

Capítulo 6

XML: Transformando la Web en una Base de Datos

Marcelo Arenas

Una de las razones para la popularización de la Web ha sido el desarrollo de una infinidad de páginas que entregan distintos servicios; buscadores como Yahoo! y Google, grandes repositorios de información como Wikipedia, tiendas electrónicas como Amazon, diarios y revistas electrónicas, página personales, etc. Bajo este desarrollo ha estado HTML, un lenguaje que permite estructurar tanto la información como las posibilidades de navegación en una página Web.

Durante los últimos años, la cantidad de información almacenada en la Web ha ido creciendo de manera dramática. Hoy ningún usuario tiene la capacidad de recorrer la Web entera en busca de información, y es necesario utilizar buscadores automáticos como Yahoo! y Google para poder revisar una fracción significativa de esta red.

Nadie puede negar la importancia y utilidad que tienen los buscadores para encontrar información en la Web. Sin embargo, muchos usuarios pueden decir que su experiencia con ellos no ha sido completamente satisfactoria. A medida que las consultas que se quiere realizar son más complejas, la búsqueda de información puede requerir de varios, o muchos, intentos en los cuales es necesario jugar con distintos parámetros. Piense por ejemplo en la

consulta “dé la lista de libros de Ariel Rubinstein”. Para realizar esta consulta basta con poner “Ariel Rubinstein” en un buscador y usar los primeros elementos de la lista de respuesta (probablemente el primero) para encontrar la página de este autor, y ahí la lista de sus libros. Pero ahora piense en la pregunta “dé la lista de libros de Ariel Rubinstein y sus precios”. ¿Qué colocaría en un buscador para encontrar la respuesta? Peor aun, piense en una pregunta como la siguiente “dé la lista de libros de Ariel Rubinstein que han bajado de precio en los últimos años”. ¿Cómo se puede buscar esta información usando Yahoo! o Google?

¿Por qué los buscadores tienen dificultades en los ejemplos anteriores? Una de las razones es el uso de HTML; este es un lenguaje que permite desplegar información que es fácil de entender para los usuarios, pero que en general es difícil de interpretar para los computadores. Estas dificultades ya pueden verse en ejemplos tan sencillos como el siguiente:

```
<html>
  <body bgcolor="#FFFFFF">
    <center>
      <h2> Todo Libros </h2>
    </center>
    <ul>
      <li><b>Teoría de Juegos.</b>
        Martin Osborne y Ariel Rubinstein. Precio: 16000.</li>
    </ul>
  </body>
</html>
```

Este archivo es usado para mostrar la lista de libros vendidos por la librería “Todo Libros”. Nótese que este archivo ha sido indentado (espaciado) de manera que sea fácil visualizar la estructura jerárquica del documento. Por ejemplo, corresponde a un ítem en la lista definida por . En un

browser tal como FireFox o Explorer, esta lista será desplegada de la siguiente forma:

Todo Libros
<ul style="list-style-type: none">● Teoría de Juegos. Martin Osborne y Ariel Rubinstein. Precio: 16000.

Para un usuario la información en esta lista es fácil de entender; es claro que hay una lista de libros, cada uno con sus autores y su precio. Sin embargo, para un computador esta información no es tan clara. Una de las razones es que el computador no tiene la información de contexto, o meta-información, que tiene el usuario. ¿Cómo puede un computador deducir que está frente a una lista de libros? Y aun si sabe esto, ¿cómo puede extraer información desde el documento, por ejemplo los precios de los libros? Es importante notar aquí que el documento HTML no tiene ninguna indicación sobre donde buscar esta información, simplemente dice cómo debe ser desplegada la lista de libros. Así, el computador debe tratar de interpretar el texto para poder extraer la lista de precios. Por ejemplo, puede buscar la palabra "Precio" y el número que lo sigue (o antecede). Aunque en este caso esto puede dar buenos resultados, la situación puede volverse más complicada si la lista contiene varios precios para un mismo libro (precio sin descuento, con descuento por compra electrónica, con descuento a clientes frecuentes, etc), o aun más complicada si se requiere de hacer algunos cálculos para saber el precio final (precio después del 15% de descuento por compra electrónica).

