el potencial para crear documentos XML propietarios tiene ocupada a la comunidad de código abierto (open source). Para muchos programadores es poco natural abstraer la información completamente de sus requerimientos de proceso.

6.1.6 Costo de los sistemas XML

Comparado con otros estándares de comercio electrónico y de intercambio de datos, XML ofrece ahorros significativos en costo y tiempo de desarrollo que hacen de una implementación XML una buena opción para organizaciones pequeñas y medianas.

Existe en el medio una gran cantidad de componentes para sistemas de intercambio de documentos y comercio electrónico: herramientas de creación de documentos, componentes de procesamiento, verificadores de validez, mapeo de datos, integración con aplicaciones, acceso a centrales de comunicaciones, seguridad y otros componentes del rompecabezas comercial. XML simplifica en gran medida estos pasos y en ocasiones los elimina.

La validación integrada, los analizadores (*parsers*) y herramientas de procesamiento de bajo costo, el mapeo basado en XSL y el uso del Internet mantienen la cadena del comercio electrónico a un bajo costo. Se pueden encontrar herramientas generales de XML que proveen soluciones de manera flexible y con bajo costo.

Las herramientas EDI (Electronic Data Interchange) son específicas a un dominio de conocimiento y experiencia que viene asociado con un precio y costo inherentes. XML hace uso de la tecnología que ha estado en uso en la comunidad de código abierto por varios años.

6.1.6.1 Desventajas

Mientras que los costos de procesamiento y de intercambio de documentos se reducen mediante el uso de XML, no se eliminan todos los costos y algunos de hecho se incrementan. Para comenzar, es importante resaltar que los sistemas deben habilitarse para comunicarse con XML antes de percibir estos beneficios. ¿Cuántos sistemas están involucrados en el proceso comercial? ¿Cuántos de estos sistemas deben ser capaces de entender XML? ¿Qué costo tiene modificar estos sistemas?

La modificación de los sistemas involucrados en el proceso no es necesaria, podrían decir algunos, dado que las puertas de enlace y traductores pueden construir esa transformación de datos desde XML al formato de comunicación nativo. Sin embargo, esto no elimina completamente el costo de la integración XML, sino que lo cambia desde el sistema a integrarse al servicio de integración o puerta de enlace.

Los sistemas de puerta de enlace, serán una implementación a la medida y requerirá el tiempo de un desarrollador, costos de software adicionales, o ambos. Por lo que la integración con sistemas en operación no es una tarea "gratis" por todos los significados, pero se mitiga por los costos que implica que los sistemas que no son XML sean de pronto compatibles con el mismo.

6.1.7 Intercambio de datos

Un esquema o DTD acordados facilitan el intercambio de datos entre documentos. La implementación de un estándar XML es sencilla. Recordando, un estándar está conformado de un conjunto de elementos, atributos, estructuras, semánticas, procesos y vocabulario surgidos de común acuerdo y con los cuales se pueden intercambiar los documentos. Este conjunto puede fácilmente representarse mediante DTDs o esquemas.

Mediante el acuerdo en estos documentos, puede nacer un estándar. El apego a un estándar es esencial para establecer una forma de trabajo grupal organizada. Cuando las personas o las organizaciones comienzan a añadir sus propios elementos, usan un lenguaje distinto, o usan el esquema no para lo que fue propuesto, todo comienza a caerse. En la práctica, los estándares se adhieren apropiadamente y XML provee de una tecnología adecuada para especificar estándares y asegurar que los documentos creados son válidos.

Los estándares bien definidos tienen la noción de extensibilidad consigo mismos. Tales estándares permiten la definición de campos de información adicionales, la expansión por las organizaciones que lo implementan, y otras necesidades de “particularización”. Esto se logra usualmente proveyendo a las áreas con el estándar para “plug-ins” y el uso de “árboles recursivos” que permiten a la especificación expandirse.

