CEU | Universidad San Pablo

XML

Sistemas Web II

Grado en Ingeniería de Sistemas de Información

Álvaro Sánchez Picot
alvaro.sanchezpicot@ceu.es

v20240227

Basado en el trabajo de:

- David González Márquez

Índice

INTRODUCCIÓN A XML

Formato de datos

- Información desestructurada (unstructured)
  - La información no está organizada de una forma predefinida
  - Ej: Archivo de texto, email, media (imágenes, audio, videos)…
  - Corresponde con el 70-80% de la información en las organizaciones
- Información estructurada
  - Modelo de datos
  - ej: base de datos, modelos semánticos…

Formato de datos

- Información semi-estructurada (semi-structured)
  - Marcadores (p. ej.: etiquetas) para indicar la estructura
  - Estructura autodescriptiva (self-describing)
  - Ej: CSV, XML, JSON

Lenguaje de marcado

- “Markup language”
- Sistema para anotar documentos
- Distinguible del texto (contenido)
- No se muestra cuando se procesa
- Sirve para formatear el texto
- Importa el orden
- Ejemplos: TeX (LaTeX), HTML, SGML

Lenguaje de marcado – LaTeX

```
\documentclass{article}
\usepackage[utf8]{inputenc}
\title{An interesting theory}
\author{Juan Perez}
\date{April 2020}
\usepackage{natbib}
\usepackage{graphicx}
\begin{document}
    \maketitle
    \section{Introduction}
        There is a theory which states that if ever anyone discovers
(...)
        \begin{figure}[h!]
            \centering
            \includegraphics[scale=1.7]{universe}
            \caption{The Universe}
            \label{fig:universe}
        \end{figure}
    \section{Conclusion}
        ''I always thought something was fundamentally wrong with the
universe'' \citep{adams1995hitchhiker}
        \bibliographystyle{plain}
        \bibliography{references}
\end{document}
```

An interesting theory

Juan Perez

April 2020

1 Introduction

There is a theory which states that if ever anyone discovers exactly what the Universe is for and why it is here, it will instantly disappear and be replaced by something even more bizarre and inexplicable. There is another theory which states that this has already happened.

Figure 1: The Universe

2 Conclusion

"I always thought something was fundamentally wrong with the universe" [1]

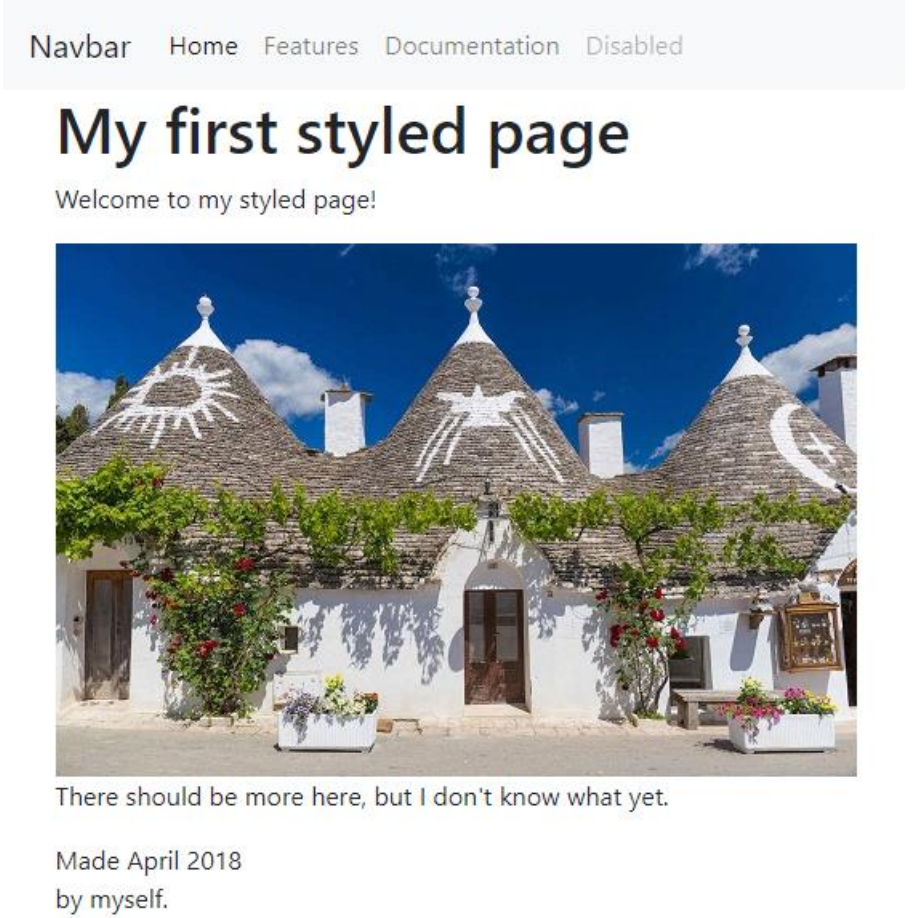
References

[1] D. Adams. *The Hitchhiker's Guide to the Galaxy*. San Val, 1995.

Lenguaje de marcado – HTML

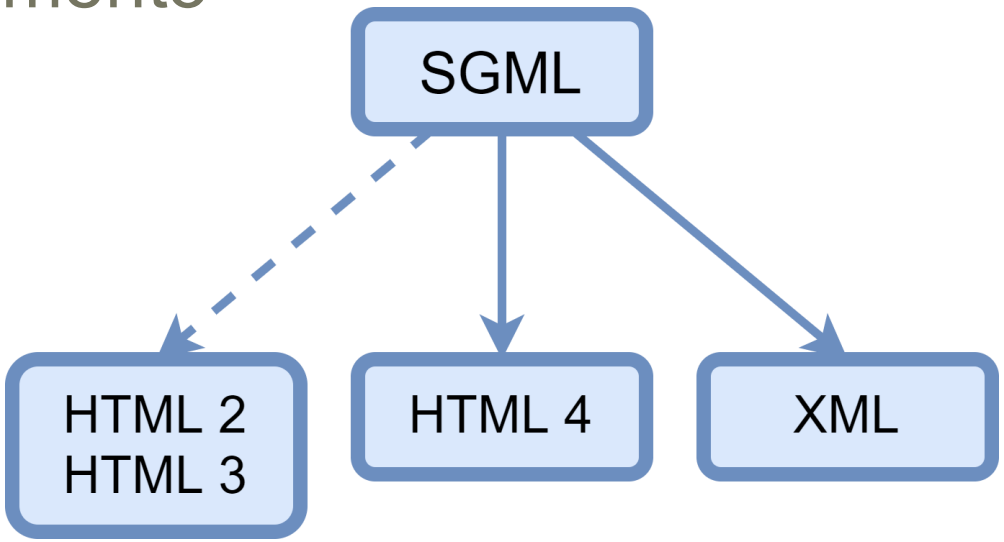
Archivo html_ejemplo.xml:

```
<!DOCTYPE html>
<html>
  <head>
    <title> A Simple HTML Document </title>
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.mi
n.css">
  </head>
  <body>
    <!-- Site navigation menu -->
    <nav class="navbar navbar-expand-lg navbar-light bg-light"> <!--
Navigation bar --></nav>
    <!-- Main content -->
    <div class="container">
    <h1>My first styled page</h1>
    <p>Welcome to my styled page!</p>
    <img src="https://www.w3schools.com/html/pic_trulli.jpg">
    <p>There should be more here, but I don't know what yet.</p>
    <address>Made April 2018<br> by myself.</address>
    </div>
  </body>
</html>
```



SGML

- Standard Generalized Markup Language
- Estándar para definir lenguajes de marcado para documentos
- Información estructural y semántica
- No define la presentación del documento
- Norma ISO 8879:1986(en)

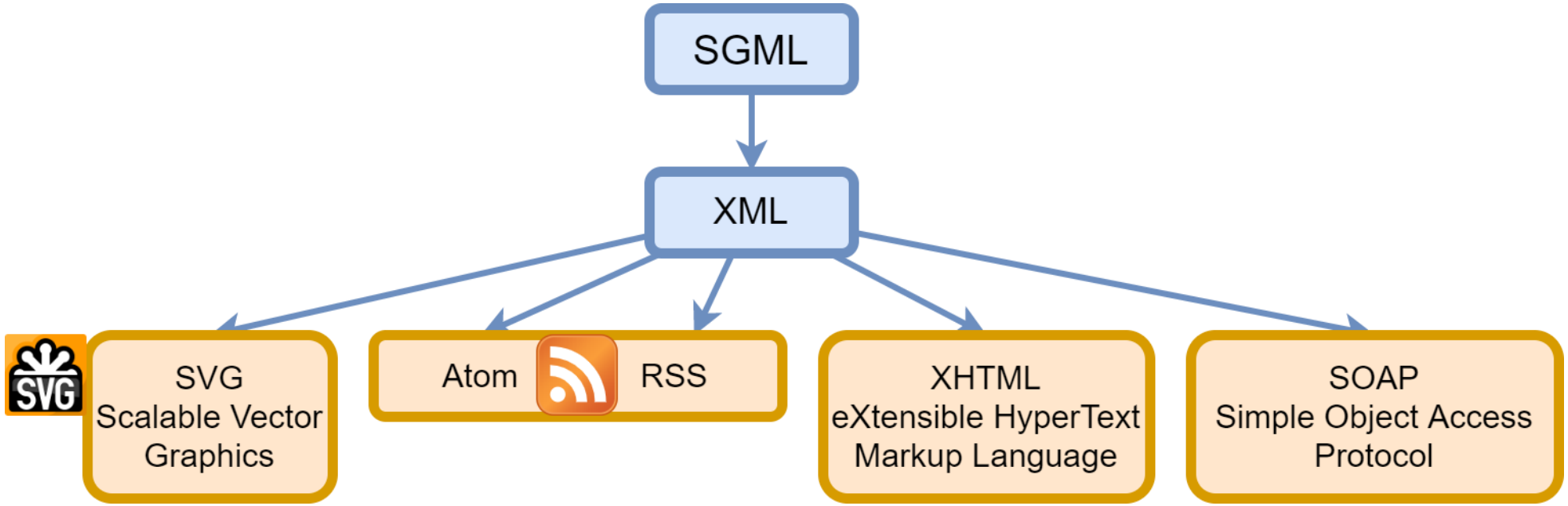


El lenguaje XML

- “Extensible Markup Language”
- Lenguaje de marcado con etiquetas
- Las etiquetas:
  - Describen el contenido
  - No describen la forma de mostrar el contenido
  - No están predefinidas
- Tiene estructura de árbol
- Estándar para la representación y envío de información
- Human and machine readable
- Recomendación de la W3C

