LA REVOLUCION XML

Uno para todos

EN NUESTRO NUMERO ANIVERSARIO, Y A CASI 10 AÑOS DE SU NACIMIENTO, LES CONTAMOS LA INCREIBLE EVOLUCION DE ESTE METALENGUAJE, PRESENTE EN TODAS LAS TECNOLOGIAS Y PLATAFORMAS QUE SE CONSIDEREN MODERNAS. EN ESTAS PAGINAS LES BRINDAMOS LAS BASES PARA CONOCER UN FORMATO QUE ESTA LLAMADO A SER UNO DE SOPORTES DE LAS COMUNICACIONES EN LOS PROXIMOS AÑOS.

Cristian Ornia
Especialista en pizza a la piedra
cristianpiedra@muzzarella.com.ar

Seguramente, muy pocos imaginaban, en aquella conferencia sobre SGML de mediados de los años noventa, que la presentación de un reporte técnico sobre un lenguaje derivado podría causar tanto alboroto en los años sucesivos. Lo cierto es que, aún hoy, no resulta nada fácil definir en pocas palabras un metalenguaje tan interesante y abarcativo como XML. Actualmente, XML está presente de alguna u otra manera en todas las plataformas tecnológicas como lenguaje común para el intercambio de información. Se funde y relaciona con muchos lenguajes y plataformas existentes, y siempre los dota de facilidades ampliamente reconocidas, distinguiéndose como un estándar global que crece sin pausa y pasa a ser sostén y nexo de dichas implementaciones.
Para aquel desarrollador que se asoma a este tema, sería conveniente proponerle que imagine XML como un gran molde que permite crear nuestros propios sublenguajes de descripción o marcas. De este modo, cada sublenguaje basado en él define su propia gramática y su propio set de reglas que gobiernan el contenido y la estructura de documentos escritos en él. Puesto que cada lenguaje derivado debe satisfacer este requisito gramatical, XML proporciona las facilidades para manejar correctamente la gramática de cualquiera de ellos.

Qué incluye un documento XML

Para empezar a trabajar en XML conviene conocer en detalle la estructura básica de su conformación. Un documento xml está compuesto por un conjunto de entidades, básicamente, texto y etiquetas.
A nivel lógico, incluye declaraciones (que definen elementos), comentarios, referencias a caracteres e instrucciones de proceso, que lo definen como autodescriptivo. Por eso, veremos siempre que un xml contiene un área de prólogo o descripción y un cuerpo propio con etiquetas que determinan el contenido de los elementos. Con esa estructura, un documento xml puede ser visto como un árbol, que se ramifica en incontables ramas o nodos. Repasemos este tema mediante un ejemplo simple para entenderlo mejor:

```
<?xml version="1.0" encoding="ISO8859-1"?>
<agenda xmlns="http://tempuri.org/ejemplo.xsd">
    <contacto indice="1">
        <nombre>Gisela Carina</nombre>
        <nacionalidad>Argentina</nacionalidad>
    </contacto>
    <contacto indice="2">
        <nombre>Maria Luisa</nombre>
        <nacionalidad>Mexico</nacionalidad>
    </contacto>
    <contacto indice="3">
        <nombre>Mariel Liria</nombre>
        <nacionalidad>Argentina</nacionalidad>
    </contacto>
</agenda>
```

Este archivo, al que llamaremos ejemplo.xml, nos servirá de referencia para analizar los aspectos principales de un documento xml bien formado:
→ **1)** La declaración XML describe algunas de las propiedades generales del documento, como la versión, el tipo de codificación y si existen declaraciones de marcas externas a la entidad documento.
→ **2)** El elemento base se llama root o elemento raíz (←agenda→).
→ **3)** Los nodos internos son elementos (←contacto→).
→ **4)** Las propiedades de esos nodos o elementos se denominan atributos (indice="1").
El orden de los hijos de un nodo es muy importante y debe conservarse de forma similar a como se ve en la figura.

Por lo general, decimos que un documento está "bien formado" si se atiene correctamente a una serie de reglas gramaticales propias de XML. Es importante seguir estas reglas, a fin de no pasar por problemas de validación que luego no podamos localizar fácilmente.
A todo objeto, programa o función que analiza y procesa dicha información se lo llama procesador XML o, más comúnmente, parser.
Cuando el parser toma el archivo xml, busca un archivo "guía", a fin de verificar y constatar que esas reglas se cumplan correctamente. Esos archivos de verificación están basados, en su mayoría, en dos tecnologías relacionadas: la definición del tipo de documento (*Document Type Definition* o DTD) y la definición del esquema de XML (XML-Shema).

Tecnologías relacionadas

SIN DUDAS, CUANDO SE EMPIEZA A APRENDER XML, LO PRIMERO QUE CREA CONFUSIÓN ES LA CANTIDAD DE ACRÓNIMOS Y SIGLAS QUE EXISTEN.

Sin dudas, cuando se empieza a aprender XML, lo primero que crea confusión es la cantidad de acrónimos y siglas que existen. Por eso, es importante ir de a poco, releer una y otra vez cada tecnología o sigla involucrada y, si es posible, anotar manual o mentalmente alguna definición corta a la que podamos recurrir al relacionar dichos acrónimos.
Decíamos que las reglas de un documento XML pueden ser definidas de dos formas distintas: los DTD y los XML-Schema. Estos sistemas de reglas permiten establecer la cantidad, el orden (y su anidación) y la especificación de los atributos que debe tener cada una de las etiquetas del archivo xml relacionado.
En la mayoría de los casos, deberemos optar entre estos dos metalenguajes que nos proveen de las herramientas necesarias para definir las tecnologías que podemos obtener partiendo de XML. A continuación veremos sus características.

DTD

El ya clásico *Document Type Definition* es un archivo de texto que consta de un conjunto de reglas acerca de la estructura y el contenido de documentos xml.
Un DTD permite describir y usar un formato común de datos, verificarlos al intercambiarlos y mantener la consistencia en general.
Esta definición se enfoca sólo a declaraciones de elementos y deben seguirse ciertas condiciones para cada caso:
→ El elemento raíz del documento tiene que ser del mismo tipo de documento definido en el archivo DTD.
→ Aquellos elementos permitidos como contenido de otro elemento deben respetarse.
→ Los elementos nunca pueden tener atributos que no hayan sido oportunamente declarados en el DTD.
Un archivo DTD enumera un sistema válido de elementos que pueden aparecer en un documento de XML, incluyendo el orden y sus cualidades. Es decir, describe los elementos (tanto los tags permitidos como sus contenidos), la estructura (el orden en el que van los tags en el documento) y el anidamiento (qué etiquetas van dentro de otras marcas).
Es interesante destacar que un mismo DTD puede residir en un archivo externo (incluso compartido por muchos documentos).
Siguiendo el documento xml que vimos antes, podríamos ejemplificar su DTD de esta manera:

```
<!ELEMENT  contacto (nacionalidad, nombre)>
<!ELEMENT nombre (#PCDATA)>
<!ELEMENT nacionalidad (#PCDATA)>
```

Esquemas XML o XML-Schema

Con el paso del tiempo, los esquemas XML resultaron más completos y expresivos que los DTD; pasaron a solucionar los inconvenientes de aquel esquema primario y representaron una evolución en términos prácticos y sintácticos por sobre los DTD. En efecto, los DTD no sólo utilizaban un lenguaje propio sino que también carecían de propiedades necesarias para el desarrollo de XML, por ejemplo, el uso de espacios de nombre o namespaces.