6.1.7.1 Desventajas

El intercambio electrónico de datos (EDI) ha mostrado que aún si una especificación es lo más completa posible, las implementaciones por individuos pueden variar tremendamente. Como tal, se requieren acuerdos adicionales y especificaciones entre las compañías que usan un estándar. Estos parámetros cambian constantemente y reflejan las necesidades de las compañías para representar apropiadamente el uso que hacen con respecto a las tecnologías de intercambio de información.

Aún cuando los DTDs no se modifican, su vocabulario y su semántica es en varias ocasiones incomprendida. Por ejemplo, un elemento denominado “AA” en un documento XML que haga referencia a un DTD en particular puede tener una interpretación o definición muy distinta del mismo elemento “AA” en otro documento XML que haga

referencia a un DTD distinto. En el primer caso, el elemento "AA" se puede referir a una oferta para venta, mientras que el otro puede referirse a una oferta para compra. La confusión resultante podría ser tal que un sistema emisor podría interpretar el DTD como una oferta para venta, mientras que el sistema receptor lo procesa como oferta para compra y por lo tanto ¡envía una factura!

Los temas relacionados con el manejo de versiones y la compatibilidad son asuntos de importancia con los DTDs compartidos. Si un DTD es el "propietario" por naturaleza, y creado por una única corporación u organización, entonces todos los grupos que utilizan el DTD tienen la responsabilidad de mantenerlo al día con las actualizaciones en el formato. Este es especialmente el caso si un DTD se liga a un producto. La compañía tiene la habilidad para cambiar los archivos de DTDs en cada versión y los usuarios tendrán la encomienda de mantener un formato que cambia constantemente.

Para que XML llegue a ser un suceso ampliamente difundido, las compañías y las organizaciones de todo tipo tendrán que adoptar estándares internacionales de manera consistente. Los compradores deben insistir en productos que se adhieran a estos estándares. El soporte de XML por sí mismo no es suficiente; los usuarios se decepcionarán cuando descubran que los DTDs propietarios se interponen a una integración del sistema.

6.1.8 Consumo de recursos máquina de XML

Las buenas características de XML tienen consigo una contraparte en el consumo de memoria, procesador y ancho de banda. XML toma una gran cantidad de espacio para representar información que pudiera modelarse de manera similar utilizando un formato binario o un formato de archivo de texto más simple. La razón de esto es simple: es el precio que pagamos porque sea un código legible al programador, de plataforma neutra, separado de los procesos, mejorado con *meta*-datos, estructurado y validado.

Esta diferencia de espacio es algo que no debe despreciarse. Los documentos XML pueden ser de 3 a 20 veces más grande que una representación binaria o un archivo de texto comparable. Los efectos de este espacio no deben subestimarse. Es posible que 1 Gigabyte de información contenida en una Base de Datos pueda resultar en 20 Gigabytes de información codificada en XML. Esta información requiere almacenarse y transmitirse a través de la red – Hechos que deberían alegrar a quienes fabrican a los procesadores, discos duros y equipos de redes.

No hay que olvidar que las computadoras necesitan procesar esta información. Los documentos XML extensos requieren cargarse en memoria antes de procesarse, y algunos documentos XML pueden extenderse en tamaño hasta varios Gigabytes de espacio. Esto puede resultar en un procesamiento intermitente, proceso recurrente de documentos, si no es que cargas de trabajo pesadas para el sistema.

Por si esto fuera poco, gran parte de la pila de protocolos requieren un trabajo intensivo para que trabajen como deben. Por ejemplo, SOAP, el cual es una plataforma de mensajes y comunicación entre sistemas para usarse en llamadas a procedimientos remotos (RPC por sus siglas en inglés), ambos entre y dentro de sistemas de servidores de datos. La

intercomunicación que ocurre entre estos procesos puede ocasionar que el rendimiento del sistema sea pobre dado que XML es después de todo un protocolo basado en texto que está siendo utilizado para realizar llamadas RPCs entre sistemas. El uso de XML en esta manera transaccional y en tiempo real puede imponer más requerimientos en el sistema tales como análisis (*parsing*) y procesamiento que el mismo sistema puede manejar.

