Introducción a XML

Publicado originalmente en inglés por IBM developerWorks
http://www.ibm.com/developerWorks

Traducido por Gratiniano Lozano para
http://www.javahispano.org

Tabla de contenidos

Si está viendo este documento online, puede pulsar sobre los tópicos de abajo para ver cada sección.

Sección 1. Acerca de este tutorial
¿Debería realizar este tutorial?

Este tutorial, recientemente revisado, discute lo que es XML, por qué fue desarrollado y como está definiendo el futuro del comercio electrónico. Llendo más lejos, echa un vistazo a diversos estándares XML e interfaces de programación, muestra cómo se puede empezar con esta tecnología y describe cómo un par de compañías han construido soluciones basadas en XML para simplificar y agilizar sus negocios.

En este tutorial, usted aprenderá:

- Porqué fue creado XML.
- Las reglas de los documentos XML.
- Cómo definir lo que puede y no puede contener un documento XML.
- Programación de interfaces que trabajen con documentos XML.
- Cuales son los principales estándares de XML y como trabajan juntos.
- Como usan XML las compañías en el mundo real.

Acerca del autor

Doug Tidwell es Programador Senior y 'evangelista' de IBM para los Servicios Web. Fue conferenciante en la primera conferencia sobre XML en 1997 y ha estado trabajando con lenguajes de marcado durante más de una década. Obtuvo una licenciatura en Ingles en la Universidad de Georgia y un Master en Informática en la Universidad de Vanderbilt. Se le puede encontrar en *dtidwell@us.ibm.co*m. También podrá ver su página web en *ibm.com/developerWorks/speakers/dtidwell/*.

Sección 2. ¿Qué es XML?
Introducción

XML, o Lenguaje de Marcado Extensible (Extensible Markup Language), es un lenguaje de marcado que puede usar para crear sus propias etiquetas. Fue creado por el Consorcio para la World Wide Web (W3C) para superar las limitaciones de HTML, el Lenguaje de Marcado de HiperTexto que es la base para todas las páginas Web. Aunque SGML ha sido usado durante décadas en la industria para publicación, su complejidad ha intimidado a mucha gente que, en otro caso, podría haberlo usado (SGML también vale como Suena Genial, Mejor Luego) (*N. del T.- interpretación libre de Sounds Great, Maybe Later). XML fue diseñado pensando en la Web.

¿Porqué necesitamos XML?

HTML es el lenguaje de marcado con más éxito del momento. Puede ver las etiquetas HMTL más simples en casi cualquier dispositivo, desde palmtops hasta los mainframes, e incluso puede convertir HTML en voz o en otros formatos con las herramientas adecuadas. Dado el éxito de HTML, ¿porqué la W3C creó XML?. Para responder a esta pregunta eche un vistazo a este documento.

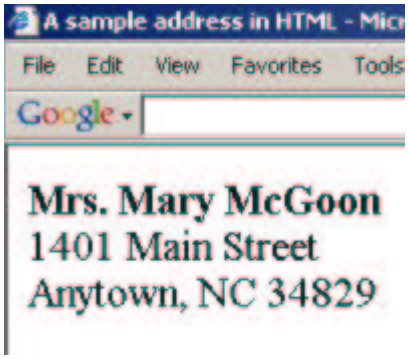
```
<p><b>Mrs. Mary McGoon</b>
<br>
1401 Main Street
<br>
Anytown, NC 34829</p>
```

El problema con HTML es que fue diseñado pensando en los humanos. Incluso sin ver el documento anterior en un navegador, podemos imaginar que se trata de la dirección postal de alguien. (Específicamente, es una dirección postal de alguien en los Estados Unidos; incluso aunque no estuviese familiarizado con los componentes de las direcciones postales de Estados Unidos podría imaginarse lo que representa.)

Como humanos, tenemos la inteligencia para comprender el significado e intención de muchos documentos. Una máquina, desafortunadamente no puede hacerlo. Mientras que las etiquetas de este documento le dicen al navegador cómo debe mostrar la información, no le dicen **lo que la información es**. Nosotros sabemos que es una dirección, pero la máquina no.

Representar HTML

Para representar HTML, el navegador únicamente sigue las instrucciones en el documento HTML. La etiqueta de párrafo le dice al navegador que empiece la presentación en una nueva línea, típicamente con una línea en blanco antes, mientras que las dos etiquetas de ruptura le dicen al navegador que avance hasta la siguiente línea sin dejar líneas en blanco en medio. Aunque el navegador da un bonito formato al documento, la máquina aún no sabe que se trata de una dirección postal.



Procesamiento de HTML

Para continuar la discusión con el ejemplo del documento HTML, consideremos la tarea de extraer el código postal de la dirección. Aquí presentamos un (intencionadamente frágil) algoritmo para encontrar la marca HTML para el código postal:

Si se encuentra un párrafo con dos etiquetas <br>, el código postal es la segunda palabra después de la primera coma en la segunda etiqueta de ruptura.

Aunque este algoritmo funciona con el ejemplo, existen en el mundo una gran cantidad de direcciones perfectamente válidas para las cuales no puede funcionar. Incluso si se pudiera escribir un algoritmo que encontrase el código postal para cualquier dirección escrita en HTML, existe un gran número de párrafos con dos etiquetas de ruptura cuyo contenido no es una dirección postal. La escritura de un algoritmo que busque en cualquier párrafo HTML y encuentre cualquier código postal que contenga sería extremadamente difícil, sino imposible.

Un documento XML de ejemplo

Echemos un vistazo ahora a un documento XML de ejemplo. Con XML, se puede asignar algún significado a las etiquetas en el documento. Más importante aún, también resulta fácil para una maquina el procesar la información. Se puede extraer el código postal de un documento simplemente localizando el contenido rodeado por las etiquetas `<codigo-postal>` y `</codigo-postal>`, técnicamente conocido como el *elemento* `<codigo-postal>`.

```
<direccion>
  <nombre>
    <titulo>Mrs.</titulo>
    <nombre>
       Mary
   </nombre>
   <apellidos>
```

```
     McGoon
   </apellidos>
  </nombre>

  <calle>
    1401 Main Street
  </calle>
  <ciudad>Anytown</ciudad>
  <estado>NC</estado>
  <codigo-postal>
    34829
  </codigo-postal>
</direccion>
```

Etiquetas, elementos y atributos

Existen tres términos comúnmente usados para describir las partes de un documento XML: *etiquetas*, *elementos* y *atributos*. Aquí está un documento de ejemplo que ilustra estos términos:

```
<direccion>
  <nombre>
    <titulo>Mrs.</titulo>
    <nombre>
      Mary
    </nombre>
    <apellidos>
      McGoon
    </apellidos>
  </nombre>
  <calle>
    1401 Main Street
  </calle>
  <ciudad estado="NC">Anytown</ciudad>
  <codigo-postal>
     34829
   </codigo-postal>
</direccion>
```

- Una etiqueta es un texto entre el símbolo menor que (<) y el símbolo mayor que (>). Existen etiquetas de inicio (como `<nombre>`) y etiquetas de fin (como `</nombre>`).
- Un elemento consta de la etiqueta de inicio, la etiqueta de fin y de todo aquello que este entre ambas. En el ejemplo anterior, el elemento `<nombre>` contiene tres elementos hijos: `<titulo>`, `<nombre>`, y `<apellidos>`.
- Un atributo es un par nombre-valor dentro de la etiqueta de inicio de un elemento. En este ejemplo, `estado` es un atributo del elemento `<ciudad>`, en ejemplos anteriores `<estado>` era un elemento (ver Un documento XML de ejemplo en la página 4).

Cómo está cambiando XML la Web

Ahora que ha visto como los desarrolladores pueden usar XML para crear documentos con datos que se auto-describen, veamos cómo se están usando esos documentos para aumentar el rendimiento de la Web. Aquí mostramos algunas áreas:

- **XML simplifica el intercambio de datos.** Debido a que diferentes organizaciones (en incluso diferentes partes dentro de una misma organización) raramente utilizan un único conjunto de herramientas, esto puede suponer una cantidad de trabajo significativa para comunicar las aplicaciones. Usando XML cada grupo crea una única utilidad que transforma sus formatos de datos internos en XML y viceversa. Lo mejor de todo, hay una gran probabilidad de que sus suministradores de software, ya proporcionen herramientas que transformen sus registros de base de datos(o directorios LDAP, o ordenes de compra, etc.) desde y hacia XML.
- **XML permite un código más ligero.** Debido a que los documentos XML pueden ser estructurados para identificar cada pieza importante de información (así como las relaciones entre dichas piezas), es posible escribir código que procese estos documentos XML sin intervención humana. El hecho de que los proveedores de software hayan gastado cantidades masivas de tiempo y dinero construyendo herramientas de desarrollo XML significa que escribir ese código es un proceso relativamente simple.
- **XML permite búsquedas más rápidas**. Aunque los motores de búsqueda han mejorado sustancialmente durante los años, es todavía bastante común encontrar resultados erróneos en una búsqueda. Si se están buscando páginas HTML de alguien llamado “Chip”, se pueden encontrar páginas sobre chips de chocolate, chips de ordenador, chips de madera y montones de otros enlaces inútiles. Buscando en documentos XML por elementos `<nombre>` que contengan el texto `Chip` debería darnos un conjunto de resultados mucho mejor.