A nivel de tipo de datos, es posible utilizar **Datatypes**, encargados de definir los tipos base que se pueden utilizar dentro de esquema de XML, así como **Arquetipos** (tipos definidos por el usuario, extensibles en términos de herencia), nombrándolos y empleándolos en distintas partes dentro del esquema. También se permite agrupar atributos y, por supuesto, se soportan espacios de nombres.
Sin embargo, en un principio, se les atribuía a los esquemas XML cierta complejidad, que requería procesamientos más lentos, y una incompatibilidad de herramientas que hoy ya no es tan visible en el mercado.
Un ejemplo de esquema XML basado en el archivo antes visto podrá aclararnos el panorama:

```
<?xml version="1.0" encoding="UTF?8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<!?? Define un nuevo tipo de datos para un elemento ??>
<xsd:complexType name="Entidad_Agenda">
<xsd:sequence>
<xsd:element name="nombre" type="xsd:string"/>
<xsd:element name="nacionalidad" type="xsd:string"/>
</xsd:sequence>
</xsd:complexType>
<!?? Crea un  elemento de este tipo ??>
<xsd:element name="Contacto" type="Entidad_Agenda" />
</xsd:schema>
```

Otros esquemas

Primero, a la sombra de DTD ,y en segundo término de XML-Shema, siguen surgiendo nuevas propuestas. En ese contexto podemos ubicar a los pioneros **RELAX y TREX**, que derivaron en **RELAX NG**, surgido como la evolución inexorable de los **XML DTDs**.
En efecto, RELAX NG sigue una definición de tipo gramatical y no necesita ser definido en el archivo XML origen. Permite trabajar con diversos tipos de datos e importar tipos, todo de manera realmente sencilla. Sin embargo, aún no cuenta con el rótulo de iniciativa oficial del W3.ORG. Así y todo, la lista continúa con nuevos esquemas, que van saliendo día a día (Schematron, Assertion Grammars, DSD, Examplotron, etc.).

Espacios de nombre (namespaces)

Seguramente, ya habrán escuchado este término en los lenguajes y plataformas más tradicionales. En XML, el namespace nace de la necesidad de evitar confusión entre etiquetas del mismo nombre;por esto, se dice que cada namespace tiene un identificador único.
Los conflictos pueden surgir cuando los tags que se usan son iguales pero tienen propósitos diferentes; esto puede suceder, incluso, dentro de un único documento Además, estos problemas pueden aparecer (entre dos o más documentos) cuando las aplicaciones tienen que procesar documentos provenientes de distintas fuentes.
A su vez, los espacios de nombre cumplen un papel clave en la necesidad fundamental de reutilizar el código y no tener que recodificar varias veces un mismo tramo de elementos o instrucciones.
Las etiquetas y los atributos se hacen únicos añadiéndoles un prefijo identificador del espacio de nombre. Así, cada namespace será identificado en forma inequívoca por un dominio de URL único.
Los namespaces son organizados en particiones globales y específicas de elementos. De esta forma, resolvemos el problema de los conflictos de nombres dentro de un único documento.
Veamos un ejemplo que aclarará el panorama, dados dos documentos xml:

```
<?xml version=1.0" encoding="iso-8859-1"?>
<impuesto>
  <nombre>Inmobiliario</nombre>
  <base>imponible</base>
  <periodicidad>anual</ periodicidad >
  <cuotas>4</cuotas>
</impuesto>

<?xml version="1.0" encoding="iso-8859-1"?>
<factura>
  <cliente>Oscar Ruben </cliente>
  <subtotal>20.30</subtotal>
  <impuesto>1.40</impuesto>
</factura>
```

En este caso, tendríamos dos elementos con el mismo nombre (←impuesto→) pero con características distintas.
En otro documento, podríamos referenciar ambos elementos y no serían iguales. Veamos cómo hacerlo:

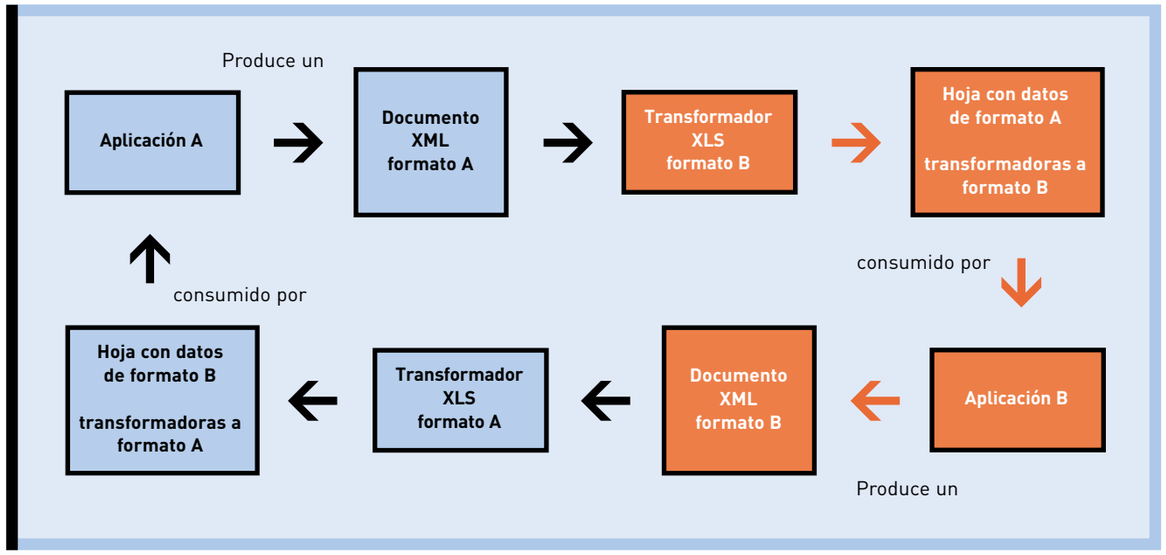
```
<?xml version="1.0" encoding="iso-8859-1"?>
<tipimp:tipos>
  <tipimp:impuesto xmlns:tipimp="http://www.dgi.com/tipos">
   <tipimp:nombre>Inmobiliario</tipimp:nombre>
   <tipimp::base>imponible<tipimp::base>
   <tipimp:periodicidad>anual</tipimp:periodicidad >
   <tipimp:cuotas>4</cuotas>
   </tipimp:tipos>
   <factimp:factura xmlns:factimp ="http://www.dgi.
com/facturas">
     <factimp:cliente>Oscar Ruben </factimp:cliente >
     <factimp:subtotal>20.30</factimp:subtotal>
     <factimp:impuesto>1.40</factimp:impuesto >
   </factimp:factura>
</empresa>
```

