Novática, revista fundada en 1975 y decana de la prensa informática española, es el órgano oficial de expresión y formación continua de **ATI** (Asociación de Técnicos de Informática). **Novática** edita también **Upgrade**, revista digital de **CEPIS** (*Council of European Professional Informatics Societies*), en lengua inglesa, y es miembro fundador de **UPENET** (*UPGRADE European NETwork*)

<http://www.ati.es/novatica/>
<http://www.upgrade-cepis.org/>

ATI es miembro fundador de **CEPIS** (*Council of European Professional Informatics Societies*) y es representante de España en **IFIP** (*International Federation for Information Processing*); tiene un acuerdo de colaboración con **ACM** (*Association for Computing Machinery*), así como acuerdos de vinculación o colaboración con **AdaSpain**, **AI2** y **ASTIC**.

Coordinación Editorial, Redacción Central y Redacción ATI Madrid
Padilla 66, 3º, dcha., 28006 Madrid
Tlfn.914029391; fax.913093685 <novatica@ati.es>
Composición, Edición y Redacción ATI Valencia
Av. del Reino de Valencia 23, 46005 Valencia
Tlfn./fax 963330392 <secreval@ati.es>
Administración y Redacción ATI Cataluña
Ciudad de Granada 131, 08018 Barcelona
Tlfn.934125235; fax 934127713 <secregen@ati.es>
Redacción ATI Andalucía
Isaac Newton, s/n, Ed. Sadiel,
Isla Cartuja 41092 Sevilla, Tlfn./fax 954460779 <secreand@ati.es>
Redacción ATI Aragón
Lagasca 9, 3-B, 50006 Zaragoza.
Tlfn./fax 976235181 <secreara@ati.es>
Redacción ATI Asturias-Cantabria <gp-astucant@ati.es>
Redacción ATI Castilla-La Mancha <gp-clmancha@ati.es>
Redacción ATI Galicia
Recinto Ferial s/n, 36540 Silleda (Pontevedra)
Tlfn.986581413; fax 986580162 <secregal@ati.es>
Suscripción y Ventas
<http://www.ati.es/novatica/interes.html>,o en ATI Cataluña o ATI Madrid
Publicidad
Padilla 66, 3º, dcha., 28006 Madrid
Tlnf.914029391; fax.913093685 <novatica.publicidad@ati.es>
Imprenta
Derra, S.A., Juan de Austria 66, 08005 Barcelona.
Depósito legal: B 15.154-1975 -- ISSN: 0211-2124; CODEN NOVAEC
Portada: Antonio Crespo Foix / © ATI 2005
Diseño: Fernando Agresta / © ATI 2005

Revista de la Asociación de Técnicos de Informática

UPGRADE

sumario

Nº 173, enero-febrero 2005, año XXXI

Monografía del próximo número: *"IPv6"*

Variaciones sobre XML

Carlos Delgado Kloos
Universidad Carlos III de Madrid

<cdk@it.uc3m.es>

Resumen: *se presenta un breve repaso a los conceptos principales de XML* (eXtensible Markup Language) *y se relacionan éstos con conceptos clásicos de programación, de forma que esta monografía sea autocontenida. El objetivo es entender mejor las aportaciones y limitaciones de este lenguaje.*