Sumado a esto, un problema que tienen varios *parsers* XML actuales, es que éstos leen el documento XML completo hacia la memoria antes de procesarlo. Esta práctica puede ser desastrosa para documentos XML muy grandes. XML es un lenguaje de datos el cual es complicado procesar al momento de hacer el análisis o *parsing* del mismo. Muy frecuentemente se incrementa la complejidad del código porque XML es más difícil de analizar que un formato de datos más simple como lo son los campos delimitados por comas o tabuladores.

Todos estos problemas pueden **contraatacarse** de varias maneras. La primera de ellas es evitar hacer uso de XML en donde sea. En donde decida utilizarse, uno debe evaluar la forma en que está siendo utilizado. ¿Se está utilizando el XML como un formato de mensajería o almacenamiento? ¿Necesita utilizarse en tiempo real, o se encuentra en modo asíncrono o por lotes? Cada uno de estos cuestionamientos tiene una solución que ataca perfectamente a cada uno de los puntos.

Para aquellos que utilizan el XML en modo por lotes que no es en tiempo real, seamos sinceros, los costos de ancho de banda y de almacenamiento en disco son cada vez más económicos. No hay duda de que XML nunca hubiera funcionado hace 20 años. La única razón por la que lo estamos considerando aún en la actualidad es porque cuesta poco almacenar e intercambiar esta información. Esto será más insignificante a medida que el espacio en disco y el ancho de banda lleguen hasta algunos centavos por gigabyte.

En esas situaciones en que el procesamiento en tiempo real y el poder de procesamiento son asuntos prioritarios, existen un conjunto de tecnologías que compactan el XML en un formato binario para ser transmitido a través del cable, y entonces se descompacta antes de procesarse. La especificación HTTP soporta compresión a nivel de transporte utilizando los estándares de compresión tales como gzip, y aunque poca gente lo implementa, pueden alcanzarse tazas de compresión de hasta 90% utilizando gzip en datos XML. Para aquellos que buscan transportar XML a través de otros protocolos, existen otras tecnologías parecidas que pueden ofrecer tazas de compresión similares. Después de todo, XML está formado por varios caracteres de menor que y espacios en blanco.

La compresión soluciona los asuntos relacionados con la transferencia de datos y el ancho de banda, pero no los requerimientos de procesamiento. Estos pormenores se solucionan a través de tecnologías que solamente cargan una porción del documento XML en memoria y entonces lo analizan de forma selectiva para obtener la información que requieren. Entonces un documento XML de 1GB puede simplificarse a los elementos mínimos necesarios para una transacción en particular.

Tal como ocurre con cualquier tecnología, existen caídas en el rendimiento, y uno de los asuntos a considerar para XML es el espacio. Sin embargo, no existe razón para descartar

una tecnología solamente porque impone más requerimientos de espacio en disco, ancho de banda y poder de procesamiento.

6.2 Uso de los criterios para evaluar los resultados obtenidos

El estándar XML le ha dado un orden a los medios existentes de transmisión de información y viene a complementar las tecnologías existentes.

La herramienta que se ha utilizado como API –Xerces— tiene varias funciones automáticas que permiten *verificar* si un *documento* está *bien formado* para poder empezar con el procesamiento de la estructura XML. Aunque la herramienta hace el *parsing* del documento XML, cuando se encuentra algún *elemento* debe realizarse una *programación específica para procesarlo*, tomando en cuenta por ejemplo, el orden de los elementos (si se busca en elementos hijos), si se va a buscar en el nombre del elemento o en el contenido del mismo, búsqueda en los atributos, etc.

El proceso definido como "Exportación de datos de SQL Server a XML" muestra ventajas claras relacionadas con el *intercambio de información* existente entre las plataformas altamente estructuradas como las BD y plataformas semiestructuradas como XML. Además, el lenguaje extensible posee una estructura que es legible por humanos, separa el contenido de su procesamiento (independencia de datos con respecto a las aplicaciones) y es una tecnología ampliamente difundida.

Formato Estructurado. La información contenida en las tablas de SQL Server quedan a un nivel jerárquico limitado y no dan una mayor información de sí misma, es decir, de los *meta*-datos que los definen. Para realizar una transformación adecuada de la información es necesario que el SQL se defina correctamente antes de ser enviado al procedimiento almacenado, ya que es ahí en donde se detectará si un documento XML está bien o mal formado.