La búsqueda de información en la Web puede mejorarse si los formatos usados para almacenar información pueden ser fácilmente interpretados por

los computadores. Una propuesta para hacer esto es el uso de XML, como se verá en las siguientes secciones.

XML: Un lenguaje para almacenar información

Un documento XML (eXtensible Markup Language [2]) es similar a un documento HTML; está compuesto por marcadores, o “tags”, que están anidados como en el caso de HTML. La mayor diferencia es que los marcadores de HTML tienen significados predefinidos, tales como <title> y , mientras que los de XML son definidos por el usuario. Por ejemplo, el siguiente es un documento XML que almacena la misma información que el documento HTML mostrado en la sección anterior:

```
<?xml version="1.0"?>
<libreria>
  <nombre>Todo Libros</nombre>
  <libro>
    <titulo>Teoría de Juegos</titulo>
    <autor>
      <nombre>Martin</nombre>
      <apellido>Osborne</apellido>
    </autor>
    <autor>
      <nombre>Ariel</nombre>
      <apellido>Rubinstein</apellido>
    </autor>
    <precio>16000</precio>
  </libro>
</libreria>
```

Como puede verse, el documento está compuesto por marcadores tales como `<libreria>`, `<libro>` y `<autor>`. Un marcador con nombre `<a>` es cerrado por uno con nombre ``. Los nombres de los marcadores fueron definidos por un usuario, y la única restricción que deben cumplir, como en el caso de HTML, es que deben estar correctamente anidados; si leyendo el documento de arriba hacia abajo `<autor>` aparece después de `<libro>`, entonces el marcador `</autor>` que lo cierra debe aparecer antes que el marcador `</libro>` que cierra a `<libro>`, vale decir, `<autor>` debe estar completamente contenido dentro de `<libro>`. A través de esto se especifica que `<autor>` es uno de los autores de `<libro>`.

Los marcadores del documento XML fueron diseñados para mostrar de forma clara la información sobre un libro. Si un computador quiere buscar el título de un libro, entonces basta con que busque el marcador `<título>`, y si quiere encontrar el precio del libro con título "Teoría de Juegos", entonces basta que encuentre un marcador `<libro>` que tenga "Teoría de Juegos" en `<título>`, y que después despliegue lo que se encuentra en el marcador `<precio>` dentro de ese libro. La forma en que la información está agrupada y los nombres de los marcadores le indican a un computador dónde buscar información.

XML entonces surge como una buena alternativa para almacenar información; un computador tiene mayores posibilidades de interpretar y extraer información desde este tipo de documentos. ¿Debemos entonces reemplazar HTML por XML? La respuesta es no. Estos dos lenguajes tienen distintas finalidades. Mientras HTML es usado para especificar cómo desplegar información en un browser, XML es usado para almacenar información y no contiene indicaciones de como mostrarla. Se tiene entonces que diseñar tecnologías que permitan sacar ventajas de los dos lenguajes. En la siguiente sección se verá cómo hacer esto.

Transformación de documentos XML

Una de las razones para la creación de XML fue tener un formato que permitiera intercambiar información en la Web. La idea es que si varias personas o empresas desean intercambiar datos sobre un tema común, por ejemplo libros, y usan formatos XML distintos para almacenar su información, entonces puedan intercambiar información de manera sencilla. La forma de hacer esto es usando algún lenguaje de transformación que permita cambiar de un formato a otro. Por ejemplo, si una empresa usa el formato:

```
<autor>
  <nombre>Martin</nombre>
  <apellido>Osborne</apellido>
</autor>
```

para almacenar los nombre de autores de libros, mientras otra usa un formato más simple donde el nombre es almacenado como una sola palabra:

```
<autor>Martin Osborne</autor>
```

entonces una regla de transformación desde el primer formato al segundo debe concatenar el nombre y apellido de un autor para generar su nombre como una sola palabra.

XML fue elegido como el lenguaje para intercambiar información por su gran flexibilidad, esencialmente cualquier documento XML es válido mientras la anidación de los marcadores sea correcta. El lenguaje elegido para especificar las transformaciones fue XSLT (Extensible Stylesheet Language Transformations [3]). Este es un lenguaje que busca patrones dentro de un documento e indica cómo reestructurarlos. Por ejemplo, busca el tag `<autor>`, y después indica que las palabras que aparecen dentro de `<nombre>` y `<apellido>` para este autor tienen que ser concatenadas.