Palabras clave: *árboles, lenguajes funcionales,XML.*

Autor

Carlos Delgado Kloos obtuvo el título de Ingeniero de Telecomunicación en la Universidad Politécnica de Madrid (UPM) en 1978 y el de Doctor en Informática de la Universidad Técnica de Múnich (Alemania) en 1986. Actualmente es Catedrático de Ingeniería Telemática en la Universidad Carlos III de Madrid y Director del Departamento de Ingeniería Telemática, Director del Máster en Comercio Electrónico y Director de la Cátedra Nokia, en esta misma universidad. Sus intereses incluyen Lenguajes y Técnicas de Diseño para Sistemas Hardware y Software basados en métodos formales, así como aplicaciones basadas en Tecnología Internet, tales como la publicación electrónica, la tele-educación o el comercio electrónico. Ha liderado un buen número de proyectos de investigación tanto a nivel europeo, como nacional y bilateral (España-Alemania y España-Francia). Entre ellos cabe destacar que actua como coordinador del proyecto E-LANE de eLearning, financiado por la Unión Europea, en el que participan 5 instituciones europeas y otras tantas latinoamericanas. Ha publicado más de 120 artículos científicos en congresos y revistas nacionales e internacionales. Además ha escrito un libro y co-editado otros cuatro. Entre los cargos que ha ocupado u ocupa se encuentran los siguientes: Vice-presidente de la Junta Directiva Estatal de la Asociación de Técnicos de Informática, representante español y vicepresidente del comité técnico nº 10 de IFIP, secretario del grupo de trabajo nº 10.5 de IFIP, miembro del Consejo editorial de la revista 'Formal Aspects of Computing' publicada por Springer-Verlag, subdirector de Ingeniería de Telecomunicación en la Universidad Carlos III de Madrid, gestor del Programa Nacional de Tecnologías de la Información y las Comunicaciones en el Ministerio de Ciencia y Tecnología español y miembro de comités de programa de más de 70 congresos, entre los que cabe resaltar la vicepresidencia del Comité de Programa del Congreso Mundial de Informática de IFIP en el año 1992 y la presidencia del Comité de Programa de DATE 2002, Telecom I+D 2003, EduTech2004 y EUNICE2005. Pertenece a diversas asociaciones extranjeras y españolas, entre ellas ATI, y es frecuente colaborador de ***Novática***, revista de cuyo Consejo Editorial fue miembro, y de ***UPGRADE***.

1. Introducción

¿Qué es XML (*eXtensible Markup Language*)?

Entre las muchas definiciones que nos encontramos en la red, elegimos ésta:

"*XML es el acrónimo de eXtensible Markup Language, el formato universal para documentos y datos estructurados en la Red. Es un dialecto simplificado de SGML (Standard Generalized Markup Language) que proporciona un metalenguaje que contiene reglas para construir lenguajes de marcado especializados*".

En la definición se habla de documentos, marcado, metalenguaje. Nos preguntamos, ¿qué tendrá que ver esto con los lenguajes de programación clásicos, si es que tiene algo que ver? Y, ¿hasta dónde podemos llegar con XML? o ¿cuál es la esencia de XML?

Profundicemos más allá de las metáforas y parábolas que se nos ofrecen con nomenclatura especializada del ámbito de los documentos. Liberemos del ámbito de aplicación inicial, para estudiar las características del lenguaje, Analicemos en primer lugar su peculiar sintaxis, para a continuación estudiar las expresiones que somos capaces de construir con XML.

2. Sintaxis

Presentemos en primer lugar un documento XML que nos va a servir de ilustración en todo el artículo. Se trata de un documento en el que se describen las propiedades de un libro:

```
<libro>
  <titulo>XML for Dummies</titulo>
  <personas>
    <autor>Ed Tittel</autor>
    <autor>Norbert Mikula</autor>
    <autor>Ramesh Chandak</autor>
  </personas>
    <publicado editorial="Hungry Minds"
pastas="blandas"/>
</libro>
```

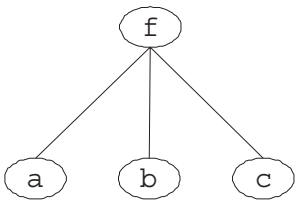
Nos encontramos estructuras de la forma **<f>a b c</f>**, en las que hay parejas de etiquetas (una de comienzo **<f>** y otra de final **</f>**) entre las que se encuentra cierto contenido, que recursivamente puede contener parejas (bien formadas) de etiquetas o bien texto plano. En el fondo, es equivalente escribir:

```
<f>
    a b c
</f>
```

en sintaxis XML, que esta otra expresión al estilo de LaTeX

```
\begin{f}
    a b c
\end{f}
```

La diferencia es "azúcar sintáctico". En ambos casos, se puede representar este código por el árbol



Esta notación, que podemos llamar *circumfijo*, incluye el identificador delante y detrás de los argumentos.