XSL y los lenguajes XSLT, XSL-FO y XPATH

La XSL (acrónimo de *XML Style Language*) es una especificación desarrollada dentro del W3C en 1998 para aplicar formato a los documentos xml de forma estandarizada.
Así como XML surge como una versión mejorada y simplificada de SGML, XSL nace relacionado a DSSSL.
En términos prácticos, XSL nos permite transformar un documento xml en otro.
En el principio, a XSL se lo dividió en dos secciones: un lenguaje para transformar documentos xml (XSLT) y otro de formateo (XSL-FO). En pleno desarrollo de la especificación, XSLT se emancipó de XSL; al primero se le dio un tratamiento mucho más independiente en su determinación, y la gente de XSL se centró más en el lenguaje de formateo inicial. De todos modos, ambas tecnologías se relacionan entre sí y, generalmente, en la jerga informática se las confunde e integra. Si bien XSL hace un uso intensivo de las plantillas XSLT, XSLT está diseñado para ser utilizado en forma independiente de XSL.
En pleno proceso de desarrollo de XSLT, la sección del lenguaje de transformación que hacía referencia a las maneras de acceder y moverse por los distintos nodos de un documento XML fue separada de XSL 1.0 y se la trabajó independientemente. Estamos hablando de **XPATH** (*XML Path Language*).

→ **XSL-FO**: es un lenguaje que permite formatear documentación XML para su impresión. Verifica cómo se definen los objetos de formato con el objetivo de convertir XML en formatos binarios.
→ **XSLT**: trabaja en ese nivel como especificación independiente que desarrolla el lenguaje de transformación usado por XSL.
→ **Pat.**: es la especificación que desarrolla una comunicación para acceder a todos los elementos de un documento XML y se la utiliza desde XSLT. Para hacerlo más sencillo, se podría definir a XPATH como un lenguaje que permite definir y seleccionar distintas áreas dentro de un documento XML.
Es por eso que mucha veces, hablando de XSL, indirectamente estamos citando a XSLT , XSL-FO o XPATH y viceversa. Incluso, se suele simplificar el tema enunciando a XSL-FO, XSLT y XPATH como sublenguajes de XSL, otorgándole a cada uno una función distinta.
Como es sabido, los documentos XSL están escritos en XML. Por lo tanto, en general se necesita utilizar un procesador de hojas de estilo para procesarlos, relacionándolos a un documento xml determinado. La flexibilidad de XSL alcanza, en su nivel de salida, información en HTML, XTML, WML, XML y hasta en texto plano.

{ Esto es texto simulado ea ea papei persistencia usar forma parte de la definición de la arquitectura. }



[Figura X] El gráfico ejemplifica una cadena de procesamiento XSL entre aplicaciones.

Además, XSL (específicamente, XSL-FO) es una opción mucho más efectiva y adecuada que CSS. ¿Por qué decimos esto? Relacionado con lo antes expuesto, vemos que XSL, en conjunto con XML, unifica la visualización de datos en diferentes equipos y plataformas. Esto abarca desde un simple navegador web en nuestra PC hasta cualquier equipo móvil, no importa su marca o tecnología. Sin embargo, debemos ser justos: hoy por hoy, XSL funciona sólo en documentos XML, no así en HTML.
En noviembre de 2004 se lanzó la versión 2.0 de XSLT, que contiene notables mejoras, aunque por el momento aparece más como una tentativa de trabajo, con la meta fija en ser un estándar.
Para realizar ese proceso de conversión del que hablamos, XSLT, a la manera de cualquier lenguaje de programación tradicional, se vale de funciones específicas. En este sentido, es muy común ver aplicaciones y códigos desarrollados para convertir un documento xml en HTML utilizando XSLT.
Es que XSL y XSLT nos permiten realizar todo tipo de operaciones dentro del XML, con el objeto de manipularlo. Esto significa, por ejemplo:

→ Reordenar elementos (veamos primero los nombres y luego las nacionalidades).
→ Filtrar elementos (visualicemos sólo los domicilios).
→ Cambiar un elemento por otro (en vez de ←nombre→, usaremos ←nombre_y_apellido→).

Repasando algunos comandos básicos de XSLT, encontramos: ←xsl:template→, ←xsl:template match="/"→ ←xsl:if→ , ←xsl:for-each→ , ←xsl:for-each→.
Estas son sólo tres de unas cuantas instrucciones, pero se mencionan como introducción a un tema apasionante: **XPATH**. En efecto, las expresiones XPATH se detectan dentro de atributos tales como match, select o test. Si bien este tema podría abarcar uno o varios libros, es muy importante vislumbrar el poder y el control que nos permite ejercer sobre nuestros documentos. Con XPATH podemos realizar todo tipo de operaciones, como trabajar con niveles de nodos interrelacionados a partir de nuestro nodo actual, descender por jerarquías, efectuar restricciones específicas, conocer el número de posición de una rama en relación a otros nodos extraídos, etc.
XSL, actualmente, cuenta con varios engines que trabajan del lado del servidor enviando el resultado modificado al cliente, y son usados con XSLT en el proceso de transformación de documentos XML. Los más importantes son: XT, XALAN y SAXON.

XQuery

XQuery es un nuevo lenguaje de consultas, utilizado por la especificación XML para definir consultas y manejar los resultados. Si bien la W3C aún debe resolver ciertos aspectos, y ésta es una tecnología en formación, esta especificación supone un avance en relación a XQL, que, si bien es similar, tiene representaciones distintas y carece de algunos agregados que hacen más poderoso a XQuery.
XQuery posee una flexibilidad bien amplia y conocida, ya que permite consultar una variada gama de fuentes de datos, que incluye distintas bases relacionales, documentos xml, servicios Web, aplicaciones, etc.
Para muestra veamos un pequeño ejemplo: suponoendo que tenemos una determinada estructura de datos, del tipo:

```
<biblioteca>
    <libro año="2004">
        <titulo>PROGRAMACION PARA
CELULARES CON JAVA</titulo>
       <autor> Maximiliano Firtman </autor>
        <precio> 40</precio>
    </libro>
    <libro año="2004">
        <titulo> C++ PROGRAMACION
ORIENTADA A OBJETOS</titulo>
        <autor> Diego Ruiz </autor>
        <precio> 35</precio>
    </libro>
    <libro año="2005">
        <titulo>ASP.NET</titulo>
       <autor> Maximiliano Firtman </autor>
        <precio> 40</precio>
    </libro>
</biblioteca>
```

En este caso, visualizamos una posible consulta de libros de Maximiliano Firtman editados por el autor después del año 2004.

```
<biblioteca>
 {
  for $libro in doc("http://www.tectimes.com/listalibros.xml")/
biblioteca/libro
  where $libro/autor = "Maximiliano Firtman" and $libro/@año> 2000
  return
    <libro año="{ $libro/@año }">
     { $libro/titulo }
    </libro>
 }
</ biblioteca >
```

La variable $libro se refiere al nodo libro, y en función de eso se va trabajando dentro del bucle. Como podrán notar, de una forma semejante a como trabajaríamos en SQL, filtramos datos contenidos en los elementos de los subnodos. Dicha consulta nos devolverá:

```
<libro año="2005">
        <titulo>ASP.NET</titulo>
        <autor> Maximiliano Firtman </autor>
        <precio> 40</precio>
 </libro>
```

Aclaramos que los datos no son totalmente reales, pero sirven para el ejemplo. Con XQuery también podemos, en una sola consulta, unificar resultados provenientes de una tabla de datos y un documento xml, todo en forma conjunta y unificada. Esta tecnología está profundamente relacionada a XPATH.