El lenguaje XML

- Es una extensión del SGML
- Diferencias entre XML y SGML
- Lista de lenguajes derivados de XML

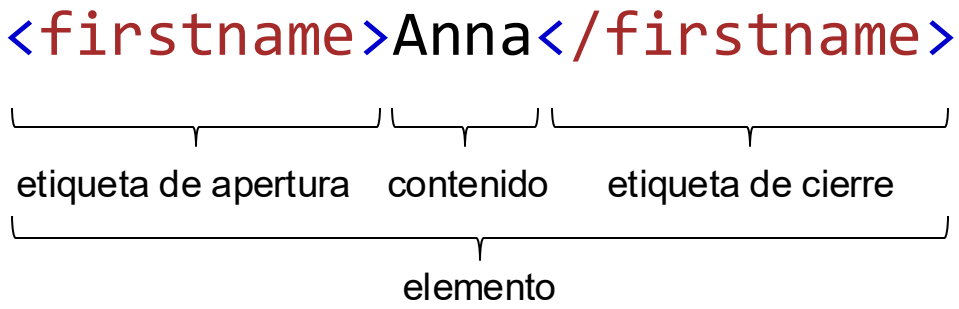


El lenguaje XML

- Usado en diferentes sectores:
  - Negocios
  - Transacciones financieras
  - Datos médicos
  - Datos matemáticos
  - Mediciones científicas
  - Información de noticias
  - Servicios meteorológicos
- Lista de lenguajes derivados del XML según su propósito

Reglas de XML

- **Etiqueta**: marca sobre un elemento
  - Delimitada por “<“ y “>”
  - Se cierra con “/” detrás de “<“
  - Exactamente el mismo nombre en la etiqueta de apertura y la de cierre
- **Contenido**: zona entre la etiqueta de inicio y la de final
  - Ejemplo: `<Title>Contenido</Title>`
- **Elemento**: etiqueta de inicio, contenido y etiqueta de cierre



Reglas de XML

- **Atributo**: información adicional asociada a una etiqueta
  - Formato: `nombre="valor"`
  - El valor delimitado siempre por las comillas ("...") o apóstrofes ('...')
  - Cada elemento sólo puede tener un atributo con el mismo nombre
  - Siempre tiene un valor asociado

    `<book price="100">` ~~`<book price>`~~
- **Comentario**: no se procesa
  - Entre “<!--” y “-->”
  - No puede contener “--”
- Se distinguen mayúsculas de minúsculas

Reglas de XML

- **Todas** las etiquetas abren y cierran

```
<book>   </book>
<remark />
```

- Escapando caracteres especiales

&lt;	<	less-than	menor que
&gt;	>	greater-than	mayor que
&amp;	&	ampersand	et
&apos;	'	apostrophe	apóstrofe
&quot;	"	double-quote	comillas

Reglas de XML

- **Nombre de las etiquetas y atributos:**
  - Case sensitive
  - Compuesto de caracteres Unicode
  - No puede contener: !"#$%&'()*+,/;<=>?@[\]^`{|}~
  - No puede contener espacios
  - No puede empezar por: -, . o número

Reglas de XML

XML prolog

- Opcional
- Si existe, tiene que estar al principio, antes del elemento raíz
- No es parte del documento XML
  - No es un error que no tenga etiqueta de cierre

```
<?xml version="1.0" encoding="UTF-8"?>
```

- Define versión XML
- Define la codificación de caracteres

Reglas de XML

- Estructura en árbol
  - Document Object Model (DOM)
- Elemento raíz (root)
  - Tiene que ser único
- Todos los elementos, excepto el raíz, tienen padre (parent)
- Los elementos pueden tener hijos (child)
- Los elementos que no tienen hijos son las hojas (leaves)
- Dos elementos que comparten padre son hermanos (sibling)

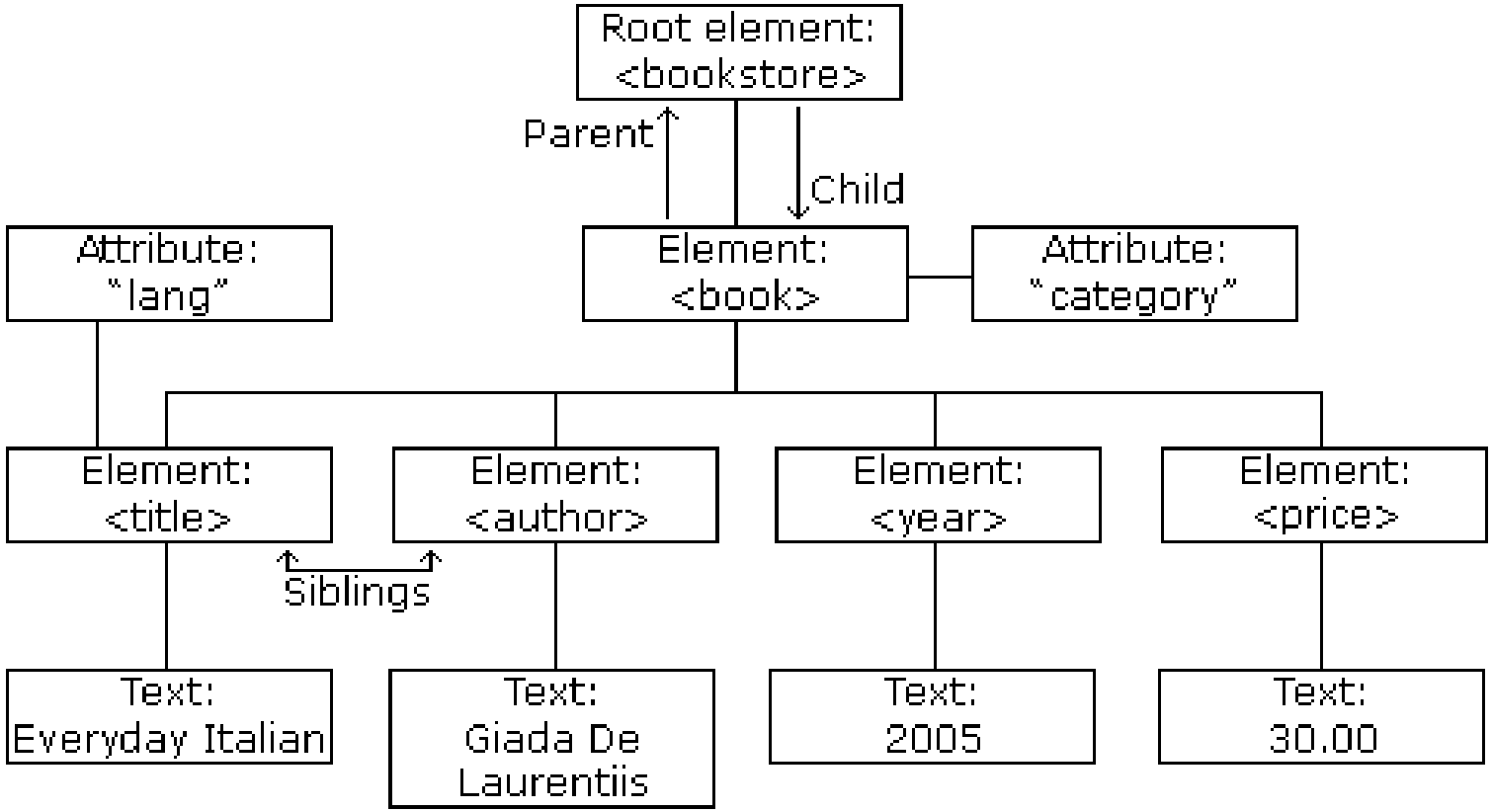
Introducción a XML

Archivo xml_ejemplo1.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
    <book category="cooking">
        <title lang="en">Everyday Italian</title>
        <author>Giada De Laurentiis</author>
        <year>2005</year>
        <price>30.00</price>
    </book>
    <book category="children">
        <title lang="en">Harry Potter</title>
        <author>J K. Rowling</author>
        <year>2005</year>
        <price>29.99</price>
    </book>
    <book category="web">
        <title lang="en">Learning XML</title>
        <author>Erik T. Ray</author>
        <year>2003</year>
        <price>39.95</price>
    </book>
</bookstore>
```

¿Cuál es la estructura de árbol?

Reglas de XML



Elementos vs Atributos

```
<note>
  <date>
    <year>2008</year>
    <month>01</month>
    <day>10</day>
  </date>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this
weekend!</body>
</note>
```

```
<note
  day="10"
  month="01"
  year="2008"
  to="Tove"
  from="Jani"
  heading="Reminder"
  body="Don't forget me this
weekend!">
</note>
```

Elementos vs Atributos

- Problemas de los atributos:
  - No pueden contener múltiples valores
  - No se pueden expandir en un futuro
  - No pueden describir estructuras
- Ventajas de los atributos
  - Legibilidad
  - Rendimiento
- En general, usar atributos únicamente para metadatos

https://www.w3schools.com/xml/xml_dtd_el_vs_attr.asp

Modelo relacional vs XML

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
    <book category="cooking">
        <title lang="en">Everyday Italian</title>
        <author>Giada De Laurentiis</author>
        <year>2005</year>
        <price>30.00</price>
    </book>
    <book category="children">
        <title lang="en">Harry Potter</title>
        <author>J K. Rowling</author>
        <year>2005</year>
        <price>29.99</price>
    </book>
    <book category="web">
        <title lang="en">Learning XML</title>
        <author>Erik T. Ray</author>
        <year>2003</year>
        <price>39.95</price>
    </book>
</bookstore>
```

¿Cuál sería el equivalente en el modelo relacional?