Discutiremos casos reales del uso de XML en Casos de Estudio en la página 28.

Sección 3. Reglas de los documentos XML. Visión General

Si ha visto los documentos XML, estará familiarizado con los conceptos básicos del uso de etiquetas para marcar el texto de un documento. Esta sección discute las diferencias entre los documentos XML y los documentos XML. Va más allá de las reglas básicas de los documentos XML y discute la terminología usada para describirlos.

Un punto importante acerca de los documentos XML: **La especificación XML requiere un parser para rechazar los documentos XML que no sigan las reglas básicas**. Muchos analizadores HTML aceptarán marcados chapuceros haciendo conjeturas sobre lo que el escritor del documento intentaba. Para evitar el revoltijo de estructuras encontrado en la mayoría de los documentos HTML, los creadores de XML decidieron reforzar la estructura del documento desde el principio.

(De hecho, si usted no está familiarizado con el termino, un parser es una pieza de código que intenta leer un documento e interpretar su contenido.)

Documentos inválidos, válidos y bien formados

Hay tres tipos de documentos XML:

- **Documentos inválidos** no siguen las reglas de sintaxis definidas por la especificación XML. Si un desarrollador tiene reglas definidas de lo que ese documento puede contener en una DTD o Esquema, y el documento no las sigue, ese documento es inválido. (Ver Definiendo el contenido del documento en la página 13 para una introducción apropiada a los DTDs y Esquemas de los documentos XML.)
- **Documentos válidos** siguen tanto las reglas de sintaxis XML como las reglas definidas en su propio DTD o Esquema.
- **Documento bien formado** sigue las reglas de sintaxis XML, pero no tiene un Esquema o DTD.

El elemento raíz

Un documento XML debe estar contenido en un elemento único. Este elemento único es llamado el **elemento raíz** y contiene todo el texto y cualquier otro elemento en el documento. En el ejemplo siguiente, el documento XML está contenido en un elemento único, el elemento `<greeting>`. Notese que el documento tiene un comentario el cual, aunque está fuera del elemento raíz, es perfectamente legal

```
<?xml version="1.0"?>
<!—Un documento bién formado-->
```

```
<saludo>
  ¡Hola Mundo!
</saludo>
```

Aquí presentamos un documento que no contiene un único elemento raíz:

```
 <?xml version="1.0"?>
<!—Un documento no válido -->
<saludo>
  ¡Hola Mundo!
</saludo>
<saludo>
  ¡Hola, el Mundo!
</saludo>
```

Es necesario un parser XML para rechazar el documento, independientemente de la información que pueda contener.

Los elementos no pueden solaparse

Los elementos XML no pueden solaparse. Aquí presentamos un marcado ilegal:

```
 <!—marcado XML ilegal -->
<p>
<b>Yo <i>amo de verdad
</b> XML.
</i>
</p>
```

Si comienza un elemento `<i>` dentro de un elemento `<b>` también deberá terminarlo dentro. Si se quiere que el texto `XML` aparezca en cursiva, hay que añadir un segundo elemento `<i>` para corregir el marcado:

```
<!—marcado XML legal -->
<p>
<b>Yo <i>amo de verdad
</i></b>
<i>XML.</i>
</p>
```

Un parser XML aceptará solamente este marcado; Los parser HTML de la mayoría de los navegadores aceptarán ambos.

Las etiquetas de fin son obligatorias

No puede olvidarse de ninguna etiqueta de fin. En el primer ejemplo que sigue, el marcado no es legal debido a que no hay etiquetas de fin de párrafo (`</p>`). Aunque esto es aceptable en HTML (y, en algunos casos, en SGML), un parser XML lo rechazaría.

```
<!—Marcado XML NO LEGAL -->
<p>Yada yada yada...
```

```
<p>Yada yada yada...
<p>...
```

Si un elemento carece de contenido se le llama **elemento vacío;** los elementos HTML para ruptura (`<br>`) e imagen (`<img>`) son dos ejemplos. En los elementos vacíos en XML pueden ponerse lo barra de cierre en la etiqueta de inicio. Los dos elementos de ruptura y los dos elementos de imagen a continuación significan la misma cosa para un parser XML:

```
<!— Dos rupturas equivalentes -->
<br></br>
<br/>
<!— Dos elementos de imagen equivalentes -->
<img src="../img/c.gif"></img>
<img src="../img/c.gif" />
```

Los elementos son sensibles a mayúsculas

Los elementos XML son sensibles a mayúsculas. En HTML, <h1> y <H1> son lo mismo; en XML no. Si intenta terminar un elemento <h1> con una etiqueta </H1>, obtendrá un error. En el ejemplo siguiente, el primero presenta un encabezado ilegal, mientras que el último es correcto.

```
<!— marcado XML ILEGAL -->
<h1>Los elementos son sensibles a mayusculas</H1>

<!— marcado XML LEGAL -->
<h1> Los elementos son sensibles a mayusculas</h1>
```

Los atributos deben tener valores entrecomillados

Existen un par de reglas para los atributos en los documentos XML:

- Los atributos deben tener valores
- Esos valores deben estar entrecomillados.

Compare los dos ejemplos que vienen a continuación. El marcado del primero es legal en HTML, pero no en XML. Para obtener el equivalente en XML, se le debe dar un valor al atributo y debe entrecomillarse.

```
<!-- marcado XML ILEGAL -->
<ol compact>

<!-- marcado XML LEGAL -->
<ol compact="yes">
```

Puede usar comillas simples o dobles siempre y cuando sea consecuente. Si el valor del atributo contiene un single o un `double` puede usarse el otro tipo de comillas para rodear el valor (como en `nombre="Doug's car"`), o usar las entidades `&quot;` para comillas

dobles y ' para comillas simples. Una *entidad* es un símbolo, como `&quot;`, que el parser XML reemplazará con otro texto como “.

Declaraciones XML

Muchos documentos XML comienzan con una *declaración XML* que proporciona al parser información básica sobre el documento. Una declaración XML está recomendada, pero no es obligatoria. Si existe una, debe ser lo primero que aparezca en el documento.

La declaración puede contener hasta tres pares nombre-valor (mucha gente les llama atributos, aunque técnicamente no lo son). `version` es la versión de XML usada; actualmente este valor debe ser `1.0`. `encoding` es el conjunto de caracteres usado en el documento. El conjunto de caracteres `ISO-8859-1` referenciado en esta declaración incluye todos los caracteres usados en la mayoría de los lenguajes de Europa Occidental. Si no se especifica `encoding` el parser XML asume que los caracteres pertenecen al conjunto `UTF-8`, un estándar Unicode que soporta virtualmente cada carácter e ideograma de cualquier lenguaje del mundo.

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
```

Finalmente, `standalone,` que solo puede ser `yes` o `no`, define si este documento puede ser procesado sin leer otros ficheros. Por ejemplo, si el documento XML no referencia ningún otro fichero, puede especificarse `standalone="yes`". Si el documento XML referencia otros ficheros que describen lo que el documento puede contener (veremos más acerca de estos ficheros en un momento), se debe especificar `standalone="no`". Debido a que `standalone="no"` es la opción por defecto, raramente veremos `standalone` en las declaraciones XML.

Otras cosas en los documentos XML

Hay algunas otras cosas que se pueden encontrar en un documento XML:

- **Comentarios:** los comentarios pueden aparecer en cualquier lugar en el documento; incluso antes o después del elemento raíz. Un comentario comienza con `<!–`y termina con `–->`. Un comentario no puede contener un guión doble `(--)`, excepto al final; con esta excepción, un comentario puede contener cualquier cosa. Aún más importante, cualquier marca dentro de un comentario será ignorada; si quiere eliminar una gran sección de un documento XML simplemente introdúzcala en un comentario. (Para restaurar la sección comentada simplemente elimine las etiquetas de comentario.) Aquí presentamos un ejemplo que contiene un comentario

  ```
  <!–Esto es una PI para Cocoon: -->
  <?cocoon-process type="sql"?>
  ```

- **Instrucciones de procesamiento:** Una instrucción de procesamiento es una marca que se interpretará como un segmento de código. En el ejemplo previo hay una instrucción de

procesamiento (algunas veces llamadas PI) para Cocoon, un marco de procesado para XML de Apache Software Foundation. Cuando Cocoon está procesando un documento XML busca instrucciones de procesamiento que comiencen con `cocoon-process`, entonces procesa el documento XML de acuerdo a ellas. En este ejemplo, el atributo `type="sql"` le dice a Cocoon que el documento XML contiene una sentencia SQL.