Xlink y Xpointer

Xlink es un lenguaje XML que permite especificar enlaces entre distintos documentos xml. Todo el manejo de enlaces en XML se realiza a través de XLink. Hoy en día, si la localización de un sitio web se modifica, quedan vínculos no resueltos y todos los enlaces de esa URL deben ser modificados manualmente. Esta especificación permite una localización más precisa y ordenada de la información, así como el uso de enlaces bidireccionales. ¿Qué significa esto? Esta clase de vínculos permite a los usuarios activar un enlace desde cualquiera de las partes seleccionadas. XLink también da la posibilidad de redireccionar enlaces a través de un catálogo de vínculos. Con ello, cuando la ubicación de un documento se modifica, sólo deberemos cambiar la entrada en dicho catálogo. Además, abarca más posibilidades que el clásico tag ←a href=...→ ... ←/a→, que nosotros enseguida relacionamos de HTML cuando nos nombran una etiqueta de enlace. Veamos un ejemplo de código:

```
<?xml version="1.0" encoding="UTF-8"?>
<enlaces>
    <cuentos xmlns:xlink="http://
www.w3.org/XML/XLink/1.0" >
        xlink:type = "simple"
        xlink:show = "new"
        xlink:href=" www.gabrielwalter.
com.ar ">
        Cuentos fantasticos de Gabriel
Walter
    </cuentos>?
</enlaces>
```

También podrán visualizar dos atributos muy importantes de un Xlink:

→ **Show**: declara lo que ocurre cuando se pulsa el enlace. Por ejemplo, mostrarlo en la misma ventana (replace), en otra (new) o bien hacer que el contenido del texto del vínculo enlazado sea incluido en el lugar del link, procesándose como si fuera parte de dicho documento original (parsed).

→ **Type**: determina dos tipos de enlaces:

→ Simples (Simple): son muy similares a los de html; es decir, se enlaza desde una página local a otra remota.

→ Extendidos (Locator): pueden tener más de un recurso por enlazar. De todos modos, esta posibilidad está siendo analizada y no hay navegadores que la soporten.

Xpointer es un lenguaje complementario, basado en XPATH, que permite realizar enlaces a distintos sectores concretos de un documento o, incluso, a determinados elementos individuales (siempre dentro de la estructura interna jerárquica de un documento xml).
Como ejemplo, podríamos usar Xlink para definir un enlace a un párrafo establecido dentro de un elemento determinado. En este caso, usaríamos la forma xpointer(xpath), del tipo xpointer(//nombre). Como detalle también podemos destacar que un Xpointer puede estar unido a otro. Por ejemplo: **xpointer(//nombre)xpointer(//apellido)**.
En el siguiente ejemplo, podremos cotejar cómo el Xpoint nos permite relacionarnos con un elemento específico ("specialreport") dentro de un espacio de nombre determinado ("seccion"):

NOMBRE	VENTAJAS	DESVENTAJAS
DOM (orientado a estructuras)	Más usable. Proporciona un árbol jerárquico de datos ordenado. Recomendado para accesos frecuentes y modificaciones precisas de elementos.	Los requerimientos de memoria. No es aconsejable en documentos muy grandes.
SAX (orientado a eventos)	Más rápido. Interfaz muy sencilla. Requiere menos memoria que DOM. Permite cortar un análisis en pleno parseo.	No permite retroceder para manipular información. Deja extraer sólo las partes más pequeñas de un documento.


```
<revistas xmlns:xlink ="http://www.w3.org/
XML/XLink/1.0"
     xlink:type = "simple"
     xlink:show = "new"
     xlink:href="http://www.tectimes.com/
revistas.xml#xmlns(seccion=
          "http://www.code.tectimes.com/
seccion:specialreport">
     Code
</revista>
```

Como verán, este Xpoint puntual apunta al elemento ←seccion:specialreport→.
La única mala noticia es que, al igual que Xlink, aún no hay navegadores que soporten Xpointer, porque todavía es una tecnología en desarrollo.

Especificaciones

Existen dos especificaciones de análisis de información XML que, actualmente, son consideradas las más importantes: una es **XML-DOM** (o DOM, acrónimo de *Document Object Model*) y la otra es **SAX** (*Simple API for XML*). Si bien ambas tienen un objetivo en común (analizar o "parsear" el contenido de un archivo xml), lo hacen de diferentes maneras y tienen sus particularidades.
La ventaja más clara de DOM es que otorga una gran flexibilidad cuando necesitemos referir y manipular la información de un nodo en cualquier sector del árbol.
El concepto de SAX es bastante distinto en su forma de analizar los datos, ya que realiza un procesamiento secuencial orientado a eventos (*event-driven*). Es el candidato ideal a la hora de manejar grandes archivos de datos. Además, la comunidad en general lo considera mucho más sencillo de usar que DOM. De todos modos, no todas son rosas para SAX. La misma naturaleza de su metodología de procesamiento le impide al analizador modificar nodos ya procesados. En cambio, en DOM esto es muy normal, ya que tenemos un árbol jerárquico de elementos cargado completamente en memoria, apto para ser modificado cuando sea necesario mediante el analizador y no, como en SAX, delegándole la tarea a la aplicación que recibe el contenido.
Veamos las diferencias:

El futuro de XML

El escenario con Internet como epicentro es clave. Y acá se trata de avanzar o retroceder drásticamente frente a otras empresas o hasta frente al poderoso movimiento Open Source. No nos olvidemos de que uno de los elementos clave de Longhorn es el prometido XAML, un lenguaje derivado de XML que servirá para crear interfaces gráficas y un camino para usar las futuras APIs de Avalon, a través de tags. Y por si todavía faltaran aditamentos, la respuesta más directa a XAML que se vislumbra vino de la Mozilla Organization, que ya dejó implementada una especificación de XML llamada **XUL** (*XML User Interface Language*), orientada, básicamente, a optimizar su famoso navegador.