Otras variantes que nos encontramos en LaTeX utilizan la notación *prefijo* en lugar de la de arriba. Por ejemplo:

```
    \f{a b c}
o
    {\f a b c}
```

que nos recuerda mucho a las *s-expressions* (o expresiones simbólicas) de Lisp:

```
    (f a b c)
```

Como en Lisp tanto datos como programas son secuencias, podemos aplicar un programa Lisp (una secuencia) a otro (otra secuencia) y así obtener una nueva secuencia (un programa o un dato). La autoaplicación permitía la modificación automática de programas, lo cual llevaba a interesantes expe-

> ¿Cuál es la esencia de XML?

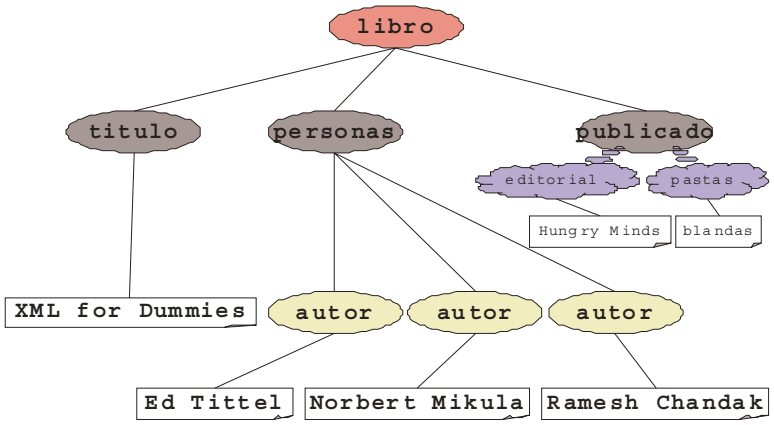


Figura 1. Representación en árbol.

rimentos. Observemos que con XML hacemos juegos parecidos: aplicamos un programa XSLT (e*Xtensible Stylesheet Language Transformations*, una aplicación XML) a un documento escrito en otra aplicación XML para obtener un documento, posiblemente conforme a una tercera aplicación XML.

La ventaja de tener una única sintaxis es que se pueden utilizar las mismas herramientas (para editar, analizar, manipular, transformar, etc.) en diversos contextos.

Hay quien ha criticado la notación XML como verbosa en exceso, sugiriendo que con una variante prefijo como la que se muestra a continuación habría sido suficiente. La obligación de repetir el elemento al final puede ayudar a un mejor entendimiento y a evitar errores, pero realmente no es frecuente que el documento se escriba a mano.

```
<libro>
  <titulo>XML for Dummies</>
  <personas>
    <autor>Ed Tittel</>
    <autor>Norbert Mikula</>
    <autor>Ramesh Chandak</>
  </>
    <publicado editorial="Hungry Minds"
pastas="blandas"/>
</>
```

3. Computación

Pero entonces, ¿XML no es más que Lisp con una nueva sintaxis? Ciertamente, no. En los ejemplos de arriba, el elemento **f** en el caso de XML es la marca que le asociamos a la secuencia **a b c**, y en el caso de Lisp es la función que aplicamos a la secuencia de argumentos **a b c**. En el caso de Lisp se le asocia un significado a ese símbolo (de función) **f**, por medio de una definición de función en el propio programa, de forma que podemos interpretar directamente la aplicación de esa **f** en una expresión.

En el caso de XML se difiere esa interpretación. Interesa en primera instancia crear simplemente el árbol, para más adelante asociarle una o varias interpretaciones. Si pensamos en una visualización para un usuario, esta interpretación será una interpretación de presentación (por ejemplo, por medio de una traducción por medio de XSLT a HTML, *HyperText Markup Language*, para su visualización en un navegador).