```
<!-- He aquí una entidad: -->
<!ENTITY dw "developerWorks">
```

- **Entidades:** El ejemplo previo define una *entidad* para el documento. Dondequiera que el procesador XML encuentre la cadena `&dw;`, la reemplazará con la cadena `developerWorks`. La especificación XML también define cinco entidades que pueden usarse en lugar de varios caracteres especiales. Las entidades son:
  - `<` para el símbolo menor que (<)
  - `>` para el símbolo mayor que (>)
  - `&quot;` para las comillas dobles (“)
  - `&apos;` para la comilla simple o apostrofe (‘)
  - `&` para el ampersand (&).

Namespaces

El poder de XML proviene de su flexibilidad, del hecho de que usted, yo y millones de otras personas podamos definir nuestras propias etiquetas para describir nuestros datos. ¿Recuerda el ejemplo del documento XML con el nombre y dirección de una persona? Ese documento incluía el elemento `<titulo>` para el título o tratamiento de cortesía, una elección perfectamente razonable como nombre de un elemento. Si usted crea una librería online, podría elegir el crear un elemento `<titulo>` para almacenar el título de un libro. Si crea una compañía hipotecaria online, podría elegir crear un elemento `<titulo>` Para el título de una propiedad. Todas son elecciones razonables, pero todas ellas crean elementos con el mismo nombre. ¿Cómo saber si, dado un elemento `<titulo>` se refiere a una persona, un libro o una propiedad? Con los *namespace*s.

Para usar un namespace, debe definir un prefijo *namespace* y mapearlo a una cadena particular.

Así es cómo se deberían definir prefijos namespace para nuestros tres elementos `<titulo>`:

```
<?xml version="1.0"?>
<resumen_usuario
xmlns:direccion="http://www.xyz.com/direcciones/"
xmlns:libros="http://www.zyx.com/libros/"
xmlns:hipoteca="http://www.yyz.com/hipotecas/"
>
... <direccion:nombre><titulo>Mrs.</titulo> ... </direccion:nombre> ...
... <libros:titulo>El Señor de los Anillos</libros:titulo> ...
... <hipoteca:titulo>NC2948-388-1983</hipoteca:titulo> ...
```

En este ejemplo, los tres prefijos del namespace son `direccion, libros` e `hipoteca`.

Dese cuenta que la definición de un namespace para un elemento particular significa que todos los elementos hijos pertenecen al mismo namespace. El primer elemento `<titulo>` pertenece al namespace `direccion` debido a que su elemento padre `<direccion:Nombre>`, ya pertenecía a ese namespace.

El punto final: **La cadena de una definición de namespace es solo una cadena.** Si, esa cadena parece una URL, pero no lo es. Podría definir `xmlns:direccion="mike"` y funcionaría exactamente igual, Lo único que importa acerca de la cadena del namespace es que sea única; por esto la mayor parte de las definiciones de namespace parecen URLs. Los parser XML no van a la dirección `http://www.zyx.com/libros/` para buscar una DTD o un Esquema, simplemente usan esos textos como cadenas. Es confuso, pero así es como funcionan los namespaces.

Sección 4. Definición del contenido de un documento

Visión General

Durante este tutorial ha estado aprendiendo acerca de las reglas básicas de los documentos XML; eso está bien, pero ahora necesita definir los elementos que usará para representar datos. Aprenderá dos maneras de hacerlo en esta sección:

- Un método es usar un **Document Type Definition** o **DTD**. Un DTD define los elementos que pueden aparecer en un documento XML, el orden en el cual pueden aparecer, cómo pueden estar anidados y otros detalles básicos de la estructura del documento XML. Los DTD son parte de la especificación original de XML y son muy similares a los DTDs de SGML.
- El otro método es usar un **Esquema XML (XML Schema)**. Un esquema puede definir todas las estructuras de documento que pudieran definirse con DTD y además, puede definir tipos de datos y reglas mucho más complicadas de las que pueden hacerse con DTD. El W3C desarrollo la especificación de Esquemas XML un par de años después que la especificación original XML.

Document Type Definitions

Un DTD permite especificar la estructura básica de un documento XML. El siguiente par de ejemplos muestra lo que aparece cómo fragmentos de DTDs. El primero de ellos es una DTD que define la estructura básica del documento de dirección ejemplo en la sección, ¿Qué es XML? En la página 3:

```
<!-- direccion.dtd -->
<!ELEMENT direccion (nombre, calle, ciudad, estado, codigo-postal)>
<!ELEMENT nombre (titulo? nombre, apellidos)>
<!ELEMENT titulo (#PCDATA)>
<!ELEMENT nombre (#PCDATA)>
<!ELEMENT apellidos (#PCDATA)>
<!ELEMENT calle (#PCDATA)>
<!ELEMENT ciudad (#PCDATA)>
<!ELEMENT estado (#PCDATA)>
<!ELEMENT codigo-postal (#PCDATA)>
```

Este DTD define todos los elementos usados en el documento de ejemplo. Define tres cosas básicas:

- Un elemento `<direccion>` que contiene un `<nombre>`, un `<calle>`, un `<ciudad>`, un `<estado>`, y un `<codigo-postal>`. Todos estos elementos deben aparecer y deben hacerlo en ese mismo orden.

- Un elemento `<nombre>` contiene un elemento `<titulo> opcional` (la marca de pregunta indica que `titulo` es opcional), seguido de un elemento `<nombre>` y de un elemento `<apellidos>`.

Todos los demás elementos contienen texto. (`#PCDATA` indica datos de carácter; no pueden incluirse otros elementos dentro de estos.)

Aunque son muy simples, las DTD aclaran que combinaciones de elementos son legales. Una documento de dirección postal que presente un elemento `<codigo-postal>` antes de un elemento `<estado>` no es legal, y ningún documento que no presente el elemento `<apellidos>` lo será.

Además, **hay que darse cuenta de que la sintaxis DTD es diferente de la sintaxis XML ordinaria.** (Los documentos de Esquema XML, por el contrario, son XML ellos mismos, lo cual tiene algunas consecuencias interesantes.) A pesar de la sintaxis diferente de los DTDs se pueden seguir insertando comentarios ordinarios dentro de ellas.

Símbolos en los DTDs

Existen unos pocos símbolos usados en los DTDs para indicar cuantas veces (si es que lo hace) algo puede aparecer en un documento XML. Aquí mostramos algunos ejemplos junto a sus significados:

- `<!ELEMENT direccion (nombre, ciudad, estado)>`

El elemento `<direccion>` debe contener un elemento `<nombre>`, un `<ciudad>`, y un `<estado>` en ese orden. Todos los elementos son obligatorios. **La coma indica una lista de items.**

- `<!ELEMENT nombre (titulo?, nombre, apellidos)>`

Esto significa que el elemento `<nombre>` contiene un elemento `<titulo>` opcional, seguido obligatoriamente de un elemento `<nombre>` y de otro `<apellidos>`. **La marca interrogación, indica que un item es opcional; Puede o no aparecer.**

- `<!ELEMENT agenda (direccion+)>`

Un elemento `<agenda>` contiene uno o más elementos `<direccion>`. Se pueden tener tantos elementos `<direccion>` como se necesiten, pero, al menos, debe existir uno. **El signo mas (+) indica que un item debe aparecer al menos una vez, pero puede aparecer cualquier número de veces.**

- `<!ELEMENT direcciones-privadas (direccion*)>`

Un elemento `<direcciones-privadas>` contiene cero o más elementos `<direccion>`.

El asterisco indica que un item puede aparecer cualquier número de veces, incluyendo cero.

- `<!ELEMENT nombre (titulo?, nombre, (inicial-segundo-nombre | segundo-nombre)?, apellidos)>`

Un elemento `<nombre>` contiene un elemento `<titulo>` opcional seguido de un elemento `<nombre>`, posiblemente seguido de un `<inicial-segundo-nombre>` o un elemento `<segundo-nombre>`, seguido de un elemento `<apellidos>`. En otras palabras, tanto `<inicial-segundo-nombre>` como `<segundo-nombre>` son opcionales y solamente se puede tener uno de los dos. **Las barras verticales indican una lista de opciones; solo se puede escoger un item de la lista.** Hay que notar que en este ejemplo se han usado paréntesis para agrupar ciertos elementos, y una marca de interrogación que afecta a todo el grupo.

- `<!ELEMENT nombre ((titulo?, nombre, apellidos) | (apellido, nombre-madre, apodo))>`

El elemento `<nombre>` puede contener una o más secuencias: Un elemento `<titulo>` opcional, seguido de un elemento `<nombre>` y otro `<apellidos>`; o un `<apellido>`, un `<nombre-madre>`, y un `<apodo>`.

Una palabra acerca de la flexibilidad

Antes de seguir, introduciremos una breve nota acerca del diseño de DTDs de cara a la flexibilidad. Considerando el ejemplo de dirección postal; lo hemos escrito claramente con la codificación postal de Estados Unidos en la mente. Si se quiere un DTD o esquema que defina reglas para otros tipos de direcciones, debería de añadir mucha más complejidad. El requerir un elemento `<estado>` puede tener sentido en Australia, pero no lo tendrá en Reino Unido. Una dirección canadiense podría ser manejada por el DTD de ejemplo de Document Type Definitions en la página 13, pero añadir un elemento `<provincia>` parece una idea mejor. Finalmente, hay que tener en cuenta que en muchos lugares del mundo, conceptos tales como título, primer nombre y segundo nombre no tienen sentido.

Como punto final: si esta definiendo la estructura de un documento XML, debería ser tan prevenido en su DTD o esquema como si estuviese diseñando el esquema de la base de datos o la estructura de datos en una aplicación. Cuanto más requerimientos de futuro pueda prevenir, tanto más fácil y más barato le será el implementarlos más tarde.

Definición de atributos

Este tutorial introductorio no puede profundizar demasiado en cómo trabajan los DTDS, pero hay un tópico básico que sí cubriremos aquí: la definición de atributos. Se pueden definir atributos para los elementos que aparecen en su documento XML. Usando una DTD también se puede:

- Definir qué atributos son obligatorios.
- Definir valores por defecto para los atributos
- Proporcionar una lista con los valores válidos para un atributo.

Supongamos que se quiere cambiar la DTD para hacer que `estado` sea un atributo del elemento `<ciudad>`. Se haría así:

```
<!ELEMENT ciudad (#PCDATA)>
<!ATTLIST ciudad estado CDATA #REQUIRED>
```

Esto define el elemento `<ciudad>` como antes, pero el ejemplo revisado también usa una declaración `ATTLIST` para enumerar los atributos del elemento. El nombre `ciudad` dentro de la enumeración de atributos le dice al parser que esos atributos están definidos para el elemento `<ciudad>`. El nombre `estado` es el nombre del atributo y las palabras reservadas `CDATA` y `#REQUIRED` le dicen al parser que el atributo `estado` contiene texto y es obligatorio (si fuese opcional, se indicaría con `CDATA #IMPLIED`).

Para definir múltiples atributos para un elemento, se escribe un `ATTLIST` como este:

```
<!ELEMENT ciudad (#PCDATA)>
<!ATTLIST ciudad estado CDATA #REQUIRED
      codigo-postal CDATA #REQUIRED>
```

Este ejemplo define `estado` y `codigo-postal` como atributos del elemento `<ciudad>`.

Finalmente, las DTDS permiten definir valores por defecto para los atributos y enumerar los valores válidos para un atributo:

```
<!ELEMENT ciudad (#PCDATA)>
<!ATTLIST ciudad estado CDATA (AZ|CA|NV|OR|UT|WA) "CA">
```

Este ejemplo indica que solo se soportan direcciones de los estados de Arizona (AZ), California (CA), Nevada (NV), Oregon (OR), Utah (UT), y Washington (WA), y que el valor por defecto es California. De esta manera se puede realizar una muy limitada forma de validación de los datos. Aunque es una función útil, es solo una pequeña parte de lo que se puede hacer con los esquemas XML (ver Esquemas XML en la página 16).

Esquemas XML

Con los esquemas XML se tiene un mayor poder para definir lo que parece un documento XML válido. Presentan varias ventajas sobre los DTDs:

- **Los esquemas usan sintaxis XML.** En otras palabras, un esquema XML es un documento XML. Esto significa que se puede procesar un esquema igual que cualquier otro documento. Por ejemplo, se puede escribir una hoja de estilo XSLT que convierta un esquema XML en un formulario Web completo con código JavaScript generado automáticamente que valide los datos conforme se vayan introduciendo.

- **Los esquemas XML soportan tipos de datos.** Mientras que los DTDs no soportan tipos de datos, esta claro que esos tipos de datos fueron desarrollados con una perspectiva de publicación. Los esquemas XML soportan todos los tipos originales de los DTDs (cosas como Ids y referencias ID). También soportan enteros, números en punto flotante, fechas, horas, cadenas de texto, URLs y otros tipos de datos útiles para el procesado y validación de datos.
- **Los esquemas XML son extensibles.** Además de los tipos de datos definidos en la especificación de esquemas XML, se pueden crear tipos de datos propios y se pueden derivar nuevos tipos de datos a partir de otros.
- **Los esquemas XML tienen mayor poder de expresión.** Por ejemplo, con esquemas XML se puede definir que el valor del un atributo <estado> no puede tener una longitud mayor de 2 caracteres, o que el valor de un elemento <codigo-postal> debe cumplir la expresión regular `[0-9]{5}(-[0-9]{4})?`. No se puede hacer ninguna de estas cosas con los DTDs.

Un esquema XML de ejemplo

A continuación presentamos un esquema XML que coincide con nuestra DTD original de nombre y dirección postal. Añado dos restricciones: el valor del elemento `<estado>` debe tener exactamente dos caracteres de longitud y el valor del elemento `<codigo-postal>` debe cumplir con la expresión regular `[0-9]{5}(-[0-9]{4})?`. Aunque el esquema es mucho mayor que la DTD, expresa de manera más clara qué documentos serán válidos. Este es el esquema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <xsd:element name="direccion">
            <xsd:complexType>
                  <xsd:sequence>
                        <xsd:element ref="nombre"/>
                        <xsd:element ref="calle"/>
                        <xsd:element ref="ciudad"/>
                        <xsd:element ref="estado"/>
                        <xsd:element ref="codigo-postal"/>
                  </xsd:sequence>
            </xsd:complexType>
      </xsd:element>

      <xsd:element name="nombre">
            <xsd:complexType>
                  <xsd:sequence>
                        <xsd:element ref="titulo" minOccurs="0"/>
                        <xsd:element ref="nombre"/>
                        <xsd:element ref="apellidos"/>
                  </xsd:sequence>
            </xsd:complexType>
      </xsd:element>

      <xsd:element name="titulo"         type="xsd:string"/>
      <xsd:element name ="nombre"  type="xsd:string"/>
      <xsd:element name ="apellidos"     type="xsd:string"/>
      <xsd:element name ="calle"         type="xsd:string"/>
      <xsd:element name ="ciudad"        type="xsd:string"/>


      <xsd:element name ="estado">
```

```
            <xsd:simpleType>
                  <xsd:restriction base="xsd:string">
                        <xsd:length value="2"/>
                  </xsd:restriction>
            </xsd:simpleType>
      </xsd:element>

      <xsd:element name ="codigo-postal">
            <xsd:simpleType>
                  <xsd:restriction base="xsd:string">
                        <xsd:pattern value="[0-9]{5}(-[0-9]{4})?"/>
                  </xsd:restriction>
            </xsd:simpleType>
      </xsd:element>

</xsd:schema>
```

Definición de elementos en los esquemas

El esquema XML presentado en Un esquema XML de ejemplo en la página 17, define varios elementos XML con el elemento `<xsd:element>`. Los primeros dos elementos definidos, `<direccion>` y `<nombre>`, estan compuestos por otros elementos. El elemento `<xsd:sequence>` define la secuencia de elementos contenidos en cada uno de ellos. A continuación mostramos un ejemplo:

```
<xsd:element name="direccion">
      <xsd:complexType>
            <xsd:sequence>
                  <xsd:element ref="nombre"/>
                  <xsd:element ref="calle"/>
                  <xsd:element ref="ciudad"/>
                  <xsd:element ref="estado"/>
                  <xsd:element ref="codigo-postal"/>
            </xsd:sequence>
      </xsd:complexType>
</xsd:element>
```

Como en la versión DTD, el esquema XML de ejemplo define que <direccion> contiene un elemento `<nombre>`, un `<calle>`, un `<ciudad>`, un `<estado>`, y un `<codigo-postal>`, en ese orden. Se debe notar que el esquema define un nuevo tipo de datos con el elemento `<xsd:complexType>`.

Definir la mayor parte de los elementos que contienen texto es sencillo. Unicamente se declara el nuevo elemento y se le da un tipo de datos `xsd:string`:

```
<xsd:element name="titulo"          type="xsd:string"/>
<xsd:element name="nombre"    type="xsd:string"/>
<xsd:element name="apellidos"       type="xsd:string"/>
<xsd:element name="calle"           type="xsd:string"/>
<xsd:element name="ciudad"          type="xsd:string"/>
```

Definición del contenido de los elementos en los esquemas

El esquema de ejemplo define restricciones para el contenido de dos elementos: el contenido del elemento `<estado>` debe tener una longitud de dos caracteres, y el contenido de un elemento `<codigo-postal>` debe cumplir la expresión regular `[0-9]{5}(-[0-9]{4})?`. Así es como se hace:

```
<xsd:element name="estado">
      <xsd:simpleType>
            <xsd:restriction base="xsd:string">
            <xsd:length value="2"/>
      </xsd:restriction>
      </xsd:simpleType>
</xsd:element>

<xsd:element name="codigo-postal">
      <xsd:simpleType>
            <xsd:restriction base="xsd:string">
                  <xsd:pattern value="[0-9]{5}(-[0-9]{4})?"/>
            </xsd:restriction>
      </xsd:simpleType>
</xsd:element>
```

Para los elementos `<estado>` y `<codigo-postal>`, el esquema define nuevos tipos de datos con restricciones. En el primer caso usa el elemento `<xsd:length>` y en el segundo usa `<xsd:pattern>` para definir una expresión regular que ese elemento debe cumplir.