Fuera de las discusiones y especulaciones pendientes, las perspectivas para XML no pueden ser mejores. Su crecimiento ha sido sostenido a la par del avance de Internet y de las comunicaciones vía servicios web. Asimismo, las principales empresas tecnológicas y financieras han confirmado su respaldo y la expansión sigue día a día. Ahora bien, ¿cuál es el techo real que tiene esta tecnología? O mejor aún, más que XML, ¿hasta dónde llegarán todas las tecnologías derivadas de XML?
Si bien la respuesta no es fácil, tampoco parece ser desalentadora, y está ligada especialmente al desarrollo y la optimización de algunas tecnologías derivadas (por ejemplo, XForms, Xpointer, Xlink, y muchas otras implementaciones que toman fuerza propia). También es cierto que mucho tendrá que ver el soporte que puedan hacer de esas tecnologías las principales aplicaciones Web (por ejemplo, los nuevos navegadores). Cuando hablamos de lenguajes con muchísimo futuro y que merecen un párrafo aparte, nos referimos a **SMIL**, un lenguaje de integración y sincronización de archivos multimedia, **VoiceXML** (*Voice Extensible Markup Language*) y **XForms**.
¡XML ha venido para quedarse!

GLOSARIO

- → **Metalenguaje:** lenguaje que permite generar otros lenguajes, en este caso, de marcación o tags.
- → **Elemento:** un elemento representa la estructura lógica de datos dentro un documento xml.
- → **Parser:** para XML, es todo objeto procesador determinado que lee un documento XML, y define la estructura y las propiedades de esos datos.
- → **Entidad:** en XML, una entidad puede ser vista como una unidad de almacenamiento virtual. Puede residir en un archivo externo, ser una cadena o un valor proveniente de una base de datos.
- → **Metadatos:** el término se usa, generalmente, para describir información sobre datos (condición, calidad y características principales).

IMPLEMENTACIONES DESARROLLADAS CON XML

RSS (Really Simple Syndication): lenguaje XML desarrollado específicamente para suministrar información que cambia con frecuencia desde diferentes sitios de noticias.

RDF (Resource Description Framework): una de las especificaciones derivadas de XML, con gran crecimiento en los últimos tiempos.

SVG (Scalable Vector Graphics): formato estándar abierto que describe gráficos vectoriales bidimensionales estáticos y animados en XML.

GML (Geography Markup Language): está descrito como una gramática en XML Schema para el modelaje, transporte y almacenamiento de información geográfica.

OSD (Open Software Description Format): este formato abierto de descripción busca implementar correctamente el desarrollo de software en múltiples plataformas, describiendo su distribución a través de la Red.

CML (Chemical Markup Language): lenguaje utilizado para aplicaciones de química, que describe gran cantidad de fórmulas.

MathML (Mathematical Markup Language): especificación muy extendida para el área de las matemáticas.

EDI (Electronic Document Interchange): vocabulario xml que facilita el intercambio electrónico de datos; si bien ya se lo utiliza, aún está en fase de desarrollo.

Tecnologías integradas a XML

NET y XML

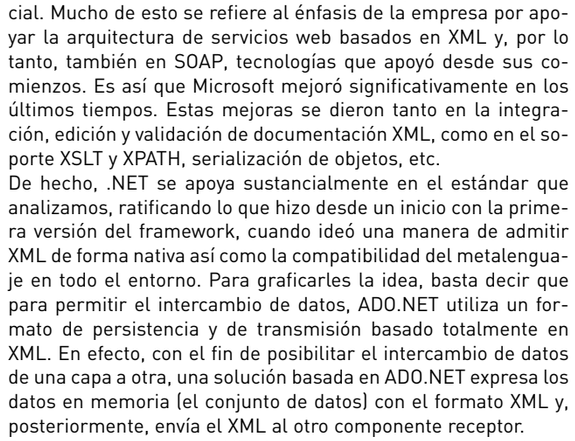
Si bien hay que reconocer que Microsoft ya sustentaba toda su estrategia de negocios en XML desde hace ya bastante tiempo, también es justo destacar que año tras año mejora sustancialmente el espacio que le otorga dentro de su plataforma comercial. Mucho de esto se refiere al énfasis de la empresa por apoyar la arquitectura de servicios web basados en XML y, por lo tanto, también en SOAP, tecnologías que apoyó desde sus comienzos. Es así que Microsoft mejoró significativamente en los últimos tiempos. Estas mejoras se dieron tanto en la integración, edición y validación de documentación XML, como en el soporte XSLT y XPATH, serialización de objetos, etc.

De hecho, .NET se apoya sustancialmente en el estándar que analizamos, ratificando lo que hizo desde un inicio con la primera versión del framework, cuando ideó una manera de admitir XML de forma nativa así como la compatibilidad del metalenguaje en todo el entorno. Para graficarles la idea, basta decir que para permitir el intercambio de datos, ADO.NET utiliza un formato de persistencia y de transmisión basado totalmente en XML. En efecto, con el fin de posibilitar el intercambio de datos de una capa a otra, una solución basada en ADO.NET expresa los datos en memoria (el conjunto de datos) con el formato XML y, posteriormente, envía el XML al otro componente receptor.

Pero vayamos al punto: .NET incorpora un buen grupo de clases que facilitan el manejo general de documentos, nodos y elementos de XML. En relación a las clases que usaremos, en general veremos que **XmlReader**, **XPathNavigator** y **XMLwriter** nos sirven para analizar y escribir información xml, en tanto que XmlDocument permite editar un documento xml en su conjunto.

En cambio, si necesitamos hacer un proceso de serialización, utilizaremos con más frecuencia XmlSerializer junto con XmlWriter, mientras que para efectuar consultas XPATH o bien algún proceso de transformación XSLT en el objeto, deberemos implementar las clases **XslTransform**, **XmlSchema** y **XPathNavigator** (en un futuro, .NET busca expandir las APIs de XML basadas en cursores de esa última clase).

A continuación, veremos un ejemplo muy simple de cómo **Visual Studio .NET** facilita el acceso a datos contenidos en XML y utiliza **System.Xml**. Dado el archivo ejemplo.xml, crearemos una aplicación de consola que lo lea en forma completa. Para hacerlo, usaremos **XmlTextReader**, una función más rápida que **XmlDocument()**, que trabaja sobre DOM, con el consabido gasto de carga en memoria.



Un cuadro representará la estructura del esquema xml.

Recordemos copiar este archivo al directorio bin de nuestro proyecto para poder sumarlo luego a él. En este punto, es interesante remarcar que el IDE nos provee de un menú con las opciones de crear el XML-Schema (que guarda bajo la extensión xsd), así como también de validar la estructura xml. Junto con el esquema que generamos, el diseñador de esquemas xml crea automáticamente otro archivo (de extensión xsx) que contiene información de diseño para componentes de la superficie de diseño.