Aunque en el ejemplo no se ha definido explícitamente un DTD o esquema de validación, es posible hacerlo mediante la inclusión en el archivo `template.txt` y así poder tener una comprobación más sólida de los archivos generados por SQL Server. De esta forma podemos garantizar que nuestros *datos cumplen con ciertas restricciones*. Como ejemplo podemos mencionar que algunas de las restricciones que se pueden incluir para mejorar el esquema de validación son:

- Definir marcas obligatorias (elementos mínimos) como `<ApPaterno>`, `<ApMaterno>`, `<Nombre>`, `<sexo>`, `<fechanacimiento>` y `<NivelEstudios>`, y atributos obligatorios como `<numper>` y `<rfc>` para la etiqueta `<Empleado>`.
- Las marcas que componen el documento, sin hacer omisiones son: `<ApPaterno>`, `<ApMaterno>`, `<Nombre>`, `<sexo>`, `<fechanacimiento>`, `<NivelEstudios>`, `<nombreConyuge>` y `<notas>`.
- La estructura en este caso es sencilla y la información se encuentra un nivel más abajo de la etiqueta que define a un empleado en específico o a dos niveles de la

etiqueta que define la tabla de empleados; se compone de ocho etiquetas y dos atributos.

El archivo generado cumple plenamente con el estándar definido por XML, creándose de esta manera un documento XML estructurado.

XML es legible por humanos. El archivo XML generado por SQL Server es resultado de una consulta que puede ser tan específica como lo requiera el usuario. A medida que esta consulta sea más específica (tenga más filtros) cobrará un mayor interés para el personal de la SEC, el documento XML generado puede ser leído directamente desde el explorador de Internet o cualquier otro visualizador de documentos XML sin necesidad de recurrir a otra aplicación o procedimiento para su interpretación.

El documento XML generado podrá exportarse a cualquier Base de Datos, pudiéndose detectar errores mediante una comprobación a simple vista de los datos antes de realizarse el proceso y si al realizar éste llegaran a presentarse errores, se podrán corregir directamente desde el servidor en donde se ejecuta la importación sin necesidad de trasladarse (físicamente o mediante un acceso remoto) a la computadora desde la cual se generó el archivo XML.

XML separa el proceso del contenido. Claramente al momento de extraer la información para incorporarla a un archivo XML, ésta queda en un formato estructurado, con todas las características de XML. Este documento puede procesarse por cualquier aplicación o lenguaje que tenga las capacidades para hacerlo.

La forma de procesamiento en sí no está definida, esa es una tarea para las aplicaciones que necesiten utilizar el archivo XML, de manera que los datos reciban un tratamiento acorde con las necesidades del negocio.

Las rutinas de procesamiento en el caso del Motor de Validación propuesto para el SIMIP pueden ser las que apliquen la normatividad vigente para la verificación del movimiento tales como el cumplimiento de horas estipuladas, la compatibilidad de horarios, etc. En otro caso los datos extraídos a través de las herramientas será necesario guardarlos en una Base de Datos o usarlos como variables de configuración a lo largo de un sistema.

Paso de información a las aplicaciones. Antes de ejecutar el procedimiento almacenado de extracción de datos, en la plantilla definida como `template.txt` en nuestro ejemplo podemos pasar información relevante al momento de procesar el documento XML, tal como la codificación en la que están representados los datos.

El proceso complementario y al cual hemos denominado “Importación de datos existentes en archivos de datos XML a BD SQL Server” muestra que entre más estructurada y clara sea la fuente original de información (lo cual se logra utilizando XML), más fácil será realizar la importación de datos a SQL Server o a cualquier otra Base de Datos que soporte el estándar XML.