Este resumen solo araña la superficie de lo que los esquemas XML pueden hacer; hay libros enteros dedicados a este asunto. Para el propósito de esta introducción basta con decir que los esquemas XML son una manera muy potente y flexible de describir lo que un documento XML debe parecer.

Sección 5. Interfaces de programación para XML

Visión General

Esta sección echa un vistazo a la gran variedad de interfaces de programación para XML. Estas interfaces proporcionan a los desarrolladores una forma consistente de trabajar con los documentos XML. Hay muchos APIs disponibles; Esta sección se centra en cuatro de la más populares y útiles: el Document Object Model (DOM), el Simple API for XML (SAX), JDOM, y el Java API for XML Parsing(JAXP) . Se puede encontrar mucha más información acerca de estos APIs a través de los enlaces en Recursos en la página 32).

Document Object Model

El Document Object Model, comúnmente llamado DOM, define un conjunto de interfaces a la versión parseada de un documento XML. El parser lee el documento completo y construye un árbol en la memoria, de forma que el código puede entonces utilizar las interfaces DOM para manipular ese árbol. Es posible moverse a través del árbol para ver lo que el documento original contenía, se pueden borrar secciones del árbol, reordenarlo, añadir nuevas ramas, etc.

DOM fue creado por el W3C, y es una Recomendación Oficial del consorcio.

Limitaciones de DOM

DOM proporciona un rico conjunto de funciones que pueden usarse para interpretar y manipular un documento XML, pero esas funciones tienen un precio. Tan pronto como el DOM para XML original empezó a desarrollarse, mucha gente en la lista de correos de XML-DEV empezó a preocuparse acerca de lo siguiente:

- DOM construye un árbol en la memoria con el documento entero. Si el documento es muy grande ser requiere una cantidad significativa de memoria.
- DOM crea objetos que los representan todo en el documento original, incluyendo elementos, texto, atributos y espacios en blanco. Si solo se tiene interés en una pequeña parte del documento original, es extremadamente costoso crear todos esos objetos que nunca se usarán.
- Un parser DOM tiene que leer el documento entero antes de que el código pueda tomar el control. Para documentos muy grandes esto puede causar un retraso significativo.

Estas son algunas de las debilidades derivadas del diseño del Document Object Model; independientemente de estas, **el API DOM es una forma muy útil de parsear documentos XML.**

Simple API for XML

Para soslayar las carencias de DOM, los participantes de XML-DEV (liderados por David Megginson) crearon la interfaz SAX. SAX tiene diversas características que solucionan las limitaciones de DOM.

- Un parser SAX envía eventos al código. El parser le dice cuando ha encontrado el comienzo y/o fín del documento, elemento, texto, etc. Usted decide que eventos son importantes y que tipo de estructura de datos desea crear para almacenarlos. Si no salva explícitamente los datos de un evento, se perderán.
- Un parser SAX no crea ningún objeto en absoluto, simplemente envía eventos hacía su aplicación. Si desea crear objetos basados en esos eventos, es cosa suya.
- Un parser SAX empieza a enviar eventos tan pronto como se inicia. Su código recibirá un evento cuando el parser encuentre el inicio del documento, cuando encuentre el inicio de un elemento, texto, etc. Su aplicación comenzará a generar resultados inmediatamente, sin tener que esperar hasta que el documento entero haya sido parseado. Aún mejor, si solo está buscando algunas cosas en el documento, su código puede elevar una excepción sólo si ha encontrado lo que estaba buscando. Una excepción detiene el parser SAX y su código entonces puede hacer cualquier cosa que necesite hacer con los datos encontrados.

Dicho esto, tanto SAX como DOM tienen su lugar respectivo. El resto de esta sección discute el porqué usted querría usar una u otra interfaz.

Limitaciones de SAX

Para ser sinceros, los parsers SAX tienen limitaciones que pueden causar preocupación:

- Los eventos SAX no tienen estado. Cuando un parser SAX encuentra texto en un documento XML, envía un evento a su código. Este evento solo le indica el texto encontrado; no le dice que elemento contenía ese texto. Si quiere saberlo, tiene que escribir el código para la gestión de estado usted mismo.
- Los eventos SAX no son permanentes. Si su aplicación necesita una estructura de datos que modele el documento XML, debe escribir ese código usted mismo. Si necesita acceder a los datos de un evento SAX y no los ha almacenado en su código, tiene que parsear el documento de nuevo.
- SAX no está controlado por una organización centralizada. Aunque esto **no ha causado problemas hasta hoy**, algunos desarrolladores se sentirían más cómodos si SAX estuviese controlado por una organización como la W3C.

JDOM

Frustrados por la dificultad en realizar ciertas tareas con los modelos DOM y SAX, Jasón Hunter y Brett McLaughlin crearon el paquete JDOM. JDOM es un proyecto open-source de tecnología basada en Java que intenta seguir la regla 80/20: Desarrollar lo que el 80% de los usuarios necesitan con el 20% de las funciones en DOM y SAX. JDOM trabaja con parsers SAX y DOM, así que está implementado como un conjunto relativamente pequeño de clases Java.

La principal ventaja de JDOM es que reduce enormemente la cantidad de código que se debe escribir. Aunque este tutorial introductorio no discute tópicos sobre programación con profundidad, el tamaño de las aplicaciones JDOM es, por lo normal, una tercera parte que el de las aplicaciones DOM y aproximadamente la mitad que el de las aplicaciones SAX. (Los puristas de DOM, por supuesto, aseguran que el uso de DOM es una buena disciplina que producirá beneficios a largo plazo.) JDOM no lo hace todo, pero para la mayor parte del parseo que se quiera hacer, es probablemente la elección correcta.

El API Java para parseo de XML

Aunque DOM, SAX y JDOM proporcionan interfaces estándar para las tareas más comunes, todavía hay varias cosas que no hacen. Por ejemplo, el proceso de creación de un objeto DOMParser en Java difiere de un parser DOM a otro. Para solucionar este problema, Sun ha publicado JAXP, el API Java para parseo de XML. Este API proporciona interfaces comunes para el procesado de documentos XML usando DOM, SAX y XSLT.

JAXP proporciona interfaces tales como `DocumentBuilderFactory` y `DocumentBuilder`, los cuales proporcionan a su vez una interfaz estándar a diferentes parsers. Existen también métodos que le permiten controlar si el parser subyacente utiliza namespaces y si usa DTDs o esquemas para validar los documentos XML.

¿Qué interfaz es la correcta para usted?

Para determinar que interfaz de programación es la correcta, necesita comprender la intención con que se diseñaron y necesita comprender que es lo que su aplicación tiene que hacer con los documentos XML que va a procesar. Considere las siguientes preguntas como una ayuda para encontrar la aproximación adecuada.

- **¿Su aplicación estará escrita en Java?** JAXP trabaja con DOM, SAX y JDOM; si esta escribiendo su código en Java, debería usar JAXP para aislar su código de los detalles de implementación de los diversos parsers.
- **¿Cómo será distribuida su aplicación?** Si su aplicación va a ser distribuida cómo un applet Java, y quiere minimizar la cantidad de código descargado, tenga en mente que los parser SAX son más pequeños que los parser DOM. También debe considerar que el uso de JDOM requiere una cantidad de código menor que los parser SAX o DOM**.**
- **¿Una vez que se ha parseado el documento, necesitará acceder a esos datos varias veces?** Si necesita volver sobre la versión parseada del documento XML, DOM es

probablemente la elección correcta. Cuando un evento SAX se ha disparado es responsabilidad suya (del desarrollador) almacenarlo si lo va a necesitar más tarde. Si necesita acceder a un evento que no salvo, debe parsear el fichero de nuevo. DOM almacena todos los datos automáticamente.

- **¿Necesita solo unas pocas cosas del documento XML?** Si solo va a necesitar unas pocas cosas del documento XML, SAX es probablemente la elección correcta. SAX no crea objetos para todo lo que contiene el documento fuente; se puede decidir que es lo importante. Con SAX puede comprobar cada evento par determinar si es relevante para sus necesidades y procesarlo entonces adecuadamente. Aún mejor, una vez que ha encontrado lo que andaba buscando, su código puede elevar una excepción para parar definitivamente el parser SAX.
- **¿Está trabajando en una máquina con muy poca memoria?** Si es así, SAX es la mejor elección, independientemente de todos los otros factores que pueda considerar.

Recuerde que existen APIs XML para otros lenguajes; Las comunidades Perl y Phyton en particular tienen muy buenas herramientas XML.

Sección 6. Estándares XML

Visión General

Existe una gran variedad de estándares en el universo XML. Además del estándar base XML, otros estándares definen esquemas, hojas de estilo, enlaces, servicios Web, seguridad y otros importantes items. Esta sección cubre los estándares más populares, y le indica referencias para encontrar otros estándares.