Una vez elegidos el lenguaje y el tipo de aplicación, vamos al código y agregamos (importamos) el namespace de XML (debemos verificar también que existe una referencia al ensamblaje System.Xml.dll):

```
Imports System.Xml
```

A continuación, creamos el objeto lector a partir de la clase XmlTextReader e invocamos al archivo xml:

```
Dim reader As XmlTextReader = New XmlTextReader
("books.xml")
```

A continuación, creamos un bucle que vaya recorriendo el archivo leyendo línea por línea el nombre de cada etiqueta:

```
Do While (lector.Read())
            Console.WriteLine(lector.Name)
Loop
```

Y agregamos, al final, una pausa en la consola con:

```
Console.ReadLine()
```

Como verán, al ejecutar este código, nos devolverá el nombre de todos los tags que tenemos. Como queremos traer también los contenidos de cada nodo, utilizaremos la propiedad **NodeType** dentro de un Select case, que nos informará si estamos al principio, al final o bien dentro del texto de un elemento. También verificamos si existe algún atributo dentro del nodo o elemento y, en ese caso, mostramos su nombre y valor. Como guía, pueden observar los comentarios detallados antes de cada procedimiento:

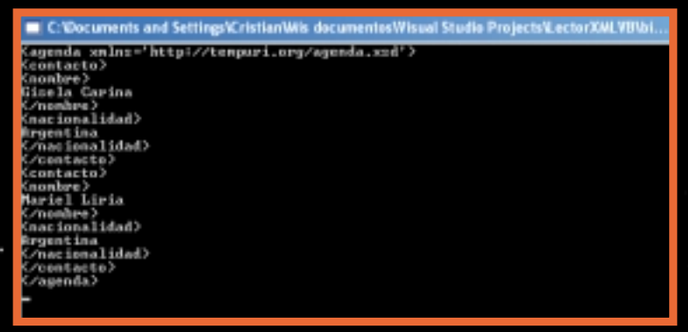
```
Select Case lector.NodeType
Mostramos el comienzo del elemento.
                Case XmlNodeType.Element
                    Console.Write("<" + lector.Name)
                    'Si un atributo existe
                    If lector.HasAttributes Then
                        'Mostramos el nombre y valor
del atributo.
                    While lector.MoveToNextAttribute()
                        Console.Write(" {O}='{1}'",
lector.Name, lector.Value)
                        End While
                    End If
                    Console.WriteLine(">")
    'Mostramos el texto de cada elemento.
                Case XmlNodeType.Text
                    Console.WriteLine(lector.Value)
    'Mostramos  el final del elemento.
                Case XmlNodeType.EndElement
                    Console.Write("</" + lector.Name)
                    Console.WriteLine(">")
End Select
```

Al probar nuestro ejemplo, en el mismo Visual Studio obtenemos por consola la lectura del archivo XML.

Java y XML

DESDE EL PRINCIPIO, CUANDO LA MAYORÍA DE LOS PROGRAMAS RELACIONADOS CON XML ESTABAN VINCULADOS A JAVA, LA RELACIÓN ENTRE AMBAS TECNOLOGÍAS CAMINÓ EN FORMA CONJUNTA Y PARALELA.

Es así que hoy Java posee una infinidad de herramientas, recursos y opciones muy robustas para trabajar con XML, una tecnología tradicionalmente afianzada y relacionada al gigante de Sun. Quizás esto represente un problema para el desarrollador que llega al entorno y necesita un tiempo para decidirse por alguna de estas opciones. De todos modos, cuando lo hace, se encuentra con una gama de alternativas realmente interesantes.



El mismo depurador nos muestra cómo el parser NET analiza los datos.

JAXP

→ **J2EE** proporciona la librería **Java API for XML Parsing** (JAXP), la cual brinda soporte básico para acceder a documentación xml desde Java, así como también una interfaz proveedora de los servicios necesarios para parsearlos y transformarlos.

→ **JAXP** es una librería bastante flexible que nos permite utilizar cualquier analizador compatible con XML, desde dentro de nuestra propia aplicación. Asimismo, incluye los dos analizadores estándar para **SAX** y **DOM**.
Como ya vieron, esto nos permite elegir la realización del análisis de nuestros datos de dos maneras alternativas:

- Como streams de eventos (con Sax).
- Como un árbol de objetos con datos (con DOM).

Esta funcionalidad se realiza a través de la capa de conectividad, la cual permite conectar una implementación de las APIs antes nombradas. A través de dicha capa, JAXP soporta XSLT y también proporciona el soporte necesario para namespaces.
Esta iniciativa de Sun nace como un intento por uniformar el desarrollo de todas las aplicaciones Java con XML en un entorno único.
Hay que destacar que JAXP no es un analizador o parser, ni funciona como ellos, sino que trabaja junto con distintas implementaciones de "parsers".
El paquete javax.xml.parsers contiene dos grupos de clases compatibles con SAX y DOM: **SAXParserFactory** (definidas por el grupo XML-DEV) y **DocumentBuilderFactory** (definidas por W3C), respectivamente.

JDOM

Nacida como una alternativa más específica y adaptada al entorno Java, encontramos a JDOM. En el análisis de documentación xml, ésta trabaja junto con otros parsers externos, por ejemplo **XERCES**. En JDOM se facilita también la creación y la lectura de documentos, ya que no usaremos factorías ni otro tipo de modelos.
Antes de ver un ejemplo de JDOM, realicemos un rápido paso a paso por el proceso de instalación, que si bien es sencillo, quizá no está bien documentado y puede traer algunos dolores de cabeza.

→ 1) Descompactamos el archivo comprimido y creamos una carpeta JDOM en nuestro raíz.

→ 2) Configuramos las distintas variables globales (esto puede cambiar según el sistema operativo Windows).

→ 3) Configuramos el JAVA_HOME donde esté nuestro directorio Java SDK

```
C:\> set JAVA_HOME=C:\jdk1.2.2
```

En Linux esto podría ser de la siguiente manera:

```
JAVA_HOME=/usr/java; export JAVA_HOME
```

→ 4) Luego corremos un archivo batch (bat o sh, según el SO) de instalación desde nuestra carpeta JDOM, de este modo:

```
C:\jdom\build
```

En Linux sería así:

```
$ ./build.sh
```

→ 5) Para terminar, seteamos las variables que nos resta configurar, es decir, CLASSPATH y JDOM_HOME. En ambos casos los valores serán:

```
JDOM_HOME=C:\java\jdom-b7
CLASSPATH=%CLASSPATH%;%JDOM_HOME%\build\classes;%JAVA_HOME%
\lib\tools.jar; %JDOM_HOME%\lib\xerces.jar;%JDOM_HOME%\lib\jaxp.jar
```


Ahora podremos probar un ejemplo que permita mostrar por consola un archivo xml dado. Hay que recordar un punto importante: este archivo debe estar bien validado; de lo contrario, nos mostrará un error por pantalla. Esto no significa que hayamos configurado algo incorrectamente, sino que el parser no convalida el proceso.
En nuestro caso, tendremos un archivo xml llamado javacode.xml:

```
<?xml version="1.0" encoding="ISO8859-1"?>
<agenda xmlns="http://tempuri.org/ejemplo.xsd">
    <contacto indice="2">
        <nombre>Carlos</nombre>
        <apodo>Carli</apodo>
        <hobby>Cine, Atletismo</hobby>
    </contacto>
    <contacto indice="3">
        <nombre>Nora</nombre>
        <apodo>Nori</apodo>
        <hobby>Cine, Atletismo</hobby>
    </contacto>
    <contacto indice="1">
        <nombre>Virginia</nombre>
        <apodo>virgi</apodo>
        <hobby>Cine, literatura y comics</hobby>
    </contacto>
</agenda>
```