Modelo relacional vs XML

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
    <book category="cooking">
        <title lang="en">Everyday Italian</title>
        <author>Giada De Laurentiis</author>
        <year>2005</year>
        <price>30.00</price>
    </book>
    <book category="children">
        <title lang="en">Harry Potter</title>
        <author>J K. Rowling</author>
        <year>2005</year>
        <price>29.99</price>
    </book>
    <book category="web">
        <title lang="en">Learning XML</title>
        <author>Erik T. Ray</author>
        <year>2003</year>
        <price>39.95</price>
    </book>
</bookstore>
```

Books

id	Title	Author	Year	Price	cat_id	lan_id
1	Everyday Italian	Giada De Laurentiis	2005	30.00	1	1
2	Harry Potter	J K. Rowling	2005	29.99	2	1
3	Learning XML	Erik T. Ray	2003	39.95	3	1

Categories

Id	name
1	cooking
2	children
3	web

lang

Id	name
1	en

Modelo relacional vs XML

	Modelo relacional	XML
Estructura	Tablas	Jerárquica en arbol
Esquema	Fijado con antelación	Flexible Auto-descriptivo
Consultas	Sencillas Intuitivas	Algo más complicadas
Ordenación	Ninguna	Implícita
Mejor para	Ordenador Almacenamiento	Humanos Streaming

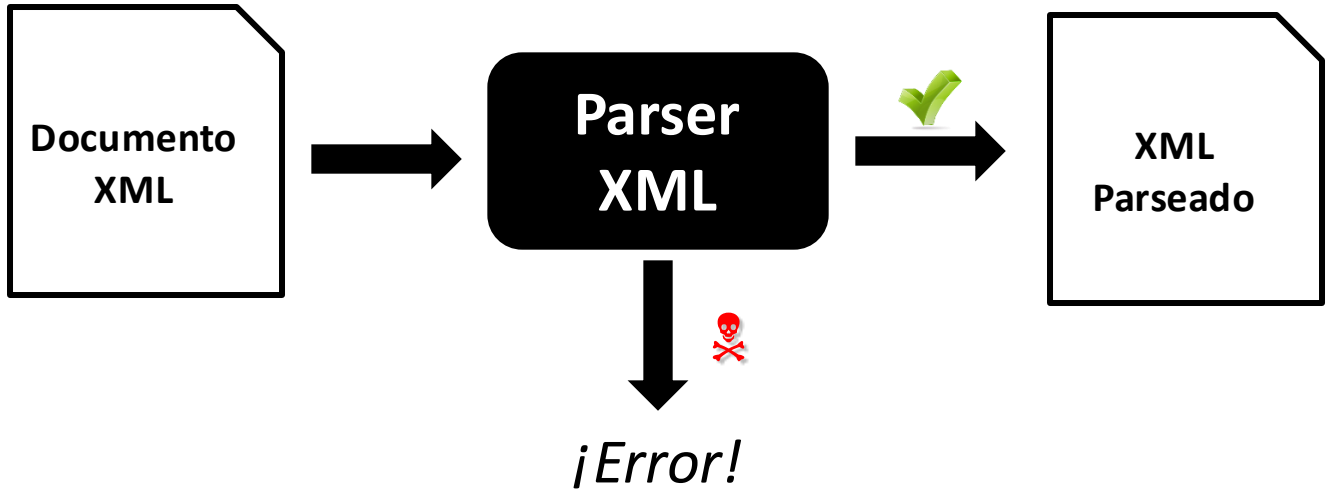
XML Bien formado

- Un documento XML está **bien formado** (well-formed) si se adhiere a unos **requisitos estructurales básicos**:
  - Un único elemento raíz
  - Etiquetas emparejadas
  - Anidamiento apropiado
  - Atributos únicos dentro de cada elemento
  - Caracteres adecuados

XML Bien formado

- Analizadores (parsers):
  - Document Object Model (DOM)
  - Simple API for XML (SAX)
  - …
- Con XML no se permiten los errores

XML Bien formado



XML Bien formado

```
<breakfast_menu>
  <food>
    <name>French Toast</name>
    <name>French Toast</name>
    <price currency="$">4.50</price>
    <description>Thick slices made from our homemade sourdough bread</DESCRIPTION>
    <calories>600</calories>
  </food>
  <food>
    <price currency>6.95</price>
    <description>Two eggs, bacon or sausage, toast, and our ever-popular hash browns
  <calories>950</calories>
  </food>
  <food></food>
</breakfastMenu>
<drinks_menu>
  <drink>
    <name>Coffe<name/>
    <price>1.5$</price>
  </drink></drinks_menu>
```

¿Está bien formado? ¿Por qué?

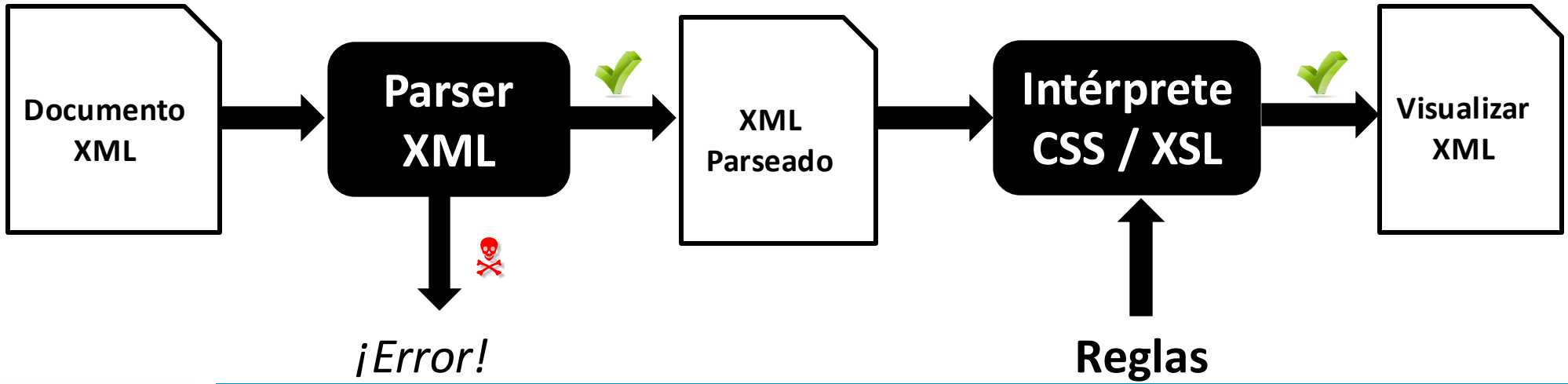
XML Bien formado

Herramientas para comprobar el XML:

- https://codebeautify.org/xmlvalidator
- https://www.w3schools.com/xml/xml_validator.asp
- http://www.xmlvalidation.com/
- http://www.xpathtester.com/

Mostrando XML

- Se utilizan lenguajes de reglas para transformar XML en HTML:
  - CSS (Cascading Style Sheets): Hojas de estilo en cascada
  - XSL (eXtensible Stylesheet Language): Lenguaje de hojas de estilo extensible
- Pasa previamente por el parser



Mostrando XML

Archivo xml_ejemplo2.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css" href="xml_ejemplo2.css"?>
<bookstore>
  <!-- Resto de contenido -->
</bookstore>
```

Archivo xml_ejemplo2.css:

```
bookstore {
    color: white;
    background-color : gray;
    padding: 20px;
}
book {    margin: 20px;}
title {
    color: green;
    font-size : 36px;
    font-weight : bold;
    background-color : powderblue;
}
author {    font-size : 24px;}
author:before {   content: "By: ";}
year:before {   content: "Year: ";}
price:after {   content: "€";}
title, author, year, price {
    display : block;
}
```