La especificación XML

Esta especificación, localizada en *w3.org/tr/rec-xml,* define las reglas básicas para los documentos XML. Todas las reglas para documentos XML definidas anteriormente en este tutorial están definidas aquí.

Además del estándar básico XML, la especificación de Namespaces es otra parte importante de XML. Puede encontrar el estándar para los namespaces en: *w3.org/TR/REC-xml-nombres/.*

Esquema XML

El lenguaje de Esquemas XML está definido en tres partes:

- **Un primario**, localizado en *w3.org/TR/xmlschema-0*, que proporciona una introducción a los documentos de esquema XML y al para qué están diseñados.
- Un estándar para los **documentos de estructura**, localizados en *w3.org/TR/xmlschema-1,* que ilustra cómo se define la estructura de los documentos XML.
- Un estándar para los **tipos de datos**, localizada en *w3.org/TR/xmlschema-2,* que define algunos tipos de datos comunes y las reglas para crear otros nuevos.

Este tutorial discute brevemente sobre los esquemas en Definición del contenido de un documento en la página 13; si desea completar detalles sobre todas las cosas que puede hacer con los esquemas XML, el **primario** es el mejor lugar para empezar.

XSL, XSLT, y XPath

El Lenguaje Extensible de Hojas de Estilo, XSL (Extensible Stylesheet Language), define un conjunto de elementos (llamados elementos de formato) que describen como los datos deben ser formateados. Por claridad, este estándar aún se designa como XSL-FO para distinguirlo de XSLT. Aunque inicialmente fue diseñado para generar documentos para impresión de gran calidad, también se pueden usar los objetos de formato para generar ficheros de audio desde XML. El estándar XSL-FO se encuentra en *w3.org/TR/xsl*/.

El Lenguaje Extensible de Hojas de Estilo para Transformaciones, XSLT (eXtensible Stylesheet Language for Transformations), es un vocabulario XML que describe cómo convertir un documento XML en algo más. El estándar esta en *w3.org/TR/xslt*

XPath, el lenguaje de rutas XML es una sintaxis que describe localizaciones en los documentos XML. XPath se usa en las hojas de estilo XSLT para describir qué partes de un documento XML se quiere transformar. XPath también es usado in otros estándares XML, siendo este el porqué es un estándar separado de XSLT. XPath está definido en *w3.org/TR/xpath.*

DOM

El Document Object Model define cómo se convierte un documento XML en una estructura de árbol en memoria. DOM está definido en un varias especificaciones en la W3C:

- **El nucleo (core) DOM** define el DOM mismo, la estructura de árbol y los tipos de nodos y excepciones que su código puede encontrar cuando se mueve a través del árbol. La especificación completa se encuentra en *w3.org/TR/DOM-Level-2-Core/.*
- **Eventos,** define los eventos que pueden sucederle al árbol como esos eventos son procesados. Esta especificación es un intento de reconciliar las diferencias en los modelos de objetos soportados por Netscape e Internet Explorer desde la versión 4 de estos navegadores. Esta especificación esta en *w3.org/TR/DOM-Level-2-Events/.*
- **Estilo** define cómo las hojas de estilo XSLT y las hojas de estilo CSS pueden ser accedidas por un programa. Esta especificación esta en *w3.org/TR/DOM-Level-2-Style/.*
- **Traversaciones y Rangos,** define interfaces que permiten a los programas traversar el árbol o definir un rango de nodos en el árbol. Puede encontrar la especificación completa en *w3.org/TR/DOM-Level-2-Traversal-Range/* .
- **Vistas,** define una interfaz `AbstractView` para el documento mismo. Puede encontrar más información en *w3.org/TR/DOM-Level-2-Views/.*

SAX, JDOM, y JAXP

El Simple API for XML define los eventos e interfaces usados para interactuar con un parser XML que cumpla con SAX. Puede encontrar la especificación completa de SAX en *www.saxproject.org*. El proyecto JDOM fue creado por Jason Hunter y Brett McLaughlin y se encuentra en *jdom.org*/. En el sitio de JDOM puede encontrar código, programas de ejemplo, y otras herramientas para

ayudarle a empezar (Para encontrar artículos sobre JDOM en *developerWorks*, mire en Recursos en la página 32.)

Un punto significativo acerca de SAX y JDOM es que ambos han salido de la comunidad de desarrolladores de XML, no de un cuerpo de estándares. Su amplia aceptación es un tributo a la participación de los desarrolladores XML en el mundo entero.

Puede encontrar todo lo que hay que saber acerca de JAX en *java.sun.com/xml/jaxp/*.

Enlazado y referencia

Hay dos estándares para el enlazado y referencia en el mundo XML: XLink y XPointer:

- **XLink**, el lenguaje XML de enlazado, define muchas maneras de enlazar diferentes recursos juntos. Puede hacer enlaces normales punto-a-punto (como con el elemento XML <a>) o enlaces extendidos, que pueden incluir enlaces multipunto, enlaces a través de terceros y reglas que definen lo que significa seguir un enlace determinado. El estándar XLink se encuentra en *w3.org/TR/XLink/*.
- **XPointer**¸ el lenguaje de apuntadores XML, usa XPath como una manera de referenciar otros recursos. También incluye algunas extensiones a XPath. Puede encontrar su especificación en *www.w3.org/TR/xptr/*.

Seguridad

Hay dos estándares significativos que determinan la seguridad de los documentos XML. Uno es el estándar **XML Digital Signature** o Firma Digital XML (*w3.org/TR/xmldsig-core/*), que define una estructura de documento XML para las firmas digitales. Puede crear una firma digital XML para cada tipo de datos, tanto si es un documento XML, un fichero HTML, texto plano, datos binarios, etc. Puede usar la firma digital para verificar que un fichero concreto no ha sido modificado después de que fuese firmado. Si el dato que ha firmado es un documento XML, puede incrustar el documento XML en el fichero de firma, lo que hace que el procesado de los datos y de la firma sea muy simple.

Los otros estándares determinan el encriptado de documentos XML. Aunque es bueno que los documentos XML sean escritos de tal manera que un humano pueda leerlos y entenderlos, esto puede significar un problema si el documento cae en las manos equivocadas. El estándar **XML Encryption** o Encriptación XML *(w3.org/TR/xmlenc-core/*) define como partes de un documento XML pueden ser encriptadas.

Usando estos estándares juntos, puede usar documentos XML con confianza. Yo puedo firmar digitalmente un importante documento XML, generando una firma que incluya el mismo documento. Puedo entonces encriptar el documento (usando mi clave privada y su clave pública) y enviárselo a usted. Cuando lo reciba, puede desencriptarlo con su clave privada y mí clave pública, lo que le permitirá saber que he sido yo el único que le ha enviado el documento (en

caso necesario, también puede probar que fui yo quién se lo envió). Una vez desencriptado el documento, puede usar la firma digital para asegurarse de que no ha sido modificado en ninguna forma.

Servicios Web

Los **Servicios Web** es un importante nuevo tipo de aplicación. Un servicio Web es un fragmento de código que puede ser encontrado, descrito y accedido usando XML. Hay una gran actividad en este espacio, pero los tres principales estándares XML para los servicios Web son:

- **SOAP:** originalmente, el Simple Object Access Protocol (Protocolo para Acceso Sencillo a Objetos), SOAP define un formato de documento XML que describe como invocar un fragmento de código remoto. Mi aplicación crea un documento XML que describe el método que quiero invocar, pasándole cualquier parámetro necesario, y entonces envía ese documento a través de la red hacia el fragmento de código. El código recibe el documento XML, lo interpreta, invoca el método que solicité y entonces retorna un documento XML que describe los resultados. La especificación de las versión 1.1 de SOAP se encuentra en *w3.org/TR/SOAP*/. Visite *w3.org/TR/* para ver todas las actividades de W3C relativas a SOAP.
- **WSDL:** El Web Services Description Language (Lenguaje para la Descripción de Servicios Web) es un vocabulario XML que describe un servicio Web. Es posible escribir un fragmento de código que tome un documento WSDL e invoque un servicio Web que nunca ha visto antes. La información del fichero WSDL define el nombre del servicio Web, los nombres de los métodos, los argumentos de esos métodos y otros detalles. Puede encontrar la última especificación de WSDL en *w3.org/TR/wsdl* .
- **UDDI:** La Universal Description, Discovery and Integration protocol (protocolo de Descripción Universal, Exploración e Integración) define una interfaz SOAP con un registro de servicios Web. La especificación UDDI define cómo añadir la descripción de un servicio al registro. Si está buscando un fragmento de código que proporciona una determinada función, la especificación UDDI define cómo consultar el registro para encontrar lo que busca. La fuente de todas las cosas acerca de UDDI es *uddi.org*.