XSLT no sólo permite hacer transformaciones entre documentos XML, en general permite generar cualquier tipo de documento desde un documento XML (HTML, texto plano, programa en algún lenguaje de programación como Java o C++, etc). En particular, hoy es usado por browsers tales como FireFox y Explorer para poder desplegar documentos XML. La idea aquí es simple: como XML es un mejor formato para almacenar información, conviene tener los documentos en la Web en este formato. Si un documento XML tiene que ser desplegado por un browser, entonces se usa un conjunto de reglas XSLT para generar un documento HTML desde la fuente XML, el cual es usado por el browser al desplegar la información. Veamos esto en el ejemplo anterior. Para indicar cuál es el programa XSLT a usar al desplegar un documento XML se usa una línea adicional en el documento:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="libreria.xslt"?>
<libreria>
  <nombre>Todo Libros</nombre>
  <libro>
    ...
  </libro>
</libreria>
```

En el campo `href="libreria.xslt"` se indica que se debe usar el archivo XSLT `libreria.xslt`. En la figura 6.1 se muestra parte del conjunto de reglas XSLT que es usado para transformar el documento XML, con información sobre libros en el documento HTML mostrado en la primera sección.

No se espera aquí que el lector pueda entender todos los detalles de un documento XSLT, pero sí que después de terminar esta sección tenga una idea de cómo funciona este lenguaje. Como puede verse en la figura 6.1, un documento XSLT está compuesto por una serie de patrones que son declarados a través del marcador `xsl:template`. Cada uno de estos patrones tiene

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="/">
    <html>
      <body bgcolor="#FFFFFF">
        <center>
          <h2>
            <xsl:apply-templates select="/libreria/nombre"/>
          </h2>
        </center>
        <ul>
          <xsl:apply-templates select="/libreria/libro"/>
        </ul>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="/libreria/nombre">
    <xsl:value-of select="."/>
  </xsl:template>

  <xsl:template match="/libreria/libro">
    ...
  </xsl:template>

  ...

</xsl:stylesheet>
```

Figura 6.1: Reglas XSLT para transformar un documento XML en HTML.

un atributo `match` que indica dónde se debe usar el patrón. Por ejemplo, el patrón:

```
<xsl:template match="/libreria/nombre">
  <xsl:value-of select="."/>
</xsl:template>
```

debe ser usado en todos los nodos del documento XML que son alcanzados siguiendo el camino `/libreria/nombre` desde el punto inicial del documento. Así, en el ejemplo se va a alcanzar el elemento con marcador `<nombre>`, que es hijo del elemento con marcador `<libreria>`. En el patrón de arriba, se utiliza `xsl:value-of` para indicar qué seleccionar desde este elemento, en este caso `Todo Libros` ya que se usa `select="."`.

Nótese que el documento XSLT tiene un solo patrón tal que `match="/"`. Este es el primer patrón que debe ser usado, y en él se indica que el documento a construir es de la forma:

```
<html>
  <body bgcolor="#FFFFFF">
    <center>
      <h2>
        <xsl:apply-templates select="/libreria/nombre"/>
      </h2>
    </center>
    <ul>
      <xsl:apply-templates select="/libreria/libro"/>
    </ul>
  </body>
</html>
```

En este documento HTML aparece dos veces `xsl:apply-templates`. Esto es usado para indicar que en esos puntos se debe colocar los resultados de aplicar los patrones correspondientes. Por ejemplo, en el caso de:

```
<h2> <xsl:apply-templates select="/libreria/nombre"/> </h2>
```

se debe usar el patrón que contiene la expresión `match="/libreria/nombre"`. Ya se había visto que este patrón genera como respuesta `Todo Libros`, por lo que al hacer el reemplazo se va a obtener:

```
<html>
  <body bgcolor="#FFFFFF">
    <center>
      <h2> Todo Libros </h2>
    </center>
    ...
  </body>
</html>
```

Si se compara esto con el documento HTML mostrado en la sección inicial, se dará cuenta que lo que se muestra arriba coincide con la primera parte del documento HTML inicial. Para construir el resto del documento se utiliza el patrón que contiene la expresión `match="/libreria/libro"`. Este patrón, y el resto del documento XSLT, son omitidos en la figura 6.1.