XML es un formato de texto estructurado. En el proceso del ejemplo se importan los datos a una tabla en SQL Server con estructura idéntica a la tabla origen por lo que la semántica que nos da la estructura no es muy significativa. Sin embargo sí lo es en el caso de que se quieran importar los datos a un sistema DatawareHouse en el cual la estructura es distinta; es necesario dedicar tiempo a estudiar las transformaciones necesarias para realizar una carga incremental adecuada a los datos existentes que pueden provenir de otras BD. En este punto, la semántica apoyada en gran medida por la estructura jerárquica de XML, es de gran ayuda ya que los campos son identificados mucho más fácilmente y por consiguiente la relación con la nueva estructura se realiza de la forma más precisa posible.

La tecnología de procesamiento de XML está ampliamente difundida y disponible, además de ser económica. Entre los aspectos que preocupan a los directivos de una compañía antes de realizar la implantación de un sistema o un estándar informático nuevo están el costo y la compatibilidad. En lo relativo a XML, estos aspectos en conjunto están plenamente cubiertos por los lenguajes y sistemas de bases de datos que se actualizan de manera constante.

En la SEC el sistema implantado está basado en una plataforma SQL Server, la cual como se ha mostrado cuenta con funciones de compatibilidad (Stored Procedures y utilerías) que permiten realizar la exportación e importación hacia y desde XML. Además la plataforma, que está basada en ASPs sobre controles ADO, permite una fácil integración con todas las funciones de XML y en especial con las referentes al DOM y al SAX. Hemos mostrado aplicaciones para SQL Server utilizando Stored Procedures y DOM con Visual Basic y aplicaciones con Java utilizando SAX incluido en Xerces.

XML es un lenguaje versátil, ampliamente utilizado y es prácticamente un estándar en el intercambio de información. El costo no es un factor determinante ya que es posible hacerlo funcionar en un ambiente GNU-Linux instalado con el JDK y con una interfaz de programación (API) como puede ser Xerces. Un sistema operativo como éste nos brinda las ventajas del software libre y además nos permite tener un servidor o una estación de trabajo al día con las actualizaciones que surjan en la comunidad GNU en torno a XML.

Los costos de administración de sistemas del departamento de informática de la SEC, tenderán a reducirse a medida que al personal le resulte familiar el código de los archivos de configuración del SIMIP que son auto-explicativos, los cambios se realizarán en menor tiempo en comparación a otras aplicaciones.

Un DTD o Esquema ya acordados derivan en documentos intercambiables. Los datos extraídos de la Base de Datos engloban un estándar conformado por un conjunto de elementos definidos, atributos específicos y una estructura que puede especificarse completamente. Esta estructura como tal puede y debe definirse mediante un DTD o esquema XML. Este tipo de estructuras de validación nos permiten analizar cómo está conformado el archivo XML.

Supongamos que en un ambiente de Red se estuvo trasmitiendo y que de un momento a otro se corta la comunicación. Este tipo de esquemas nos permitirá saber si el archivo contiene la estructura correcta, con sus etiquetas de inicio y cierre de elementos, el

contenido de éstos, etc. La validación automática previa se traduce en ahorro de tiempo de validación manual y en evitar errores en documento XML al intentar incorporarlo a una tabla de la BD.

El uso de XML en aplicaciones más extensas como en el caso del SIMIP implantado en la SEC Veracruz permite mejoras. Una de las carencias más notorias de la Secretaría es la falta de unificación de criterios que se solucionan en alguna medida con el sistema en ASPs. En este sentido XML le da orden y sentido a los datos que se intercambian entre las Áreas ya que mediante el uso de los esquemas de validación XML se imponen la estructura y los formatos aprobados por la Dirección permeando al interior de toda la organización.

Los sistemas que tradicionalmente se venían utilizando en el área de Recursos Humanos de la SEC, se basan principalmente en aplicaciones con formatos propietarios como los archivos *.dbf que maneja Microsoft Fox Pro. Estas aplicaciones son muy rígidas y cualquier cambio en la forma de procesamiento influirá directamente sobre la programación interna de la Base de Datos.

En la SEC el uso de esquemas puede hacerse extensivo a los documentos Inter-áreas tal como el comprobante de recepción de documentos de un área a otra, el Aviso de Movimientos de Personal, el Cuadro de Cancelación-Creación, los documentos enviados a SEFIPLAN para la obtención de números de personal, etc.