Otros estándares

Existe un gran número de otros estándares XML que no veremos aquí. Además de los estándares de aplicación general como Scalable Vector Graphics *(www.w3.org/TR/SVG*/) Vector Graphics y SMIL, el Syncronized Multimedia Integration Language *(www.w3.org/TR/smil20/*), hay muchos estándares específicos en la industria. Por ejemplo, el Consorcio HR-XML ha definido una gran cantidad de estándares XML para Recursos Humanos; puede encontrarlos en *hr-xml.org* .

Finalmente, para una buena fuente de estándares XML visite el Repositorio XML en *xml.org/xml/registry.jsp*. Este sitio proporciona cientos de estándares para una gran variedad de industrias.

Sección 7. Casos de estudio

Ejemplos del mundo real

En este punto, espero que este convencido del tremendo potencial de XML para revolucionar la forma en la que trabaja el comercio electrónico. Aunque su potencial es grande lo que realmente cuenta son los resultados actuales en el mercado. Esta sección describe tres casos de estudio de organizaciones que han usado XML para agilizar sus procesos de negocio y mejorar sus resultados.

Todos los casos de estudio discutidos aquí provienen del programa IBM jStart. El equipo jStart existe para ayudar a los clientes a usar nuevas tecnologías para solucionar problemas. Cuando un cliente da su conformidad al contrato jStart sabe que el proyecto resultante será usado como caso de estudio. Si quiere ver más casos de estudio, incluyendo casos de estudio que implican servicios Web y otras tecnologías nuevas, visite la página Web de jStart en *ibm.com/software/jstart*.

Debe saber que el equipo jStart ya no está realizando contratos para proyectos XML; el equipo actualmente esta centrado en la contratación de servicios Web. Los servicios Web usan XML en una forma especializada, tipicamente a través de los estándares SOAP, WSDL y UDDI como mencionamos antes en Servicios Web, en la página 27.

Provincia de Manitoba

El gobierno de la Provincia de Manitoba creó el Registro de la Propiedad Personal para proporcionar a los propietarios servicios internet sobre el estado-del-arte durante las 24 horas. Los principales beneficios de la aplicación fueron un acceso más rápido y conveniente a los datos de las propiedades, menos pasos manuales en el proceso de administración de las propiedades y menos llamadas a la central de llamadas del gobierno. En otras palabras darle a los usuarios un mejor servicio, reduciendo el gasto y la carga de trabajo del gobierno.

Diseño de la aplicación

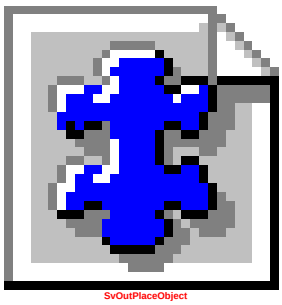
La aplicación fue diseñada como una aplicación multi-capas, con la interfaz separada de la lógica de negocio. Los datos para cada transacción necesitaron ser transformado en varias formas diferentes, dependiendo de si debían ser formateados en un dispositivo, presentados en una aplicación o formateados por el sistema de procesado back-end**.** En otras palabras esta aplicación era una oportunidad perfecta para usar XML.

Como con cualquier aplicación, la interfaz de usuario era extremadamente importante. Para simplificar la primera implementación, los datos XML necesarios fueron transformado en HTML. Esto proporciono a los usuarios una interfaz de navegador para la aplicación. El registro fue construido con VisualAge for Java, específicamente el componente Visual Servlet Builder. También se uso Enterprise Java Beans (EJBs), incluyendo beans de Sesión y de Entidad.

Generando múltiples interfaces de usuario con XML

Además de la interfaz HTML, un interfaz cliente Java y un interfaz electrónico para B2B fueron planeados también. Para todos estos interfaces, los datos estructurados en XML fueron transformados en las estructuras y documentos apropiados. Las pruebas iniciales del servicio autorizaron a un único cliente, Canadian Securities Registration Systems, para enviar datos de transacciones XML usando SSL. Los datos de transacción XML fueron transformados en el formato apropiado por las transacciones back-end .

El resultado final es que la Provincia de Manitoba fue capaz de crear una aplicación nueva y flexible y sus usuarios finales pudieron acceder la información del registro de propiedad más fácil y rápidamente. Dado que la provincia usa XML como formato de datos el equipo de IT del gobierno tiene una gran capacidad de flexibilidad para el diseño de nuevos interfaces y métodos de acceso. Lo mejor de todo, los sistemas back-end no tienen que cambiar en absoluto.



El First Union Bank en XML

El First Union National Bank, uno de los mayores bancos en los U.S., esta en proceso de rediseñar muchas de sus aplicaciones usando Java y XML. Como muchas compañías grandes, su entorno es heterogéneo, con servidores OS/390, AIX, Solaris, HP/9000 y Windows NT y clientes Windows NT, Windows 98, Solaris y AIX. Dado este entorno, First Union eligió Java y XML para conseguir código y datos independientes de la plataforma.

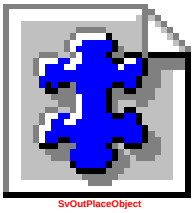
Un sistema basado en mensajes

Las aplicaciones distribuidas del banco se construyeron sobre una infraestructura de mensajería, usando MQSeries de IBM para enviar mensajes al sistema OS/390. El contenido del mensaje está basado en una especificación llamada Common Interface Message (CIM), un estándar propietario de First Union. Los componentes del front-end y el back-end de la aplicación son dependientes del formato del mensaje. Usando XML como formato de datos, aislamos ambos lados de la aplicación de cambios futuros así como del protocolo de mensajería.

Uso de herramientas XML para automatizar el flujo de datos

Durante el desarrollo de la aplicación basada en XML, el First Union y el equipo de IBM crearon un servicio que convierte CIM en documentos XML. Otra parte de la aplicación convierte la solicitud XML en un formato apropiado para para su proceso por los sistemas de back-end. Finalmente, un tercer servicio convierte los copy books de COBOL en DTDs. Una vez que los *copy books* han sido convertido a DTDs, First Union pudo usar el DTD y el parser XML4J para validar los documentos XML automáticamente; el banco pudo entonces estar seguro de que los documentos XML coincidían con la estructura de datos COBOL que esperaba el OS/390.

El uso de tecnología Java y XML ha sido muy exitoso para First Union. Según Bill Barnett, Administrador del Distributed Object Integration Team en First Union, “La combinación de Java y XML realmente nos ha impulsado. Sin un entorno independiente de la plataforma como Java y la independencia del protocolo de mensajería obtenida mediante XML, no habríamos confiado en que nuestra infraestructura distribuida pudiera evolucionar para cubrir la demanda de nuestra siempre creciente base de datos de clientes”.



Hewitt Associates LLC

Hewitt Associates LLC es una firma de consultoría global especializada en soluciones de recursos humanos. La compañía tiene más de 200 corporaciones cómo clientes en todo el mundo; estas compañías tienen a su vez más de 10 millones de empleados. Los clientes de Hewitt demandaban una distribución más rápida y fiable de información sobre esos 10 millones de empleados.

Antes de su asociación a jStart, Hewitt construyo su propias soluciones propietarias cuando sus clientes solicitaban datos sobre recursos humanos. Estas soluciones propietarias eran tipicamente pasarelas a las aplicaciones ya existentes de Hewitt. Estas aplicaciones de cliente eran muy costosas de desarrollar, comprobar y distribuir, llevando a Hewitt a investigar los servicios Web.

¡Los servicios Web al rescate!

Para resolver estos problemas, Hewitt y el equipo de jStart trabajaron juntos para construir servicios Web que solucionaran las necesidades de los clientes de Hewitt. Los servicios Web son un nuevo tipo de aplicación que usa XML en varias maneras interesantes:

- En primer lugar, los servicios Web, tipicamente usan SOAP, un estándar XML para mover datos de un lugar a otro.

- En segundo lugar, los interfaces proporcionados por un servicio Web (nombres de métodos, parámetros, tipos de datos, etc..) están descritos con XML.
- Lo siguiente es que la descripción de un servicio Web puede estar almacenada o ser devuelta por un registro UDDI y toda la información que entra o sale del registro está en formato XML.
- Finalmente, los datos proporcionados por el servicio Web son ellos mismos XML.

Hewitt ha desarrollado dos aplicaciones que ilustran su habilidad para distribuir datos de muchas y flexibles maneras:

- Con Secure Participant Mailbox, los usuarios autorizados pueden solicitar informes conteniendo información personalizada sobre jubilaciones y otros beneficios de los empleados.
- Con Retirement Access B2B Connection, los usuarios autorizados pueden obtener detalles sobre la información financiera de un cliente.(401k)

Ambas aplicaciones devuelven datos desde sistemas ya existentes, usando XML para formatear los datos y distribuyendo la información ya formateada a través de la Web. Debido a que estas aplicaciones están construidas sobre estándares abiertos, Hewitt puede distribuirlas rápidamente. Lo mejor de todo, la flexibilidad de estas aplicaciones ayuda a Hewitt a distinguirse de sus competidores.