Con un esquema xml validado convenientemente:

```
<?xml version="1.0"?>
<xs:schema id="agenda" targetNamespace="http://tempuri.org/ejemplo.xsd" xmlns:mstns="http://tempuri.org/ejemplo.xsd" xmlns="http://tempuri.org/ejemplo.xsd" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:msdata="urn:schemas-microsoft-com:xml-msdata" attributeFormDefault="qualified" elementFormDefault="qualified">
  <xs:element name="agenda" msdata:IsDataSet="true" msdata:Locale="es-AR" msdata:EnforceConstraints="False">
    <xs:complexType>
      <xs:choice maxOccurs="unbounded">
        <xs:element name="contacto">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="nombre" type="xs:string" minOccurs="0" msdata:Ordinal="0" />
              <xs:element name="nacionalidad" type="xs:string" minOccurs="0" msdata:Ordinal="1" />
              <xs:element name="hobby" type="xs:string" minOccurs="0" msdata:Ordinal="2" />
            </xs:sequence>
            <xs:attribute name="indice" form="unqualified" type="xs:string" />
          </xs:complexType>
        </xs:element>
      </xs:choice>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Creamos la clase EjemploJDOM.java y, primero, cargamos las librerías necesarias:

```
import java.io.*;
import org.jdom.*;
import org.jdom.input.*;
import org.jdom.output.*;
```

La estructura del procesamiento de codificación sería así:

→ **a)** Definimos el argumento para correr el programa.

→ **b)** Creamos la instancia del analizador SAX sobre Xerces.

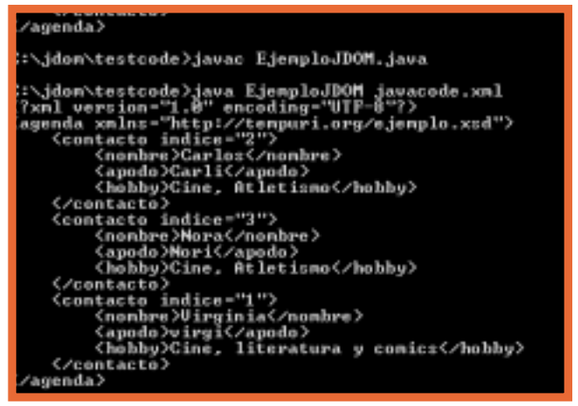
→ **c)** Instanciamos la variable del documento contenedor genérico JDOM y le pasamos como argumento la variable antes creada, apuntando al archivo xml a procesar (el cual puede estar en nuestro disco o vía web).

→ **d)** Creamos la instancia de salida, usando un formato estándar, y configuramos el control de errores.

```
public class EjemploJDOM{
  public static void main(String[] args)
{
    String archivo = args[0];
  try{
      SAXBuilder analizadorSAX = new SAX
Builder();
      Document documento = analizadorSAX.build
(new File(archivo));
      XMLOutputter salida = new XMLOutputter();
      salida.output(documento, System.out);
    } catch (Exception e) {
      e.printStackTrace();
    }
}
}
```

Luego, compilamos el archivo y ejecutamos:

```
Javac EjemploJDOM.java
Java EjemploJDOM javacode.xml
```



Como resultado, deberíamos obtener una pantalla similar a ésta.

PHP y XML

La plataforma XML de PHP no sólo ha sido mejorada y ampliada, sino que, fue rescrita a fin de optimizar su uso.
El procesamiento XML en PHP se sostiene, básicamente, sobre el paquete libxml2, y está contenido en las librerías libxml2, xmllint y xmlcatalog, muy útiles a la hora de analizar ficheros XML. Estas librerías permiten manejar tanto informes como catálogos xml con suma facilidad, y manejan en forma simultánea las extensiones de DOM, Sax y SimpleXML, con lo que es posible efectuar operaciones entre ellas sin afectar la perfomance general.
Además de esas tres librerías importantes, en este momento también podemos citar las extensiones de XPath y XSL que, al igual que la SOAP, se encuentran en fase experimental.

DOM: Si bien antes seguía en parte el estándar general, hoy el soporte es total y, junto con SimpleXML, son las librerías más utilizadas en forma profesional por los desarrolladores PHP. Fue mejorada y fusionada dentro de la librería general libxml2.
SAX: hasta PHP 4, era la principal extensión XML. Si bien es óptima (en términos de perfomance frente a DOM), requiere de una frecuente codificación más extensa.
SimpleXML: la nueva extensión de PHP 5 hace más sencillo y rápido todo el procesamiento de acceso de archivos xml, permitiendo manipular documentos xml como simples objetos.
También maneja análisis de RSS, configuración de datos, procesamiento con REST, etc. Ahorra, muchos de los llamados requeridos por las APIs DOM y SAX.
Primero realizaremos un ejemplo con DOM utilizando orientación a objetos, ingresando un elemento por pantalla en un formulario y mostrando, a la vez, que lo creamos como archivo.
La estructura general estará incluida en un condicional, que verificará el traspaso del formulario al bloque de código de procesamiento DOM, verificándose a través del botón de envío:

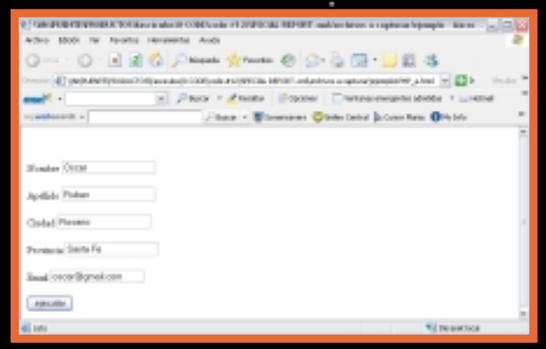
```
if ($enviar==""){
//formulario
}else{
//bloque de procesmamiento DOM
}
```

Dentro del formulario, creamos otro simple con cinco campos, que se envían a la misma página (formularioDOM.xml):

```
<form name="form1" method="post" action="Analizador
DOM.php">
  <p> </p>
  <p>Nombre
    <input type="text" name="xml_nombre">
  </p>
  <p>Apellido
    <input type="text" name="xml_apellido">
  </p>
  <p>Ciudad
    <input type="text" name="xml_ciudad">
  </p>
  <p>Provincia
    <input type="text" name="xml_provincia">
  </p>
  <p>Email
    <input type="text" name="xml_email">
</p>
  <p>
   <input type="submit" name="enviar" value="ejecutar">
</p>
</form>
```

En el otro bloque estará el sector a analizar. Primero, traemos los valores de los campos y los cargamos en variables:

```
$nombre= $_POST['xml_nombre'];
$apellido=$_POST['xml_apellido'];
$ciudad=$_POST['xml_ciudad'];
$provincia=$_POST['xml_provincia'];
$email=$_POST['xml_email'];
```



En nuestra prueba utilizamos el parser de Internet Explorer.

1967/ Necesidad de separar, dentro de un documento, la información de su formato.

1969/ Nace GML (Generalized Markup LanguageL).

1978/ Primeros pasos en la creación de SGML.

1985/ Un borrador de SGML es adoptado como norma ISO 8879:1986.

1990/ Nace la World Wide Web.

1996/ Nace XML.

1997/ Primera conferencia sobre XML. Nacen XLL, MathXL 1.0, DOM y SMIL.

1998/ Se lanza XML 1.0. Surgen Namespace, XSL, XHTML y DOM 2.

1999/ Nacen XPointer, Xlink, XPATH, XSLT, XML Schema, SVG, RDF, RDF Schema y XML Fragmentation.

2000/ Aparece DOM3, XForms y el protocolo XP.

2001/ Surge X-Schema. Se actualiza XHTML (V1.1). Nacen Smil, X-Link, xBase y SVG 1.0.

2002/ Surge una recomendación candidata para XML v1.1, XML Encryption, Decryption, y es propuesta como recomendación. SOAP 1.2. XPointer se convierte en estándar. Se publican guías de ayuda para desarrolladores XML.

2003/ Surgen XForms 1.0 y MathML 2.0 como recomendaciones.

2004/ Salen XML 1.1 y Namespace 1.1. Se aprueba como estándar XInclude. SSML permite generar voz sintética controlando la pronunciación. Surgen recomendaciones de la W3C para VoiceXML 2.0.

[special report]

Hasta aquí, tenemos un simple procesamiento de formularios en PHP. Instanciamos un documento de tipo DOM, pasándole al constructor el tipo de versión y codificación del xml:

```
$dom = new DOMDocument('1.0', 'utf-8');
```

Luego, empezamos a crear e instanciar nodos, cargándolos en variables. En este caso, son Agenda y Contacto, utilizando el método **appendchild()** de la clase principal DOM:

```
$Agenda = $DocumentoDOM->appendChild(new DOMElement
('agenda'));
$Contacto = $Agenda->appendChild(new DOMElement
('contacto'));
```

Una vez instanciados los nodos, nos abocamos a los elementos y atributos. Es así que creamos un nuevo atributo, configurando su nombre ("Indice") y su valor (1):

```
$Contacto->appendChild(new DOMAttr('indice', 1));
```

Completamos todos los elementos con **DOMElement** (nombre de elemento, variable):

```
$Contacto->appendChild(new DOMElement('nombre',
$nombre));
$Contacto->appendChild(new DOMElement('apellido',
$apellido));
$Contacto->appendChild(new DOMElement('ciudad', $ciudad));
$Contacto->appendChild(new DOMElement('provincia',
$provincia));
$Contacto->appendChild(new DOMElement('email', $email));
```

Luego, habilitamos la indentación mediante la propiedad formatOutput. Cuando ejecutemos, podremos visualizar la estructura XML creada mientras salvamos el código, automáticamente, en un archivo llamado ejemploDOM.xml:

```
$DocumentoDOM->formatOutput = true;
Echo($DocumentoDOM->saveXML( ));
$DocumentoDOM->save('ejemplo_dom.xml');
```

En el siguiente caso, nos concentraremos brevemente en la librería **SimpleXML**, la cual es soportada en dos de sus funciones, que son muy destacadas: **simplexml_load_string()** y **simplxml_load_file()**. Como verán en los ejemplos, los nodos y los elementos se tratan directamente como objetos php , sin ni siquiera declararlos de una forma tradicional (sólo la raíz principal). Los ejemplos demuestran el gran poder que tiene esta funcionalidad cuando deseamos manipular datos con muy pocas líneas.
Con SimpleXML podemos elegir la modalidad de carga del código xml, es decir, cargarlo desde una cadena o desde un archivo. En el primer caso, usaríamos la función simplexml_load_string() de la siguiente manera:

```
$cadena = <<<XML
<?xml version='1.0'?>
<lista>
<familiar>
<nombre>Norma</nombre>
<parentesco>Tia politica</parentesco>
</lista>
</familiar>XML;
```

Así, cargaríamos la cadena en la función:

```
$lista = simplexml_load_string($cadena) or die ("No se
cargo
cadena xml");
```

O elegimos entre una estructura con la función var_dump():

```
var_dump($lista);
```

o con un elemento de un nodo:

```
echo("El nombre es:". $lista->familiar->nombre);
```

La otra forma, más habitual, es sacando la información desde un archivo. En este caso, una lista de nombres, que denominamos lista.xml:

```
<?xml version="1.0" encoding="ISO8859-1" ?>
<lista>
<familiar>
<nombre>Carlos Alberto</nombre>
<parentesco>Suegro</parentesco>
</familiar>
<familiar>
<nombre>Marta Beatriz</nombre>
<parentesco>Suegra</parentesco>
</familiar>
</lista>
```

La extracción se haría de manera similar a la anterior, pero usando la función simplexml_load_file('archivo.xml'):

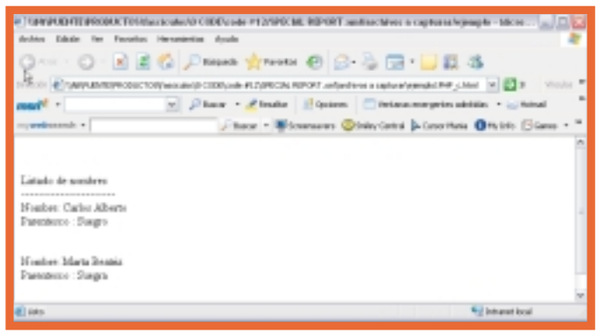
```
$lista = simplexml_load_file('lista.xml') or die ("No se
cargo archivo xml");
```

Mediante una simple iteración con foreach(), lograríamos ir cargando los nodos en forma progresiva:

```
echo ("<br>Listado de nombres");
echo("<br>--------------------");
foreach($lista->familiar as $familiar) {
       // loop through the nombres
       foreach($familiar->nombre as $nombre) {
           print  "<br>Nombre: " . $nombre. "<br>" ;
        print  "Parentesco: " . $familiar->parentesco.
"<br>" ;
       }
}
?>
```

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <agenda>
3  <contacto indice="5">
4  <nombre>Oscar</nombre>
5  <apellido>Ruben</apellido>
6  <ciudad>Rosario</ciudad>
7  <provincia>Santa Fe</provincia>
8  <email>oscar@gmail.com</email>
9  </contacto>
10 </agenda>
```

El IE nos presenta el ejemplo en forma estructurada y amena.



Visualizaremos reportes de datos configurados a nuestro gusto.