En si mismo, la sintaxis de XML no impediría asociar significado a los elementos en el propio documento. Este ejercicio se ha realizado en una serie de propuestas, tales como [1] o [6]. En esta última, se definen los lenguajes ConciseXML [2] y Water [7], que permiten la asociación de abstracciones lambda a símbolos de forma que se puedan realizar computaciones. Veamos un ejemplo:

```
<!— define el método 'factorial'—>
<defmethod factorial n>
   <if> n.<is 0/>    1
   else n.<times <factorial n.<minus 1/> /> />
   </if>
</defmethod>
<!— llama a 'factorial' con argumento 7—>
<factorial 7/>
```

Para entender mejor las expresiones de arriba, tengamos en cuenta que en ConciseXML los tipos de los valores de atributos pueden ser cualesquiera, no sólo *strings*, y que no hace falta poner los nombres de los atributos, pues se utiliza la posición para saber a qué atributo nos estamos refiriendo. Incluso los autores van más allá y definen servicios web sin abandonar el lenguaje:

```
<!— sirve el servicio web 'factorial' en el puerto 8383—>
<server factorial port=8383/>

<!— abre navegador y llama el servicio web 'factorial' —
>
<open_browser_window  "http://localhost:8383/?n=7"/
>

<!— define llamada a 'factorial' y llama el servicio —>
<web "http://localhost:8383/?n=7"/>.<execute/>
```

Éste es un enfoque contrario al seguido por el conjunto de lenguajes SOAP (*Simple Object Access Protocol*), WSDL (*Web Services Description Language*) y UDDI (*Universal Description, Discovery and Integration*), en el que se definen lenguajes separados para aspectos independientes. Si algo se observa aquí es que esta notación verbosa de XML no es realmente la mejor para representar computaciones, y eso que ConciseXML es el resultado de simplificar enormemente la sintaxis de XML.

Ésa es la conclusión a la que han llegado los autores de Curl [3][4], al proponer un lenguaje parecido a Lisp en su sintaxis (pero con llaves en lugar de paréntesis), que permite además de programar, dar instrucciones de formateo al estilo de HTML o LaTeX. Así el texto fuente

Podemos escribir una palabra en {italic itálica} y hacer cálculos como el siguiente: 12+3 es igual a {+ 12 3}.

se presenta interpretado como:

Podemos escribir una palabra en *itálica* y hacer cálculos como el siguiente: 12+3 es igual a 15.

en donde la función **italic** produce el efecto

> Los lenguajes funcionales tienen una notación concisa y cómoda para definir tipos de datos

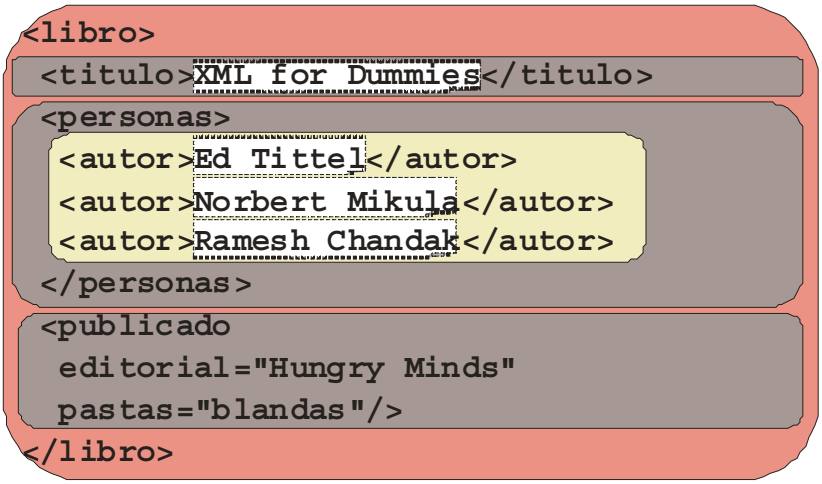


Figura 2. Representación en cajas anidadas.

de presentar en cursiva el contenido (el argumento) y la función + simplemente suma: presentación y computación en una misma notación uniforme. Con HTML y JavaScript u otros lenguajes de *scripting* se consigue el mismo efecto, pero esto supone la utilización de dos lenguajes de características y sintaxis bien distintas.

4. Estructuras de datos

Por tanto, XML podría haber sido perfectamente un lenguaje de programación, pero se ha quedado en un lenguaje para definir datos, árboles en concreto, o si se quiere ver de otra forma, de listas anidadas. Nuestro ejemplo inicial se puede representar gráficamente por medio del árbol de la **figura 1**. O bien, por medio de las cajas anidadas de la **figura 2**.