En el enfoque para almacenar información descrito en esta sección, los datos son almacenados en un archivo XML, el cual es desplegado en un browser usando un conjunto de reglas XSLT que indican cómo generar un archivo HTML desde el archivo XML original. Para sacar el mayor provecho a este enfoque, todavía nos falta indicar cómo se puede extraer información desde un documento XML. Esto se verá en la siguiente sección.

Extracción de información desde XML

En las secciones anteriores se mostró un enfoque para almacenar información en la Web en el cual los datos son almacenados en XML y mostrados a los usuarios en HTML (utilizando transformaciones escritas en XSLT). Se argumentó que éste era un buen enfoque porque permitía tener lo mejor de dos mundos: por una parte para un computador es más fácil interpretar información escrita en XML, y por lo tanto es más fácil extraer información desde este formato; y por otra parte HTML provee de buenas herramientas para desplegar información en la Web.

Para que el enfoque anterior pueda llevarse a cabo es necesario tener buenos lenguajes de consulta para XML. Estos lenguajes deben ser suficientemente expresivos como para permitir al usuario expresar consultas generales, y también deben estar acompañados de procedimientos eficientes para evaluar consultas. En esta sección se va a introducir XPath y XQuery, los dos lenguajes de consulta más populares para XML.

La primera versión estandarizada de XPath es de 1999 [4]. XPath puede ser considerado como el lenguaje de consulta más popular para XML, ya que forma parte de la mayor parte de los lenguajes de consulta para XML y, en particular, es parte de XQuery [1], como se verá más adelante. XPath provee una serie de herramientas que permiten navegar un documento XML, seleccionar elementos desde él y extraerlos para ser desplegados o usados por otras consultas. Una de las razones de la popularidad de XPath es que estas herramientas son simples de usar, y son lo suficientemente expresivas para poder manejar muchas de las consultas que los usuarios tienen en la práctica. Además, la estructura simple de este lenguaje ha permitido el desarrollo de procedimientos eficientes para evaluar consultas.

La mejor manera de entender XPath es a través de algunos ejemplos. Suponga que se está utilizando el documento XML con información sobre libros descrito en la sección 6.1, y que se ha utilizado repetidas veces en este capítulo. Si un usuario quiere extraer el nombre de la librería, entonces puede utilizar la siguiente consulta XPath:

```
child/?nombre/text()
```

Esencialmente una consulta en XPath consiste de un camino, y su respuesta es el conjunto de todos los elementos que pueden ser alcanzados en un documento XML, siguiendo el camino desde el primer elemento de este documento. En una consulta XPath se pueden utilizar palabras que tienen un significado reservado (`child` y `text()` en el ejemplo) o palabras cuyo significado está dado por un documento (`nombre` en el ejemplo). Además, en una expresión XPath se puede utilizar el símbolo `?` para indicar que se quiere chequear una condición. En el ejemplo, la palabra reservada `child` es utilizada para pasar de un elemento a sus hijos y `?nombre` indica que sólo se va a considerar los elementos con marcador `<nombre>`. De esta forma, utilizando la expresión `child` en el ejemplo se pasa de un elemento con marcador `<libreria>` a los que tiene marcadores `<nombre>` y `<titulo>`, y luego utilizando el test `?nombre` se selecciona el único elemento con marcador `<nombre>` hijo del elemento con marcador `<libreria>`. Finalmente se utiliza `text()` para extraer el texto almacenado dentro del elemento con marcador `<nombre>`, vale decir, `Todo Libros`.

Es importante destacar que para simplificar la presentación del lenguaje XPath, no se está usando aquí la sintaxis de XPath definida en [4], sino que una versión simplificada (pero que refleja la forma en que trabaja XPath).