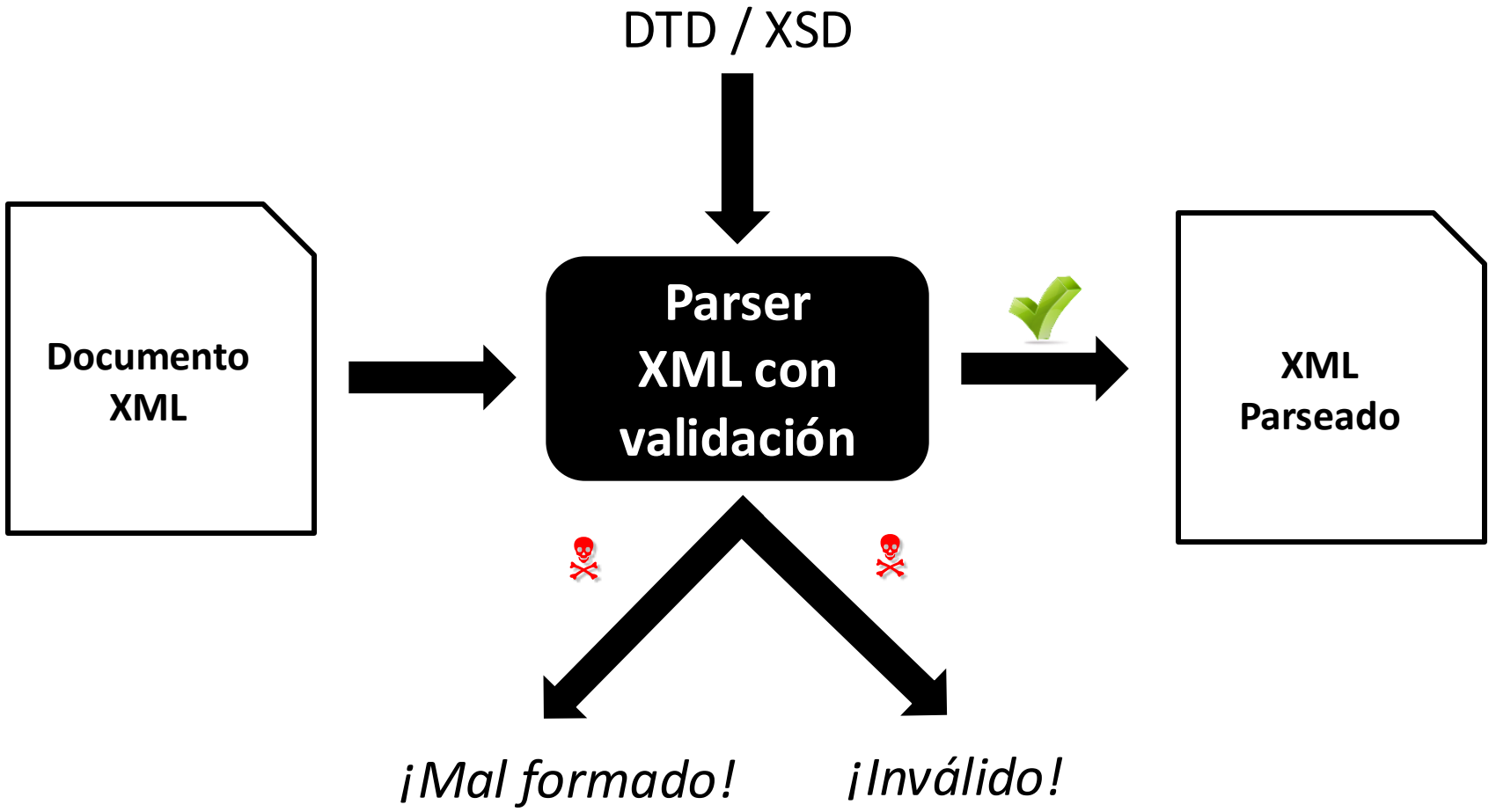
Estándares XML

- XML es el estándar de representación e intercambio de datos más extendido
- Cuenta con un gran número de estándares asociados:
  - DTD
  - XSD
  - XPath
  - Xquery
  - XSL

XML Válido

- Documento XML **bien formado** (well-formed) → cumple **requisitos estructurales** básicos
- Un documento XML es **válido** (valid):
  - Está bien formado
  - Se adhiere a unos **requisitos de contenido** especificados por:
    - **DTD**: Document Type Descriptor
    - **XSD**: XML Schema

XML Válido


DTD / XSD
Documento XML
Parser XML con validación
XML Parseado
¡Mal formado!
¡Inválido!

DTD Y XSD

DTD

- “Document Type Descriptor”
- Descriptor de tipo de documento
- Estándar para validar XML.
- Lenguaje que proporciona una gramática para especificar:
  - Elementos
  - Atributos
  - Anidado
  - Ordenación
  - Número de apariciones

DTD

```
<!DOCTYPE root-element [
<!ELEMENT root-element (…)>
<!ELEMENT element1 (…)>
<!-- Resto de elementos-->
<!ATTLIST element1 attribute1 attribute-type
attribute-value …>
<!-- Resto de atributos-->
]>
```

DTD – Elementos

```
<!DOCTYPE root-element [...]>
```

- **root-element:**
  - Nombre del elemento raíz
  - Tendrá que estar definido a continuación

DTD – Elementos

```
<!ELEMENT element-name category>
```

- **element-name:** Nombre del elemento
- **category:**
  - EMPTY: Vacío.
  - (#PCDATA): Parsed Character Data. Texto que se va a parsear
  - ANY: Cualquier combinación de datos parseables

DTD – Elementos

```
<!ELEMENT element-name category>
```

- **category (cont.):**
  - (child1,child2,...):
    - Los hijos tienen que aparecer en el mismo orden
    - Se puede especificar el número de ocurrencias:
      - +: 1 or more
      - *: 0 or more
      - ?: 0 or 1
      - Si no se especifica, implica uno y sólo uno
  - Se puede usar | para permitir una elección

DTD – Elementos

Ejemplos:

```
<!ELEMENT name (#PCDATA)> → <name>Ana</name>
<!ELEMENT email (from,to+,body)> →
   <email>
     <from>…</from>
     <to>…</to>
     <to>…</to>
     <body>…</body>
   </email>
```

DTD – Atributos

```
<!ATTLIST element-name attribute-name attribute-type
attribute-value [attribute-name attribute-type attribute-value]*>
```

- **element-name:** Nombre de elemento al que pertenece
- **attribute-name:** Nombre del atributo
- **attribute-type:**
  - CDATA: Character Data. Texto que no se va a parsear
  - (*en1*|*en2*|..): Sólo se permiten esos valores específicos
  - ID
  - IDREF...

DTD – Atributos

```
<!ATTLIST element-name attribute-name attribute-type
attribute-value [attribute-name attribute-type attribute-value]*>
```

- **attribute-value:**
  - *value,*
  - #REQUIRED: El atributo tiene que estar
  - #IMPLIED: El atributo es opcional
- `[attribute-name attribute-type attribute-value]*`
  - Se pueden añadir más atributos con la misma sintaxis
  - También se pueden definir en otra etiqueta ATTLIST

DTD Ejemplo

Archivo dtd_ejemplo1.xml:

```
<?xml version="1.0"?>
  <!DOCTYPE note [<!ELEMENT note
(to,from,heading,body)>
  <!ELEMENT to (#PCDATA)>
  <!ELEMENT from (#PCDATA)>
  <!ELEMENT heading (#PCDATA)>
  <!ELEMENT body (#PCDATA)>
]>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend</body>
</note>
```

Archivo dtd_ejemplo2.xml:

```
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM "dtd_ejemplo2.dtd">
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this
weekend!</body>
</note>
```

Archivo note.dtd

```
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

DTD Ejemplo

Archivo dtd_ejemplo3.xml:

```
<?xml version="1.0"?>
<!DOCTYPE Bookstore [
   <!ELEMENT Bookstore (Book | Magazine)*>
   <!ELEMENT Book (Title, Authors, Remark?)>
   <!ATTLIST Book ISBN CDATA #REQUIRED
                  Price CDATA #REQUIRED
                  Edition CDATA #IMPLIED>
   <!ELEMENT Magazine (Title)>
   <!ATTLIST Magazine Month CDATA #REQUIRED Year CDATA #REQUIRED>
   <!ELEMENT Title (#PCDATA)>
   <!ELEMENT Authors (Author+)>
   <!ELEMENT Remark (#PCDATA)>
   <!ELEMENT Author (First_Name, Last_Name)>
   <!ELEMENT First_Name (#PCDATA)>
   <!ELEMENT Last_Name (#PCDATA)>
]>
```

DTD Ejemplo

Archivo dtd_ejemplo3.xml (cont.):

```
<Bookstore>
  <Book ISBN="ISBN-0-13-713526-2" Price=" 100"
Edition="3rd">
    <Title>A First Course in Database Systems</Title>
    <Authors>
      <Author>
        <First_Name>Jeffrey</First_Name>
        <Last_Name>Ullman</Last_Name>
      </Author>
      <Author>
        <First_Name>Jennifer</First_Name>
        <Last_Name>Widom</Last_Name>
      </Author>
    </Authors>
  </Book>
  <!-- Continúa -->
  <Book ISBN="ISBN-0-13-815504-6" Price="100">
    <Title>Database Systems: The Complete
Book</Title>
    <Authors>
      <Author>
        <First_Name>Hector</First_Name>
        <Last_Name>Garcia-Molina</Last_Name>
      </Author>
      <Author>
        <First_Name>Jeffrey</First_Name>
        <Last_Name>Ullman</Last_Name>
      </Author>
      <Author>
        <First_Name>Jennifer</First_Name>
        <Last_Name>Widom</Last_Name>
      </Author>
    </Authors>
    <Remark>
      Buy this book bundled with "A First Course"
- a great deal!
    </Remark>
  </Book>
</Bookstore>
```

DTD Ejercicio 1

Crea un XML que sea válido para el siguiente DTD:

```
<!DOCTYPE TVSCHEDULE [
<!ELEMENT TVSCHEDULE (CHANNEL+)>
<!ELEMENT CHANNEL (BANNER,DAY+)>
<!ELEMENT BANNER (#PCDATA)>
<!ELEMENT DAY (DATE,(HOLIDAY|PROGRAMSLOT+)+)>
<!ELEMENT HOLIDAY (#PCDATA)>
<!ELEMENT DATE (#PCDATA)>
<!ELEMENT PROGRAMSLOT (TIME,TITLE,DESCRIPTION?)>
<!ELEMENT TIME (#PCDATA)>
<!ELEMENT TITLE (#PCDATA)>
<!ELEMENT DESCRIPTION (#PCDATA)>

<!ATTLIST TVSCHEDULE NAME CDATA #REQUIRED>
<!ATTLIST CHANNEL CHAN CDATA #REQUIRED>
<!ATTLIST PROGRAMSLOT VTR CDATA #IMPLIED>
<!ATTLIST TITLE RATING CDATA #IMPLIED>
<!ATTLIST TITLE LANGUAGE CDATA #IMPLIED>
]>
```

DTD Ejercicio 2

Crea un DTD que valide el siguiente XML:

```
<articles>
  <article id="x34675">
    <name>Apache Spark Architecture</name>
    <month>december</month>
    <author name="kay vennisla"/>
    <reviews lang=""/>
    <feedback > high rating</feedback>
    <reviews lang="de">The best content with diagrams</reviews>
  </article>
</articles>
```

DTD Ejemplo con punteros

Archivo dtd_ejemplo4.xml:

```
<?xml version="1.0"?>
<!DOCTYPE Bookstore [
   <!ELEMENT Bookstore (Book*, Author*)>
   <!ELEMENT Book (Title, Remark?)>
   <!ATTLIST Book ISBN ID #REQUIRED
                  Price CDATA #REQUIRED
                  Authors IDREFS #REQUIRED>
   <!ELEMENT Title (#PCDATA)>
   <!ELEMENT Remark (#PCDATA | BookRef)*>
   <!ELEMENT BookRef EMPTY>
   <!ATTLIST BookRef book IDREF #REQUIRED>
   <!ELEMENT Author (First_Name, Last_Name)>
   <!ATTLIST Author Ident ID #REQUIRED>
   <!ELEMENT First_Name (#PCDATA)>
   <!ELEMENT Last_Name (#PCDATA)>
]>
```

DTD Ejemplo con punteros

Archivo dtd_ejemplo4.xml (cont.):

```
<Bookstore>
   <Book ISBN="ISBN-0-13-713526-2" Price="100" Authors="JU JW">
      <Title>A First Course in Database Systems</Title>
   </Book>
   <Book ISBN="ISBN-0-13-815504-6" Price="85" Authors="HG JU JW">
      <Title>Database Systems: The Complete Book</Title>
      <Remark>
         Amazon.com says: Buy this book bundled with
         <BookRef book="ISBN-0-13-713526-2" /> - a great deal!
      </Remark>
   </Book>
   <Author Ident="HG">
      <First_Name>Hector</First_Name>
      <Last_Name>Garcia-Molina</Last_Name>
   </Author>
   <Author Ident="JU">
      <First_Name>Jeffrey</First_Name>
      <Last_Name>Ullman</Last_Name>
   </Author>
   <Author Ident="JW">
      <First_Name>Jennifer</First_Name>
      <Last_Name>Widom</Last_Name>
   </Author></Bookstore>
```

DTD

- Ejemplos de validadores online:

https://www.truugo.com/xml_validator/

https://www.xmlvalidation.com/

http://xmlvalidator.new-studio.org/

- Más información sobre DTD:

https://www.w3schools.com/xml/xml_dtd_intro.asp

XSD

- “XML Schema Definition”
- También conocido simplemente como "XML Schema"
- Estándar para validar XML (esp1, esp2)
- Basado en XML
- Lenguaje que proporciona una gramática para especificar:
  - Elementos, orden, anidación y número de apariciones
  - Atributos
  - Tipos de datos y valores por defecto
  - …

XSD

Archivo xsd_ejemplo1.xml:

```
<?xml version="1.0"?>
<note xmlns="https://www.w3schools.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:schemaLocation="https://www.w3schools.com/
xml xsd_ejemplo1.xsd">
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

Archivo xsd_ejemplo1.xsd:

```
<?xml version="1.0"?>
<xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="https://www.w3schools.com"
xmlns="https://www.w3schools.com"
elementFormDefault="qualified">
  <xs:element name="note">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="to" type="xs:string"/>
        <xs:element name="from" type="xs:string"/>
        <xs:element name="heading"
type="xs:string"/>
        <xs:element name="body" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

XSD – Schema

`<xs:schema [archivo_xsd]>…</xs:schema>`

- Elemento raíz
- Atributos:
  - `xmlns:xs="http://www.w3.org/2001/XMLSchema"`
    - Especifica el namespace
    - Todos los elementos y tipos de datos tienen que usar el prefijo xs (se puede cambiar pej xsi)
  - `xs:schemaLocation="location"`: Localización del archivo xsd
- `[archivo_xsd]`: En caso de definirse el xsd en otro archivo

XSD – Simple Element

- Un elemento que sólo contiene tipos de datos simples
  - No puede contener otros elementos ni atributos
- Hay 19 tipos de datos primitivos simples y otros 25 derivados
  - Se pueden componer para crear nuevos tipos de datos:
    - Restringiendo: mínimo, máximo, expresiones regulares, longitud…
    - Listando una secuencia de valores permitidos

XSD – Simple Element

- Tipos de datos principales:
  - xs:string
  - xs:decimal
  - xs:integer
  - xs:boolean
  - xs:date
  - xs:time

XSD – Simple Element

```
<xs:element name="xxx" type="yyy"/>
```

- xs: suponiendo el prefijo xs definido en la etiqueta schema
- xxx: Nombre del elemento
- yyy: Tipo de datos
- Otros atributos opcionales:
  - `default="valor"`: Valor por defecto si no se especifica uno
  - `fixed="valor"`: Valor fijo que no se puede cambiar

XSD – Simple Element

- Ejemplo de un fragmento XML:

```
<lastname>Fernández</lastname>
<dateborn>1970-03-27</dateborn>
<postal_code>28668</postal_code>
```

- Ejemplo de un fragmento XSD:

```
<xs:element name="lastname" type="xs:string"/>
<xs:element name="dateborn" type="xs:date"/>
<xs:element name="postal_code" type="xs:integer"/>
```

XSD – Simple Element

Constraint	Description
enumeration	Defines a list of acceptable values
fractionDigits	Specifies the maximum number of decimal places allowed. Must be equal to or greater than zero
length	Specifies the exact number of characters or list items allowed. Must be equal to or greater than zero
maxExclusive	Specifies the upper bounds for numeric values (the value must be less than this value)
maxInclusive	Specifies the upper bounds for numeric values (the value must be less than or equal to this value)
maxLength	Specifies the maximum number of characters or list items allowed. Must be equal to or greater than zero
minExclusive	Specifies the lower bounds for numeric values (the value must be greater than this value)
minInclusive	Specifies the lower bounds for numeric values (the value must be greater than or equal to this value)
minLength	Specifies the minimum number of characters or list items allowed. Must be equal to or greater than zero
pattern	Defines the exact sequence of characters that are acceptable
totalDigits	Specifies the exact number of digits allowed. Must be greater than zero
whiteSpace	Specifies how white space (line feeds, tabs, spaces, and carriage returns) is handled

Fuente

XSD – Simple Element

- Ejemplo con restricciones 1:

```
<xs:element name="age">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="120"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

XSD – Simple Element

- Ejemplo 2:

```
<xs:element name="car">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="Audi"/>
      <xs:enumeration value="Golf"/>
      <xs:enumeration value="BMW"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

XSD – Simple Element

- Ejemplo 3:

```
<xs:element name="initials">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[A-Z][A-Z][A-Z]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

XSD – Simple Element

- Ejemplo 4:

```
<xs:element name="password">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:minLength value="5"/>
      <xs:maxLength value="8"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

XSD – Complex Element

- Contiene otros elementos y/o atributos
- Tipos:
  - Vacíos
  - Elementos que solo contienen otros elementos
  - Elementos que solo contienen texto
  - Elementos con otros elementos y texto

XSD – Complex Element

```
<xs:attribute name="xxx" type="yyy"/>
```

- `xxx`: Nombre del atributo
- `yyy`: Tipo de dato
- Los atributos son opcionales por defecto
- Otros atributos opcionales:
  - `default="valor"`: Valor por defecto si no se especifica uno
  - `fixed="valor"`: Valor fijo que no se puede cambiar
  - `use="required"`: Para indicar que el atributo es obligatorio

XSD – Complex Element

- Empty:

```
<xs:element name="product">
  <xs:complexType>
    <xs:attribute name="prodid" type="xs:positiveInteger"/>
  </xs:complexType>
</xs:element>
```

- Ejemplo de uso:

```
<product prodid="1345" />
```

XSD – Complex Element

- También se le puede dar un nombre y definir aparte:

```
<xs:element name="product" type="prodtype"/>

<xs:complexType name="prodtype">
  <xs:attribute name="prodid" type="xs:positiveInteger"/>
</xs:complexType>
```

XSD – Complex Element

- Elementos

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

XSD – Complex Element

`<xs:sequence>`: Indica el orden en el que tienen que aparecer los elementos

- Ejemplo de uso:

```
<person>
  <firstname>John</firstname>
  <lastname>Smith</lastname>
</person>
```

XSD – Complex Element

- Texto:

```
<xs:element name="shoesize">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:integer">
        <xs:attribute name="country" type="xs:string" />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

XSD – Complex Element

`<xs:extension base="xs:integer">:`

- Para limitar el tipo de datos del texto

- Ejemplo de uso:

```
<shoesize country="france">35</shoesize>
```

XSD – Complex Element

- Mixto:

```
<xs:element name="letter">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="orderid" type="xs:positiveInteger"/>
      <xs:element name="shipdate" type="xs:date"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

XSD – Complex Element

- `mixed="true"`: Para permitir que el texto aparezca entre los elementos
- Ejemplo de uso:

```
<letter>
  Dear Mr. <name>John Smith</name>.
  Your order <orderid>1032</orderid>
  will be shipped on <shipdate>2021-07-13</shipdate>.
</letter>
```

XSD – Complex Element

- En vez de `<xs:sequence>` se pueden usar otros indicadores:
  - `<xs:all>`:
    - Los hijos pueden aparecer en cualquier orden
    - Cada hijo tiene que aparecer una vez
  - `<xs:choice>`:
    - Sólo puede aparecer uno de los hijos una vez

XSD – Complex Element

- También se puede especificar la ocurrencia de cada elemento como atributo:

`maxOccurs="value"`:

  - Máximo número de ocurrencias
  - Por defecto 1
  - Se puede usar el valor `unbounded` para indicar que no hay límite

`minOccurs="value"`:

  - Mínimo número de ocurrencias
  - Por defecto 1

XSD – Complex Element

`<xs:any/>`

- Para añadir elementos no definidos
- Permite que el documento sea extensible

`<xs:anyAttribute/>`

- Para añadir atributos no especificados
- Permite que el documento sea extensible

XSD – Ejercicio 1

- Cree el XSD que valide el siguiente XML:

```
<items>
    <item>
        <name>Item 1</name>
        <photo>http://example.com/photo1.png</photo>
        <tags>Tag1, Tag2</tags>
        <diameter>32</diameter>
        <weight>540</weight>
        <price>60</price>
        <size>Big</size>
    </item>
    <item>
        <name>Item 2</name>
        <photo>http://example.com/photo2.jpg</photo>
        <tags>Tag1</tags>
        <diameter>23</diameter>
        <weight>340</weight>
        <price>50.1</price>
    </item>
</items>
```

Tenga en cuenta que los únicos valores válidos de la etiqueta size, si aparece, son:

- Big
- Small

XSD – Ejercicio 2

- Cree un XML que sea válido con el siguiente XSD:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Orderdetails">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" name="Customer">
          <xs:complexType>
            <xs:sequence>
              <xs:element minOccurs="0" name="fname">
                <xs:complexType>
                  <xs:simpleContent>
                    <xs:extension base="xs:string">
                      <xs:attribute name="from" type="xs:string" use="required" />
                    </xs:extension>
                  </xs:simpleContent>
                </xs:complexType>
              </xs:element>
```

XSD – Ejercicio 2

```
      <xs:element minOccurs="0" name="cname" type="xs:string" />
      <xs:element name="destination">
        <xs:complexType>
          <xs:simpleContent>
            <xs:extension base="xs:string">
              <xs:attribute name="Country" type="xs:string" use="required" />
              <xs:attribute name="Delivdate" type="xs:string" use="required" />
            </xs:extension>
          </xs:simpleContent>
        </xs:complexType>
      </xs:element>
      <xs:element minOccurs="0" name="email" type="xs:string" />
      <xs:element minOccurs="0" name="eid" type="xs:string" />
    </xs:sequence>
    <xs:attribute name="id" type="xs:unsignedByte" use="required" />
  </xs:complexType>
 </xs:element>
</xs:sequence>
</xs:complexType>
</xs:element></xs:schema>
```

XSD

- Validadores online

https://www.freeformatter.com/xml-validator-xsd.html

https://www.liquid-technologies.com/online-xsd-validator

https://www.corefiling.com/opensource/schemaValidate/

- Más información sobre XSD

https://www.w3schools.com/xml/schema_intro.asp

DTD / XSD – Ventajas e inconvenientes

Ventajas

- Las aplicaciones pueden asumir que existe una estructura concreta
- Se puede usar CSS/XML para dar forma a esa estructura
- Es más fácil escribir documentación y hay una estructura fija
- Resto de ventajas del tipado fuerte

Inconvenientes

- Un documento XML simplemente bien formado es más flexible y fácil de modificar
- Los DTDs/XSDs pueden llegar a ser muy complejos y difíciles de manipular (ej. RSS)
- Resto de desventajas del tipado débil

DTD / XSD – Comparativa

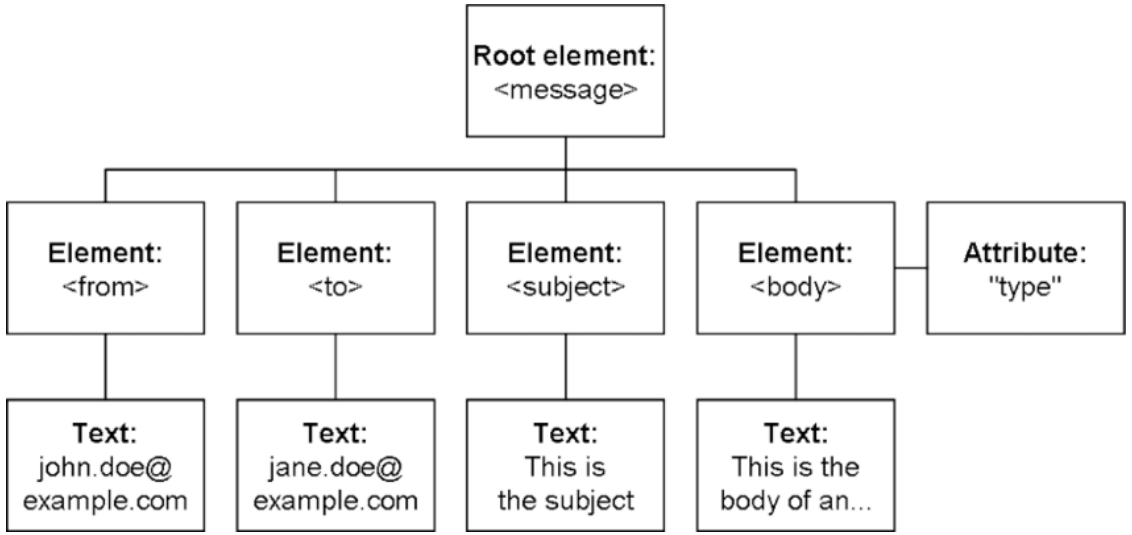
DTD	XSD
Document Type Definition	XML Schema Definition
Derived from SGML syntax	Written in XML
Doesn't support datatypes	Supports datatypes for elements and attributes
Doesn't support namespace	Supports namespace
Very limited control over number of elements	Can precisely define number of elements
Not extensible	Extensible
Not simple to learn	Simple to learn because you don't need to learn new language
Less control on XML structure	More control on XML structure
Doesn't allow duplicated definition of elements	Allows duplicated definition of elements

XPATH, XQUERY Y XSLT

XPath

¿Cómo navegamos en un documento XML?

- Hay que pensar en el documento XML como en un árbol
  - Document Objecto Model (DOM)
- Para navegar en dicho árbol utilizaremos XPath



XPath

- XML Path Language
- Sintaxis parecida a la de los directorios (paths)
- Usado en otros estándares como XSLT
- Recomendación W3C
  - Versión 3.1 en 2017 (última versión)
- Web para testear

XPath

Tipos de nodos:

- Elementos
- Atributos
- Texto
- Namespace
- Instrucción a procesar
- Comentario
- Documento

XPath

Construcciones básicas para rutas:

- `/`: Separador. Si empieza por /, referencia el nodo raíz
- `//`: Yo, y cualquier elemento descendiente
- `/Book`: Etiquetas
- `/@ISBN`: Atributos
  - Se obtiene su valor con `/data(@ISBN)`
- `|`: OR lógico
- `*`: Comodín
- `../`: Ir un nivel hacia atrás

XPath

```
<?xml version="1.0" encoding="UTF-8"?>
<class>
<name>L.1.1.1</name>
<student id="035">
  <name>Mark</name>
  <year>1999</year>
</student>
<student id="007">
  <name>Ana</name>
  <year>1998</year>
</student>
</class>
```

- ¿La información de toda la clase?
- ¿Todos los estudiantes?
- ¿Todos los nombres de los estudiantes?
- ¿Todas las ids de los estudiantes?

XPath

```
<?xml version="1.0" encoding="UTF-8"?>
<class>
<name>L.1.1.1</name>
<student id="035">
  <name>Mark</name>
  <year>1999</year>
</student>
<student id="007">
  <name>Ana</name>
  <year>1998</year>
</student>
</class>
```

`student`

- Todos los nodos con el nombre student

`/class`

- Elemento raíz

`//name`

- Todos los elementos name independientemente de donde estén

`/class/student/name`

- Todos los nombres de los alumnos

`//@id`

- Todos los atributos con nombre id

XPath

Predicados:

- `[ ]`: Separador del predicado
  - Se pueden anidar para agrupar condiciones
- `<, >, =, !=`: Comparadores
- `and`: AND lógico, enlaza condiciones
- `or`: OR lógico, enlaza condiciones
- `[número]`: Contador, se empieza a contar en 1
- `[@attr]`: Selecciona los atributos de nombre `attr`
- `[@attr='value']`: Atributos `attr` con valor `value`

XPath

```
<?xml version="1.0" encoding="UTF-8"?>
<class>
<name>L.1.1.1</name>
<student id="035">
  <name>Mark</name>
  <year>1999</year>
</student>
<student id="007">
  <name>Ana</name>
  <year>1998</year>
</student>
</class>
```

- ¿El segundo estudiante?
- ¿Estudiante con id 035?
- ¿Estudiantes del año 1999 o posteriores?
- ¿Todos los elementos student que tengan al menos un atributo?

XPath

```
<?xml version="1.0" encoding="UTF-8"?>
<class>
<name>L.1.1.1</name>
<student id="035">
  <name>Mark</name>
  <year>1999</year>
</student>
<student id="007">
  <name>Ana</name>
  <year>1998</year>
</student>
</class>
```

- ¿El segundo estudiante?

```
/class/student[2]
```

- ¿Estudiante con id 035?

```
/class/student[@id="035"]
```

- Estudiantes del año >= 1999

```
/class/student[year>=1999]
```

- student con algún atributo

```
/class/student[@*]
```

XPath

Funciones incluidas en XPath:

- Cuenta con multitud
- Se pueden usar en los predicados
- Ejemplos:
  - Contiene: `contains(elemento, “texto”)`
  - Nombre: `name()`
  - Contador: `count()`
  - Contenido del nodo: `text()`
  - Último elemento: `last()`

XPath

```
<?xml version="1.0" encoding="UTF-8"?>
<class>
<name>L.1.1.1</name>
<student id="035">
  <name>Mark</name>
  <year>1999</year>
</student>
<student id="007">
  <name>Ana</name>
  <year>1998</year>
</student>
</class>
```

- ¿Cuántos estudiantes hay?
- ¿Nombre de los estudiantes que han nacido en 1999?
- ¿Año de nacimiento de Ana?
- ¿Último estudiante?