En una organización gubernamental de tamaño considerable como lo es la SEC, la estandarización es fundamental para que las diversas Áreas puedan comunicarse con una mayor facilidad. Un sistema que implemente XML podrá mostrar claramente sus ventajas y así fomentar su expansión al interior de la Secretaría, pudiéndose definir de esta manera el formato de todos los documentos que se utilizan tales como los Cuadros de Cancelación-Creación, el Aviso de Movimientos de Personal (AMP), las Cédulas de Movimientos de Personal, etc.

En síntesis, XML provee a la SEC de varios beneficios:

- Disminución de errores ocasionados por la ambigüedad de datos que ocurre en archivos de texto plano. La comunicación entre sistemas será más ágil reduciendo para el magisterio el tiempo de proceso de trámites mediante la eliminación de validaciones innecesarias de los datos de los documentos electrónicos.
- El costo de implantación en la Secretaría se reduce al menos por el uso de una herramienta gratuita; hay que considerar los costos de consultoría informática, de las herramientas necesarias que sean ajenas a XML, de modificación de los sistemas informáticos que no manejen XML y de los equipos necesarios. Aún cuando se utilicen herramientas propietarias, como en el caso de Microsoft, podremos estar seguros de que gracias a la amplia difusión y aceptación del lenguaje, siempre vamos a contar con la actualización correcta para la solución XML que estemos implementando sin costos adicionales..

- También hay una reducción de costos al no tener que realizar llamadas al técnico de la solución propietaria para que pueda configurar su sistema escrito en algún lenguaje que sólo la compañía entiende y maneja. El lenguaje XML es claro y el personal de sistemas de la SEC puede realizar los cambios para adaptar la aplicación a sus necesidades.
- La carga y descarga de datos con XML de dispositivos móviles a Bases de datos es un ejemplo propuesto para el funcionamiento del motor de validación mostrado en el capítulo anterior, el cual puede trabajar en lugares apartados mediante la instalación de "ventanillas móviles" en handhelds. Sin embargo, ésta no es la única aplicación que podría tomar ventaja de este concepto, ya que el mismo SIMIP instalado sobre el IIS (Internet Information Service) puede exportar datos en XML para ser importados por aplicaciones que aún se sigan utilizando. Las aplicaciones antiguas mediante la exportación a XML de sus datos cierran así el círculo concerniente al intercambio de información aplicado a sistemas internos.
- Se reduce el costo en el desarrollo de aplicaciones ya que en el proceso de realización de cambios continuos no es necesario estar trabajando con toda la aplicación, por ejemplo, en caso de una corrección de datos entonces se podrá operar sobre el documento XML; por el contrario si es una corrección de estilo podrá trabajarse con CSS, XSLT, o en su defecto con la aplicación asociada con la presentación de información.

Capítulo 7

Conclusiones

En el presente trabajo se ha explorado y se ha propuesto aplicar las características del lenguaje XML para lograr una mejor forma de estructurar y manejar la información del SIMIP en la SEC del Estado de Veracruz, siendo las que mejor se adaptan para este caso: 1) la que proporciona independencia de datos con respecto a las aplicaciones, al hacer posible el intercambio de datos a sistemas como el de nóminas, y 2) la que brinda semántica y estructura a dichos datos ayudando a que personal de sistemas que no tiene nociones de su funcionamiento lo pueda entender rápidamente.

Se han examinado con detalle los componentes del lenguaje XML para la creación de documentos y las herramientas existentes para su validación. Estos componentes permiten desde la definición del lenguaje en su estructura interna, pasando por la forma de validarlo ya sea de forma interna o externa, el uso de entidades para la incorporación de datos binarios, hasta su presentación mediante hojas de estilo en cascada (CSS) o a través del lenguaje de estilos de XML (XSL).

Se ha presentado un panorama general de los datos y procesos que se realizan para los trámites administrativos del sistema de personal de la SEC del estado de Veracruz y los tipos de problemas que se tienen con el sistema actual, mencionando principalmente los que tienen que ver con XML pero sin dejar de mencionar a los que deben cambiarse tal y como lo describen las mejores prácticas en la programación.