Suponga ahora que se quiere extraer la lista de apellidos de todos los autores de libros. Para hacer esto, se puede utilizar la siguiente consulta:

```
descendant/?apellido/text()
```

La mayor diferencia con la consulta anterior es la utilización de la palabra reservada `descendant`, la cuál indica que se debe utilizar a los descendientes del primer elemento del documento, vale decir, a los elementos que son alcanzables utilizando los caminos `child`, `child/child`, `child/child/child`, etc. Nótese que esta consulta funciona incluso en casos en que la información sobre autores es dada de manera menos estructurada:

```
...
<primer_autor>
  <nombre>Martin</nombre>
  <apellido>Osborne</apellido>
</primer_autor>
<segundo_autor>
  <nombre>Ariel</nombre>
  <apellido>Rubinstein</apellido>
</segundo_autor>
...
```

En general, se considera una ventaja de XPath el que pueda funcionar sobre información semi-estructurada, ya que en la práctica la estructura de muchos documentos XML es irregular.

En este punto, el lector probablemente se ha dado cuenta de que la consulta anterior puede funcionar de manera incorrecta si el documento no sólo contiene apellidos de autores (por ejemplo, contiene los apellidos de la gente que trabaja en la librería). En ese caso se puede utilizar la consulta `descendant/?libro/descendant/?apellido/text()` que busca apellidos que aparezcan dentro de elementos con marcador `<libro>`.

Una de las limitaciones de XPath es la falta de herramientas para estructurar la información que se extrae; una consulta en XPath retorna un conjunto de elementos y no un documento XML. XQuery es un lenguaje más

completo, que usa XPath para navegar documentos XML y tiene herramientas para estructurar la información extraída como un documento XML [1]. En el siguiente ejemplo se muestra una consulta XQuery:

```
let $lib := doc("libreria.xml")
return
  <lista>
  {
    for $x in $bib/child/?libro
    for $y in $x/descendant/?apellido
    where $y/text() = Rubinstein
    return
      <libro>
      {
        <titulo> $x/descendant/?titulo/text() </titulo>
        <precio> $x/descendant/?precio/text() </precio>
      }
      </libro>
  }
</lista>
```

Al igual que para el caso de XPath, en una consulta XQuery pueden aparecer elementos que tienen un significado predefinido y otros que deben ser interpretados en un documento XML. En la consulta anterior, `let` es utilizado para indicar que la variable `$lib` está ligada al documento `libreria.xml` (una variable en XQuery comienza con el símbolo `$`). Además, en esta consulta `for` es usado para indicar que una variable debe tomar todos los valores alcanzados al utilizar un camino en XPath. Por ejemplo, `for $x in $bib/child/?libro` indica que `$x` va a tomar como valor los elementos con marcador `<libro>` que son hijos del primer elemento del documento. Nótese que al igual que en un lenguaje de programación, las

instrucciones que utilizan `for` pueden aparecer anidadas. En la consulta anterior, `where` es usado para chequear una condición y `return` para indicar que algo debe estar en la salida de la consulta. Así, por ejemplo, en la condición `where $y/text() = Rubinstein` se chequea que el apellido del autor que se va a utilizar sea `Rubinstein`. Es importante destacar que en una consulta XQuery se puede indicar cómo se va a estructurar la respuesta colocando marcadores XML. En el ejemplo, `<lista>` es el marcador del primer elemento del documento de salida, y contiene como hijos una serie de libros con marcador `<libro>`.

Seguramente el lector ya se ha dado cuenta que la consulta anterior retorna la lista de libros escritos por Rubinstein con sus precios. Esta es una de las consultas que se planteó al principio de este capítulo, y para las cuales no era claro como responderlas si la información era almacenada en documentos HTML. Como se muestra en el ejemplo, si la información se almacena en formato XML, una simple consulta en XQuery puede bastar para extraer la información deseada. Incluso en el caso de la consulta más compleja vista al comienzo de este capítulo (“dé la lista de libros de Rubinstein que han bajado de precio en los últimos años”), una consulta en XQuery puede ser usada para extraer la información deseada.

Para recordar

¿Qué debería recordar el lector después de navegar por este capítulo? El lector debería estar satisfecho si la arquitectura presentada en la Figura 6.2 le resulta familiar.