“Nosotros vemos los servicios Web como un vehículo para proporcionar acceso abierto y no propietario a los participantes a nuestros servicios de negocio y datos a través de una red de datos ubicua”, dijo Tim Hilgenberg, Jefe de Tecnología Estratégica en Hewitt. El resultado final: Hewitt desarrolla aplicaciones más flexibles, de un modo más rápido y barato, los clientes tienen mejores acceso a sus datos y las aplicaciones existentes en Hewitt no han tenido que cambiar.

Resumen de Casos de estudio

En todos los casos de estudio, las compañías usaron XML para crear un formato de datos independiente del sistema. Los documentos XML representan datos estructurados que pueden ser movidos de un sistema o proceso a otro. Tanto el front-end como el back-end cambian, el XML que viaja entre ambos permanece constante. Aún mejor, cuantas más aplicaciones front-end y back-end se añadan a la mezcla, tanto más, el uso de XML, protegerá a las aplicaciones existentes frente a los cambios. Conforme los servicios Web sean más comunes, XML también será usado para transportar los datos.

Para más información sobre estos casos de estudio, contacte con Sam Thompson de IBM en *thompsam@us.ibm.com* . Puede encontrar más información sobre la Provincia de Manitoba en *www.gov.mb.ca*, el sitio Web de First Union es *firstunion.com* y Hewitt Associates está en la Web en *hewitt.com*.

Sección 8. Consejos y recursos

¡Adelante!

En este punto, espero que este convencido de que XML es la mejor manera de mover y manipular datos estructurados. Si no esta usando aún XML, ¿a qué espera para empezar?. Aquí tiene algunas sugerencias:

- **Decida que datos quiere convertir en XML.** Tipicamente estos son los datos que necesita mover entre sistemas o que deben ser transformados en varios formatos.
- **Compruebe si ya existe algún estándar XML**. Si esta usando datos muy comunes cómo ordenes de compra, registros médicos, o reservas en almacén, tiene muchas posibilidades de que alguien ya haya definido un estándar XML para esos datos.
- **Compruebe si sus herramientas actuales soportan XML.** Si está usando una versión reciente de un paquete de base de datos, hoja de cálculo o algún otro tipo de herramienta para la administración o manejo de datos, es probable que sus herramienta (o alguna actualización) pueden usar XML como formato para la entrada o salida de datos.
- **Aprenda cómo construir aplicaciones basadas en XML.** Necesita comprender como sus datos están almacenados actualmente, como necesitan ser transformados y cómo integrar sus esfuerzos de desarrollo XML con la aplicaciones existentes. La columna de Benoît Marchall's *Working XML* es un estupendo lugar donde empezar; puede encontrar un listado actualizado de todas sus columnas en *http://www-106.ibm.com/developerworks/xml/library/x-wxxmcol/* .
- **Unase al grupo de estándares apropiado.** Considere unirse a grupos como el World-Wide Web Consortium (W3C) , así como a grupos de industrias específicas cómo HR-XML.org. Ser un miembro de esos grupos le ayudará a estar informado sobre lo que ocurren en la industria, y le da la oportunidad de modelar el futuro de los estándares XML.
- **Evite soluciones propietarias.** Es importante que use sólo tecnología basada en estándares en sus esfuerzos de desarrollo; resista los señuelos de los vendedores que le ofrecen "mejoras". Una de las ventajas de XML es que puede tener control completo sobre sus datos. Una vez que ha sido tomados como rehenes por un formato de datos propietario, se padece una tremenda perdida del control.
- **Contacte con el equipo jStart.** Si piensa que su empresa podría trabajar con el modelo de contrato de jStart, contacte con el equipo y compruebe cuales son sus posibilidades.
- **Permanezca atento a *developerWorks*.** Nuestra zona XML tiene miles de paginas cuyo contenido abarca varios tópicos XML, incluyendo desarrollo de DTDs y de esquemas, programación XML y creación de hojas de estilo XSLT.

Recursos

Aquí presentamos algunos recursos para ayudarle a empezar:

The dW XML zone Uno de nuestro lugares favoritos para ver recursos XML. Ver *www-106.ibm.com/developerworks/xml* para todo lo que siempre quiso saber sobre XML.

XML tools: *developerWorks* tiene "Fill your XML toolbox" con artículos que describen herramientas de programación XML para varios lenguajes:

- **C/C++:** Ver el artículo de Rick Parrish en *www-106.ibm.com/developerworks/library/x-ctlbx.html (developerWork*s, Septiembre 2001).
- **Java:** Ver el artículo de Doug Tidwell en *www-106.ibm.com/developerworks/library/java-xml-toolkit/index.html (developerWork*s, Mayo 2000).
- **Perl:** Ver el artículo de Tony Darugar en *www-106.ibm.com/developerworks/library/perl-xml-toolkit/index.html (developerWork*s, Junio 2001).
- **PHP:** Ver el artículo de Craig Knudsen en *www-106.ibm.com/developerworks/library/php-xml-toolkit.html (developerWork*s, Junio 2000).

Además de estos artículos puede ver el examen que David Mertz hace a las herramientas XML para Phyton en su artículo "Charming Python: Revisiting XML tools for Python" en*www-106.ibm.com/developerworks/library/l-pxml.html* .

Tutoriales XML: Docenas de tutoriales sobre XML están disponibles en *developerWork*s; puede ver *http://www-105.ibm.com/developerworks/education.nsf/dw/xml-onlinecourse-bytitle* para la lista más actualizada.

IBM's jStart team: El equipo de jStart trabaja a muy bajo precio para ayudar a los clientes a construir soluciones usando las nuevas tecnologías (Servicios Web XML, por ejemplo). A cambio, esos clientes le permiten a IBM publicar sus proyectos cómo casos de estudio. Para mas información, ver *ibm.com/software/jstart*.

Estándares XML: Presentamos una lista por orden alfabético de los estándares mencionados en este tutorial.

- **DOM**, el Document Object Model:
  - **Core Specification:** *w3.org/TR/DOM-Level-2-Core/*
  - **Events Specification:** *w3.org/TR/DOM-Level-2-Events/*
  - **Style Specification:** *w3.org/TR/DOM-Level-2-Style/*
  - **Traversal and Range Specification:***w3.org/TR/DOM-Level-2-Traversal-Range/.*
    - **Views Specification:** *w3.org/TR/DOM-Level-2-Views/*

HR-XML.org, El Human Resources XML Consortium: *hr-xml.org*
JAXP, el Java API for XML: *java.sun.com/xml/jaxp/*
JDOM, que actualmente no está recibiendo soporte *jdom.org/*
SAX, el Simple API for XML: *saxproject.org/*
SMIL, el Synchronized Multimedia Integration Language: *www.w3.org/TR/smil20/*
SOAP, que estaba soportado por Simple Object Access Protocol, pero que ahora, oficialmente no esta soportado por nadie: *w3.org/TR/SOAP/*
SVG, Scalable Vector Graphics: *www.w3.org/TR/SVG/*
UDDI, Universal Description, Discovery, and Integration Protocol: *uddi.org*

WSDL, Web Services Description Language: *w3.org/TR/wsdl* (no closing slash)
XLink, XML Linking Language: *w3.org/TR/XLink/*
XML, el estándar que lo comenzó todo: *w3.org/TR/REC-xml*
XML Digital Signatures: *w3.org/TR/xmldsig-core/*
XML Encryption: *w3.org/TR/xmlenc-core/*
XML Namespaces: *w3.org/TR/REC-xml-nombres/*
XML Repository de DTDs y esquemas: *xml.org/xml/registry.jsp*
XML Schema:
Part 0 - **Primer**: *w3.org/TR/xmlschema-0*
Part 1 - **Document structure**s: *w3.org/TR/xmlschema-1*
Part 2 - **Datatype**s: *w3.org/TR/xmlschema-2*
XPath, el lenguaje XML Path: *w3.org/TR/xpath* (sin barra de cierre)
XPointer, el lenguaje de punteros para XML: *www.w3.org/TR/xptr/*
XSL-FO, Extensible Stylesheet Language for Formatting Objects: *w3.org/TR/xsl/*
XSLT, Extensible Stylesheet Language: *w3.org/TR/xslt* (sin barra de cierre)

Para saber más sobre JDOM, puede leer los siguientes artículos de *developerWorks*:

Simplify XML programming with JDOM at
*www-106.ibm.com/developerworks/java/library/j-jdom/ (developerWork*s, Mayo 2001)

Converting from DOM en
*www-106.ibm.com/developerworks/java/library/x-tipcdm.html (developerWork*s, Abril 2001)

Converting from SAX en
*www-106.ibm.com/developerworks/java/library/x-tipcsx.html (developerWork*s, Abril 2001)

• **Using JDOM and XSLT** en
*www-106.ibm.com/developerworks/java/library/x-tipjdom.html (developerWork*s, Marzo 2001)

Opiniones

Por favor, envie sus opiniones sobre este tutorial. Lo hemos revisado extensivamente (Agosto 2002) basandonos en sus opiniones y en los nuevos desarrollos en el mundo XML. Siempre estamos dispuestos a escucharle.