Bien es sabido que los lenguajes funcionales, como Haskell o Gofer, tienen una notación concisa y cómoda para definir tipos de datos. Estudiemos qué relación hay entre una declaración de elementos (lo que se llama *Document Type Definition*, o DTD, siguiendo la tradición de SGML del mundo de las editoriales) y la declaración de tipos de datos en Haskell [5].

Un posible DTD para el ejemplo de los libros podría ser el siguiente:

```
<!ELEMENT libro (titulo, personas, publicado?)>
<!ELEMENT titulo #PCDATA>
<!ELEMENT personas (autor* | editor*)>
<!ELEMENT autor #PCDATA>
<!ELEMENT editor #PCDATA>
<!ELEMENT publicado EMPTY>
  <!ATTLIST publicado
    editorial CDATA          #IMPLIED
    pastas    (duras | blandas) #REQUIRED>
```

Para cada elemento, hay una declaración de su contenido (**!ELEMENT**) y posiblemente de su lista de atributos (**!ATTLIST**). El contenido se especifica por medio de secuencias (,), alternativas (|), repeticiones (* o +) y opciones (?).

Se permite el anidamiento de elementos y en última instancia tendremos texto plano (**#PCDATA**). Así, el contenido del elemento **personas** (es decir, lo que nos podemos encontrar entre una etiqueta **<personas>** y una etiqueta **</personas>**) es o bien una secuencia de elementos **autor** o una secuencia de elementos **editor**. Los atributos tienen nombre y valor. Los valores pueden ser cadenas de caracteres (**CDATA**) o pertenecer a tipos enumerativos definidos de manera ad-hoc. Los atributos pueden ser obligatorios (**#REQUIRED**) u optativos (**#IMPLIED**).

La definición de un tipo de datos en Haskell correspondiente a este DTD podría tener la siguiente forma:

```
data Libro = Libro Titulo Personas (Maybe Publicado)
data Titulo = Titulo String
data Personas = PersonasAutor [Autor]
              | PersonasEditor [Editor]
data Autor = Autor String
data Editor = Editor String
data Publicado = Publicado PublicadoAt
data PublicadoAt = PublicadoAt {
   editorial :: Maybe String,
   pastas    :: Duras | Blandas }
```

Como se observa, la correspondencia no puede ser más directa. Cada declaración de elemento se corresponde con la declaración de un nuevo tipo en Haskell. La secuencia (,) se corresponde con un producto de tipos, es decir, valores con un constructor, que se representa por simple yuxtaposición de los tipos (no hay símbolo de separación). La alternativa (|) se corresponde con la unión de tipos, es decir, valores con varios constructores. El símbolo correspondiente también es la barra vertical (|). La repetición (*) se corresponde con la lista ([ ]) y la opción (?) con **Maybe**. Las listas de atributos también se traducen con facilidad. Los atributos no están ordenados secuencialmente como ocurre con las secuencias de elementos. Los atributos se identifican por el nombre en lugar de por la posición.

Por tanto, se pueden usar las etiquetas de campos de Haskell. Los atributos opcionales tienen **Maybe** en su tipo y si los valores están restringidos a un determinado conjunto, se define el correspondiente tipo enumerativo (|).

El documento XML de arriba se corresponde con el siguiente valor en Haskell de tipo **Libro**:

```
Libro (Titulo "XML for Dummies")
    PersonasAutor [(Autor "Ed Tittel"),
                   (Autor "Norbert Mikula"),
                   (Autor "Ramesh Chandak")]
    Publicado (PublicadoAt
              {editorial="Hungry Minds",
               pastas=Blandas})
```

La correspondencia entre Haskell y DTDs no es total; hay algunas diferencias. Además del cambio de notación (prefijo/circumfijo), en el caso de Haskell se utilizan los constructores para formar el valor, mientras que en el caso de los DTDs no existe el concepto de constructor como tal y se utiliza el tipo. Esta diferenciación es especialmente importante en el caso de la unión de tipos (|).