A continuación se propone la utilización de XML como herramienta para mejorar la representación semántica de los datos que se manejan; se consideró la forma de intercambiar datos en representación XML y datos contenidos en bases de datos relacionales; posteriormente se analizaron diversas herramientas basadas en tecnología Java y disponibles en forma libre para realizar procesamiento de documentos XML bajo dos enfoques principales, DOM y SAX. A través del procesamiento es posible presentar el mismo documento XML de diferentes formas para uso final o para intercambio con otras aplicaciones.

Se han mencionado algunos criterios que permiten evaluar las bondades de la tecnología XML. Concluimos que dicha tecnología ofrece ventajas de legibilidad, disponibilidad y bajo costo de desarrollo y aprendizaje, independencia de datos, interoperabilidad y flexibilidad para definir nuevos sistemas.

La principal desventaja actual de XML radica en el alto consumo de recursos de hardware y el tiempo de procesamiento requerido; esta situación debe cambiar pronto considerando el continuo avance en las tecnologías de almacenamiento y procesamiento.

Bibliografía

[Delgado, 2001] Delgado, Albert, "Microsoft SQL Server 2000", Prentice Hall, 2001, Madrid, España ISBN 8420530131.

[FunderBurk, 2002] FunderBurk J.E., Malaika S., Reinwald B., "XML programming with SQL/XML and XQuery", IBM SYSTEMS JOURNAL, VOL. 41, No. 4, 2002.

[Hunter, 2000] Hunter, David. "Beginning XML", Ed. Wrox 2000 ISBN 1861005598.

[Kalani, 2004] Kalani, A., Kalani P., "Developing XML Web Services and Server Components with Visual C# .NET and the .NET Framework, Exam Cram 2", Ed. Que, 2004, EUA ISBN 0789728974.

[Marchal, 2000] Marchal B., "XML by Example", Ed. Que, 2000, EUA.

[Martínez, 2001] Gutierrez A., Martinez R., "XML a través de ejemplos", Alfaomega, 2001, México.

[Rusty, 1999] Rusty H. Elliote, "XML Bible", IDG Books Worldwide, 1999, EUA ISBN 0764532367.

[Rusty, 2003] Rusty H. Elliote, "Processing XML with Java: a guide to SAX, DOM, JDOM, JAXP, and TrAX", Addison Wesley, 2003, EUA ISBN 0201771861.

[Sills, 2002] Sills A., Ahmed M., Boumphrey F., Et. Al, "XML .NET Developer's Guide", Syngress Publishing, Inc., 2002, EUA ISBN 1928994474.

[Sturm, 2000] Sturm, Jake, "Developing XML Solutions", Microsoft Press, 2000, EUA ISBN 0735607966.

[T. Ray, 2001] T. Ray, Erik, "Learning XML", O'Reilly, 2001, ISBN 0596000464.

[ZapThink, 2001] ZapThink Research Report, The "Pros and Cons" of XML, ZapThink LLC. 2001.

Referencias electrónicas

[DOM L2] DOM Nivel 2 versión 1.0: http://www.w3.org/TR/DOM-Level-2/

[HTML 4] Especificación de HTML 4.0: http://www.w3.org/TR/WD-html40

[NAMESPACES] Espacios de Nombres: http://www.w3.org/TR/REC-xml-names/

[SAX] Simple API for XML: http://www.saxproject.org/

[SEP] Secretaría de Educación Pública: http://www.sep.gob.mx

[Stylus] Stylus Studio 2006: http://www.stylusstudio.com/xml_parsers.html

[Xerces] Apache XML project Xerces: http://xml.apache.org/xerces2-j/xni-xerces2.html

[XML Parsers] Microsoft MSXML XML Parser:
http://www.topxml.com/parsers/default.asp

[XMLNOTEPAD] XML Notepad: http://msdn.microsoft.com/xml

[XMLSPY] XML Spy: http://www.xmlspy.com