En caso de que el lector no recuerde todos los componentes de la Figura 6.2, aquí damos un breve resumen de lo que se trató este capítulo. El lenguaje HTML es usado para indicar a un browser (tal como FireFox o Explorer) la

forma en que se debe desplegar la información. Aunque el resultado de desplegar esta información es fácil de entender para los usuarios (como vemos a diario en las páginas Web que visitamos), es, en general, difícil de entender para un computador. Para solucionar este problema, XML ha surgido como un lenguaje para almacenar información, que es de fácil procesamiento para un computador. Es importante destacar que XML no ha venido a reemplazar HTML, muy por el contrario se ha convertido en su complemento; la información se almacena en XML y se despliega utilizando HTML, lo que nos permite tener lo mejor de estos dos mundos. Una serie de tecnologías han sido desarrolladas para sacar el máximo de provecho al matrimonio entre HTML y XML. Por una parte, es necesario utilizar el lenguaje de transformación XSLT para poder desplegar como HTML información que es guardada como XML. Por otra parte, lenguajes de consulta tales como XPath y XQuery son utilizados para extraer y analizar información que es almacenada en XML.

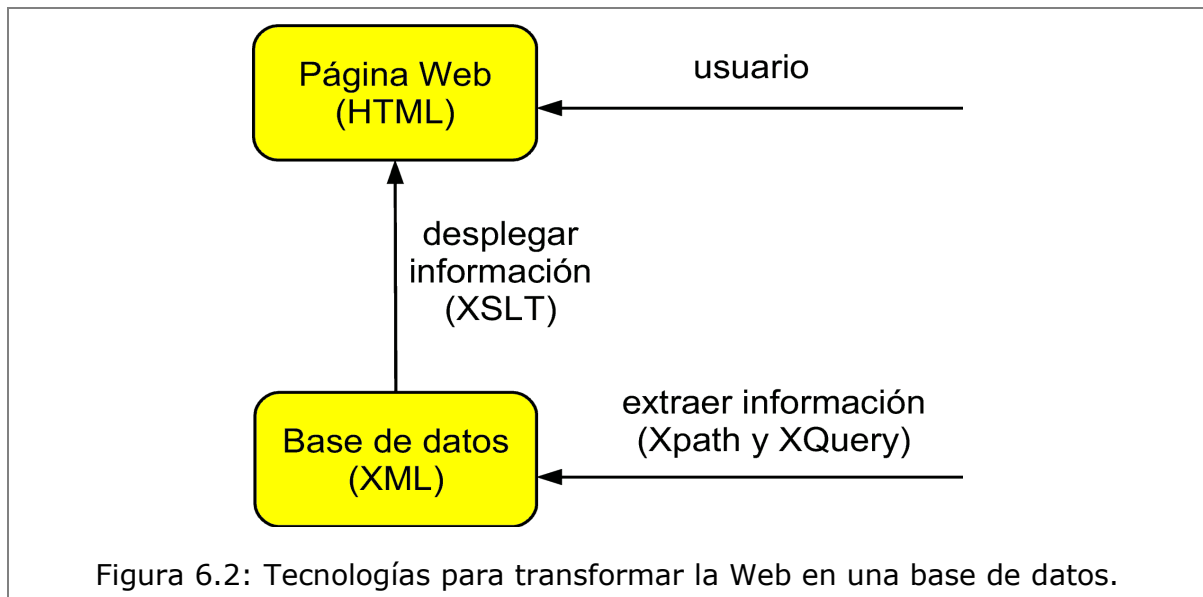


Figura 6.2: Tecnologías para transformar la Web en una base de datos.

Para saber más

- ◆ El sitio de la World Wide Web Consortium o simplemente W3C (<http://www.w3.org/>) es un buen lugar para informarse de los avances en las tecnologías Web como XML.
- ◆ El sitio <http://www.w3schools.com/> tiene tutoriales sobre HTML, XML, XSLT, XPath, Xquery, etc.

Referencias

1. S. Boag, D. Chamberlin, M. F. Fernández, D. Florescu, J. Robie y J. Siméon. XQuery 1.0: An XML Query Language. Recomendación de la W3C, enero 2007, <http://www.w3.org/TR/xquery/>
2. T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, F. Yergeau. Extensible Markup Language (XML) 1.0. Recomendación de la W3C, agosto 2006, <http://www.w3.org/TR/2006/REC-xml-20060816/>
3. J. Clark. XSL Transformations (XSLT) Version 1.0. Recomendación de la W3C, noviembre 1999, <http://www.w3.org/TR/xslt>
4. J. Clark y S. DeRose. XML Path Language (XPath) Version 1.0. Recomendación de la W3C, noviembre 1999, <http://www.w3.org/TR/xpath>