Por completitud incluimos también el esquema XML correspondiente:

> La definición de los tipos se realiza de forma separada de los valores de esos tipos

```
<xs:element name="libro">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="titulo" type="string"/>
            <xs:element name="personas"
type="personasType"/<
          <xs:element name="publicado"
type="publicadoType"
     minOccurs="0" maxOccurs="1"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>

<xs:complexType name=»personasType»>
 <xs:choice>
  <xs:element name="autor" type="string"
   minOccurs="0" maxOccurs="unbounded"/>
  <xs:element name="editor" type="string"
   minOccurs="0" maxOccurs="unbounded"/>
 </xs:choice>
</xs:complexType>

<xs:complexType name="publicadoType">
 <xs:attribute name="editorial"
  type="string" use="optional"/>
 <xs:attribute name="pastas"
  type="pastasType" use="required"/>
</xs:complexType>

<xs:simpleType name="pastasType">
 <xs:restriction base="xs:string">
  <xs:enumeration value="duras"/>
  <xs:enumeration value="blandas"/>
 </xs:restriction>
</xs:simpleType>
```

Vemos que se pierde la concisión de los DTDs con respecto a los esquemas XML. Además se abandona la notación de expresiones regulares por otra más intuitiva y verbosa. Sin embargo, se gana en otros aspectos: sintaxis XML, un sistema de tipos de datos más rico, modelo de contenido abierto, etc.

5. Conclusión

Por tanto, podemos dar finalmente una definición más técnica de XML. Definimos XML como un lenguaje funcional de definición de tipos de datos con notación circumfijo y sin constructores propiamente dichos (o, mejor dicho, utilizando los nombres de los tipos como constructores).

La definición de los tipos se realiza de forma separada de los valores de esos tipos (la primera es el DTD y la segunda un documento concreto). También se puede asociar una semántica a los constructores, pero esto se hace nuevamente de forma separada. Primero se trabaja con el álgebra inicial y luego por medio de otros mecanismos (típicamente de transformación con XSLT) se puede traducir a estructuras directamente interpretables (por ejemplo a HTML). Con estas breves reflexiones sobre XML hemos querido referir los nuevos conceptos y la nueva jerga introducidos por XML a conceptos clásicos de programación.

En muchas ocasiones, cuando se define un nuevo paradigma o lenguaje, se definen nuevas metáforas y se usan denominaciones desconocidas y por ello es difícil distinguir las nuevas aportaciones respecto de lo existente. En el caso de XML, además se hereda nomenclatura del ámbito de las editoriales, pues no olvidemos que SGML, el predecesor de XML, proviene de ese mundo. Con estas reflexiones esperamos haber puesto XML en contexto y aclarado algunas de sus construcciones.

Agradecimientos

Agradezco a Bernhard Möller sus comentarios a versiones preliminares de este artículo. Este trabajo ha sido realizado en el contexto del proyecto SIEMPRE (TIC2002-03635), financiado por el Programa Nacional de Tecnologías de la Información y las Comunicaciones de la Comisión Interministerial de Ciencia y Tecnología.

Referencias

[1] Peter T. Breuer, Carlos Delgado Kloos, Vicente Luque Centeno, Luis Sánchez Fernández. "Higher Order Applicative XML Documents", *Lecture Notes in Computer Science* 2941, pp. 91-107. Heidelberg, Springer-Verlag, 2004.

[2] ConciseXML. <http://www.concisexml. org> [consultado nov. 2004].

[3] Curl. <http://www.curl.com> [consultado nov. 2004].

[4] Bruce Mount, Gary Gray, Nikhil Damle. *Curl Programming Bible*. Wiley, 2002.

[5] Simon Peyton Jones. *Haskell 98 Language and Libraries*. Cambridge, Cambridge University Press, 2003.

[6] Mike Plusch. *Water: Simplified Web Services and XML Programming*. Indianapolis, Indiana. Wiley. 2002.

[7] Water, lenguage para servicios Web. <http://www.waterlanguage.com> [consultado nov. 2004].

[8] Malcolm Wallace, Colin Runciman. *Haskell and XML: Generic Combinators or Type-Based Translation?*, Proc. International Conference on Functional Programming, ACM, Paris, Sept 1999, <http://www.cs.york.ac.uk/fp/HaXml/icfp99.html> [consultado nov. 2004].