XPath

```
<?xml version="1.0" encoding="UTF-8"?>
<class>
<name>L.1.1.1</name>
<student id="035">
  <name>Mark</name>
  <year>1999</year>
</student>
<student id="007">
  <name>Ana</name>
  <year>1998</year>
</student>
</class>
```

- ¿Cuántos estudiantes hay?

```
count(//student)
```

- ¿Nombre de los estudiantes que han nacido en 1999?

```
//student[year='1999']/name/text()
```

- ¿Año de nacimiento de Ana?

```
//student/name[contains(text(),'Ana')]/../year/text()
//student/name[contains(.,'Ana')]/../year/text()
//student[name='Ana']/year/text()
```

- ¿Último estudiante?

```
//student[last()]
```

XPath

Ejes de navegación en Xpath:

- Incluye 13 distintos
- Ejemplos:
  - Padre: `parent::`
  - Hermano que le precede: `preceding-sibling::`
  - Hermano que le siga: `following-sibling::`
  - Descendientes: `descendant:`
  - Propia etiqueta: `self:`
  - …así hasta 13 diferentes…

XPath – Ejemplos

Partiendo del archivo xml_ejemplo1.xml

- Todos los títulos de libro:

```
/bookstore/book/title
```

- Todos los títulos, aparezcan dónde aparezcan:

```
//title
```

XPath – Ejemplos

Partiendo del archivo xml_ejemplo1.xml

- Título de los libros con un precio mayor que 35:

```
/bookstore/book[price>35]/title
```

- Todos los títulos con un atributo “lang” con valor ‘en’:

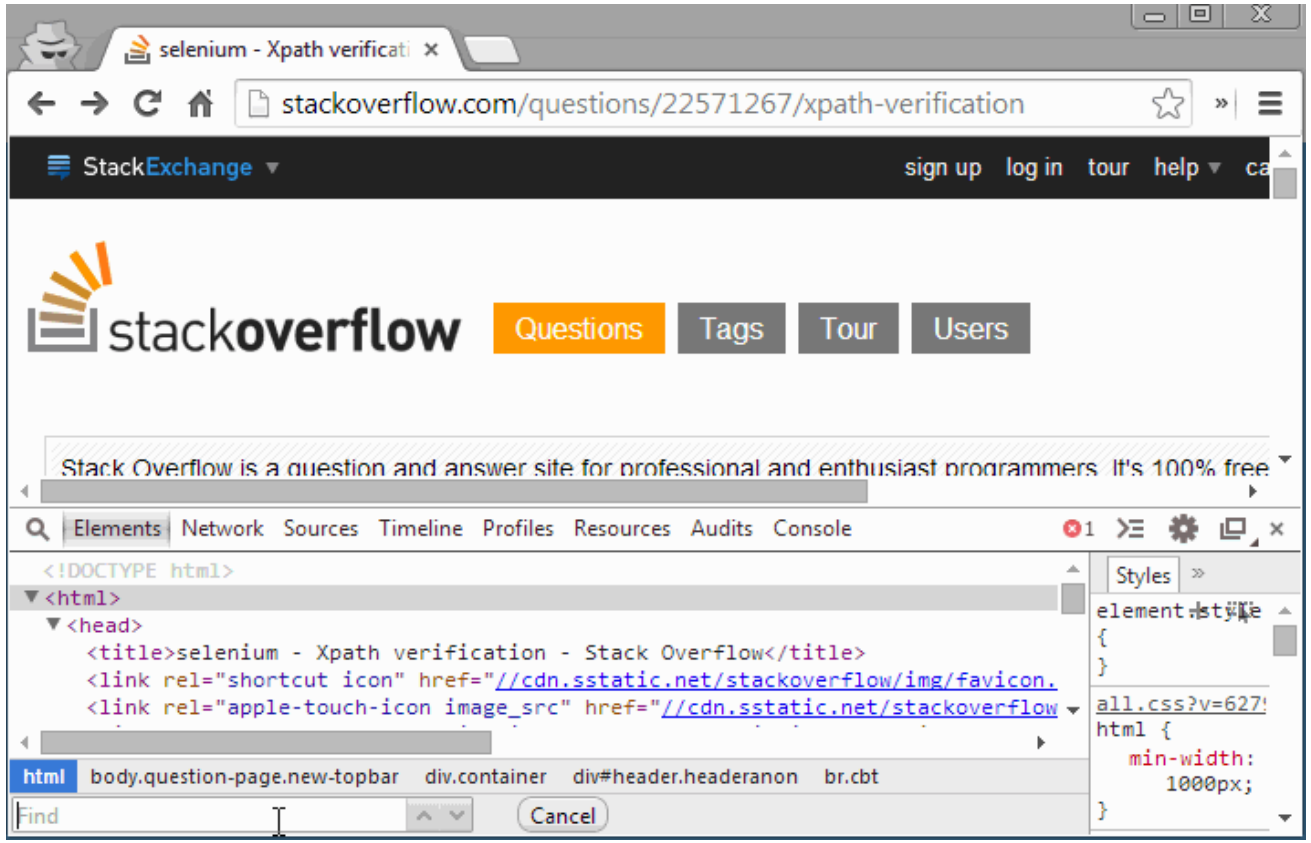
```
//title[@lang='en']
```

- Todos los elementos título con al menos un atributo:

```
//title[@*]
```

XPath

- En la pestaña "Elements" pulsar: Ctrl + F



XPath – Ejemplo

- Dado el siguiente XML:

```
<?xml version="1.0" encoding="utf-8"?>
<Wikimedia>
  <projects>
    <project name="Wikipedia" launch="2001-01-05">
      <editions>
        <edition language="English">en.wikipedia.org</edition>
        <edition language="German">de.wikipedia.org</edition>
        <edition language="French">fr.wikipedia.org</edition>
        <edition language="Polish">pl.wikipedia.org</edition>
        <edition language="Spanish">es.wikipedia.org</edition>
      </editions>
    </project>
    <project name="Wiktionary" launch="2002-12-12">
      <editions>
        <edition language="English">en.wiktionary.org</edition>
        <edition language="French">fr.wiktionary.org</edition>
        <edition language="Vietnamese">vi.wiktionary.org</edition>
        <edition language="Turkish">tr.wiktionary.org</edition>
        <edition language="Spanish">es.wiktionary.org</edition>
      </editions>
    </project>
  </projects>
</Wikimedia>
```

XPath – Ejemplo

Haz las consultas XPath que cumplan lo siguiente y el resultado sea el que se muestre después:

- Los nombres de todos los proyectos

```
Wikipedia
Wiktionary
```

- Solo las URL de todos los proyectos en español

```
es.wikipedia.org
es.wiktionary.org
```

XPath – Ejemplo

- Todas las ediciones de todos los proyectos

```
<edition language="English">en.wikipedia.org</edition>
<edition language="German">de.wikipedia.org</edition>
<edition language="French">fr.wikipedia.org</edition>
<edition language="Polish">pl.wikipedia.org</edition>
<edition language="Spanish">es.wikipedia.org</edition>
<edition language="English">en.wiktionary.org</edition>
<edition language="French">fr.wiktionary.org</edition>
<edition language="Vietnamese">vi.wiktionary.org</edition>
<edition language="Turkish">tr.wiktionary.org</edition>
<edition language="Spanish">es.wiktionary.org</edition>
```

XPath – Ejemplo

- Sólo las URL de todas las Wikipedias

```
en.wikipedia.org
de.wikipedia.org
fr.wikipedia.org
pl.wikipedia.org
es.wikipedia.org
```

- La cuarta edición del Wiktionary

```
<edition language="Turkish">tr.wiktionary.org</edition>
```

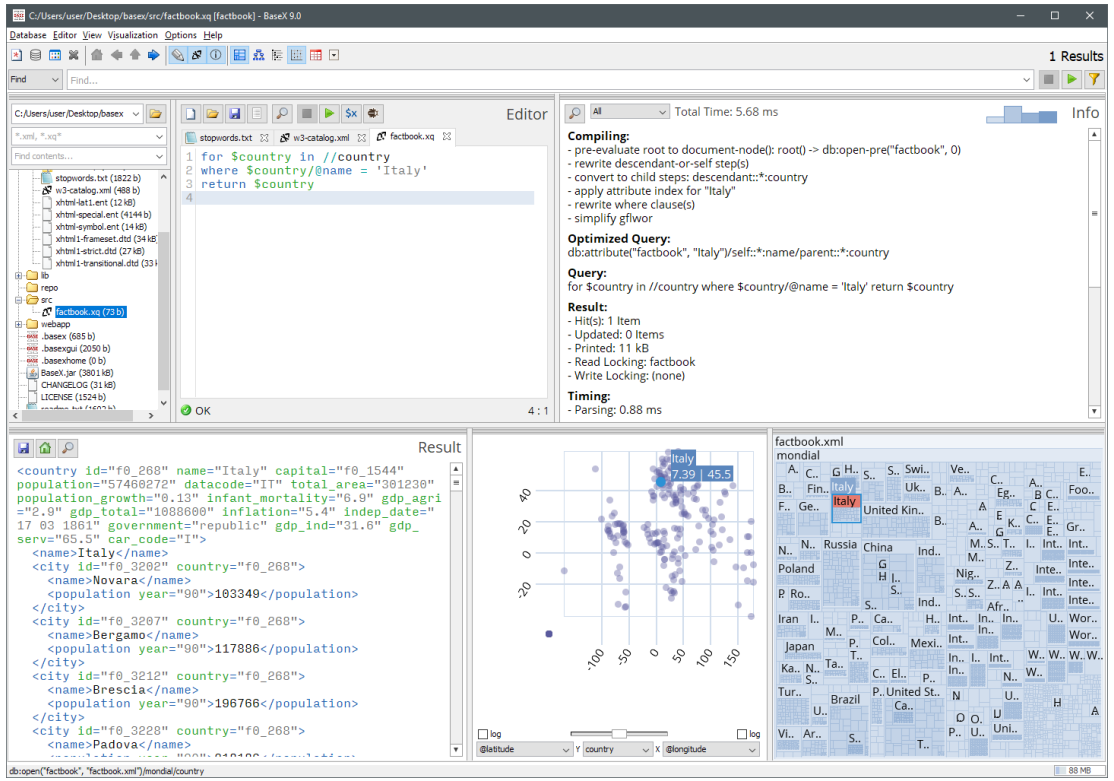
XPath

- Para trabajar con ficheros grandes vamos a usar BaseX:

http://basex.org/

- Requiere tener Java instalado
- Para cargar el XML:

```
doc("hamlet.xml")
```



XPath – Ejercicio 1

Usa Xpath para responder a las siguientes preguntas sobre el archivo `hamlet.xml`:

1. Los títulos de todas las escenas del segundo acto
2. ¿Cuántas veces habla Hamlet?
3. El texto de las líneas que contengan la palabra "king"
4. El segundo diálogo (speech) de Bernardo en el tercer acto
5. Número de líneas de cada diálogo (speech)
6. Número de líneas del diálogo (speech) con más líneas
7. Todos los diálogos (speech) de la tercera PERSONA del primer PGROUP dentro de PERSONAE

XPath – Ejercicio 2

Web scraping

- Extraer datos de páginas web
- Normalmente de forma automática haciendo uso de bots
- Una vez extraída la información se procesa
- Normalmente se repite la operación a lo largo del tiempo para comparar la evolución de los datos

XPath – Ejercicio 2

Web scraping

- Ejemplos:
  - Google extrayendo información de las webs para el buscador
  - Internet Archive para conservar webs antiguas
  - Páginas de comparación de precios

XPath – Ejercicio 2

- Accede a la web de cualquier tienda online y extrae los precios y nombres de los productos haciendo uso de las herramientas del navegador y usando XPath

XPath

- Ejemplos:

https://www.w3schools.com/xml/xpath_examples.asp

- XPath tester:

http://xpather.com/: Se puede usar el Ctrl para ver las selecciones

https://extendsclass.com/xpath-tester.html

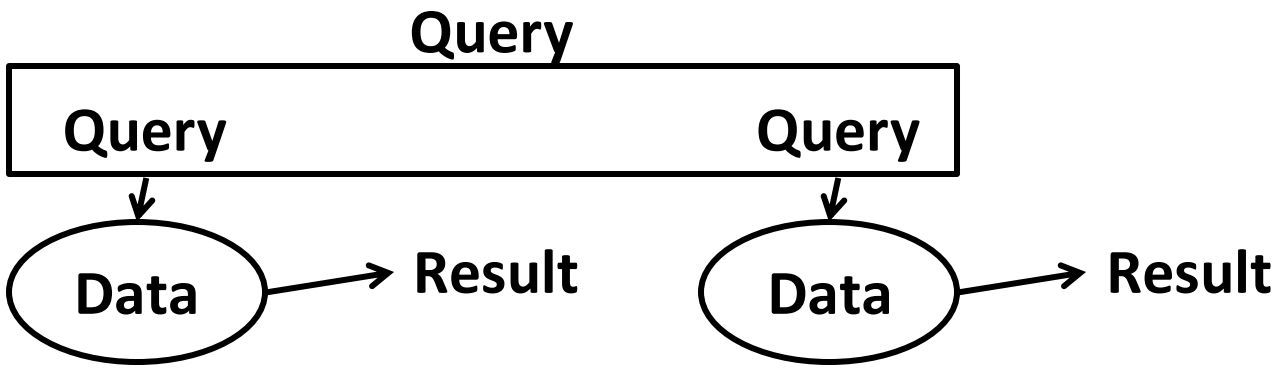
- XPath cheatsheet:

https://devhints.io/xpath

XQuery

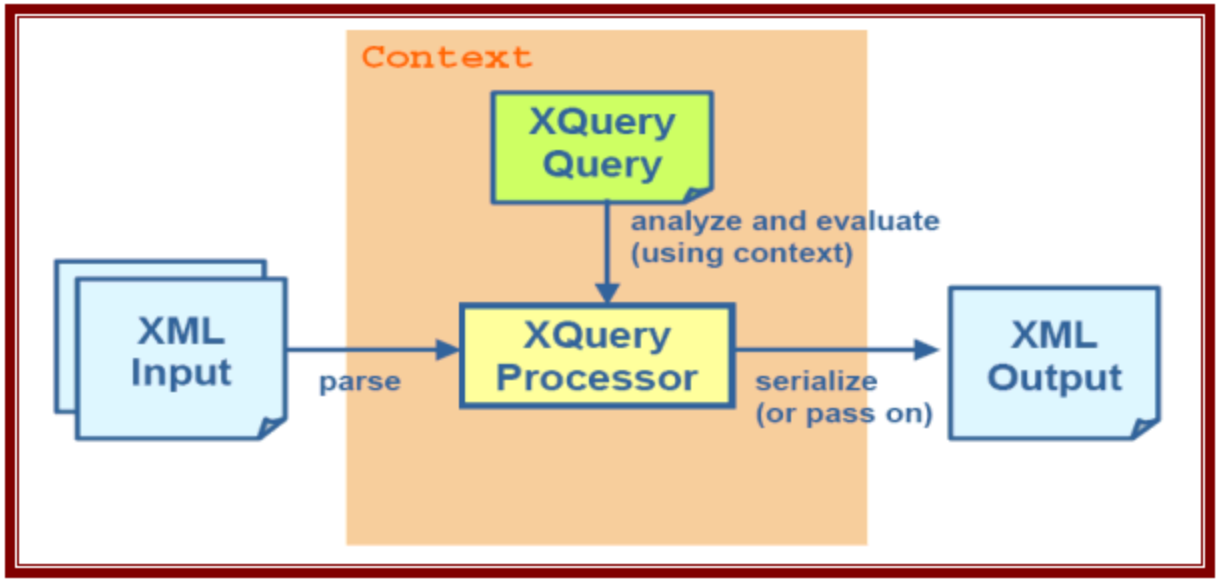
- Lenguaje para realizar consultas sobre XML
- Similar en concepto a SQL
- Versión 3.1 en 2017
- Cada expresión opera sobre y devuelve una secuencia de elementos:
  - Documento XML o Stream XML
- XPath es uno de los tipos de expresiones que soporta
  - Podemos incluir XPath dentro de XQuery

XQuery



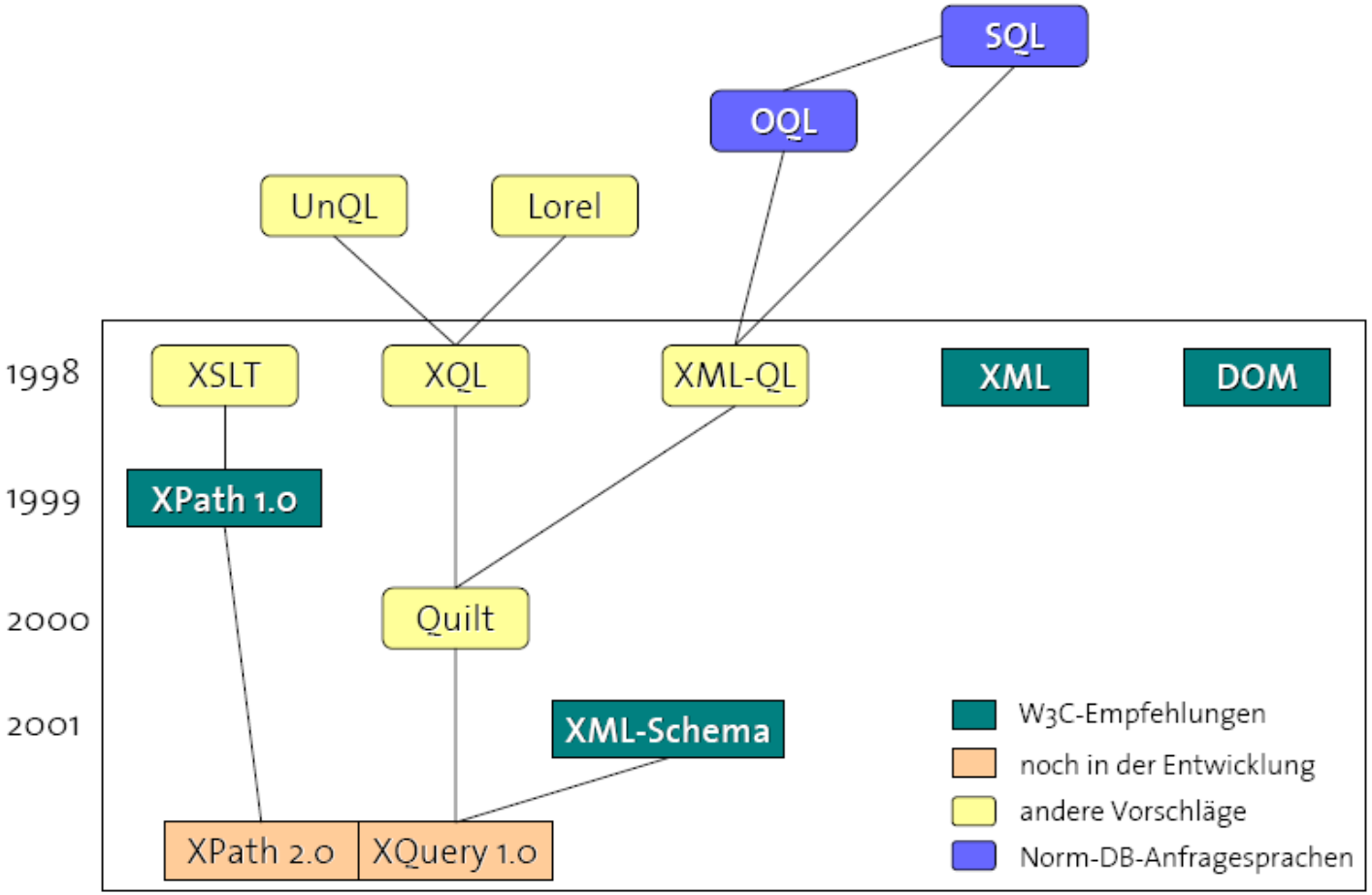
XQuery

https://www.w3schools.com/xml/xquery_example.asp



(Figure 1: XQuery Processing Model)

XQuery



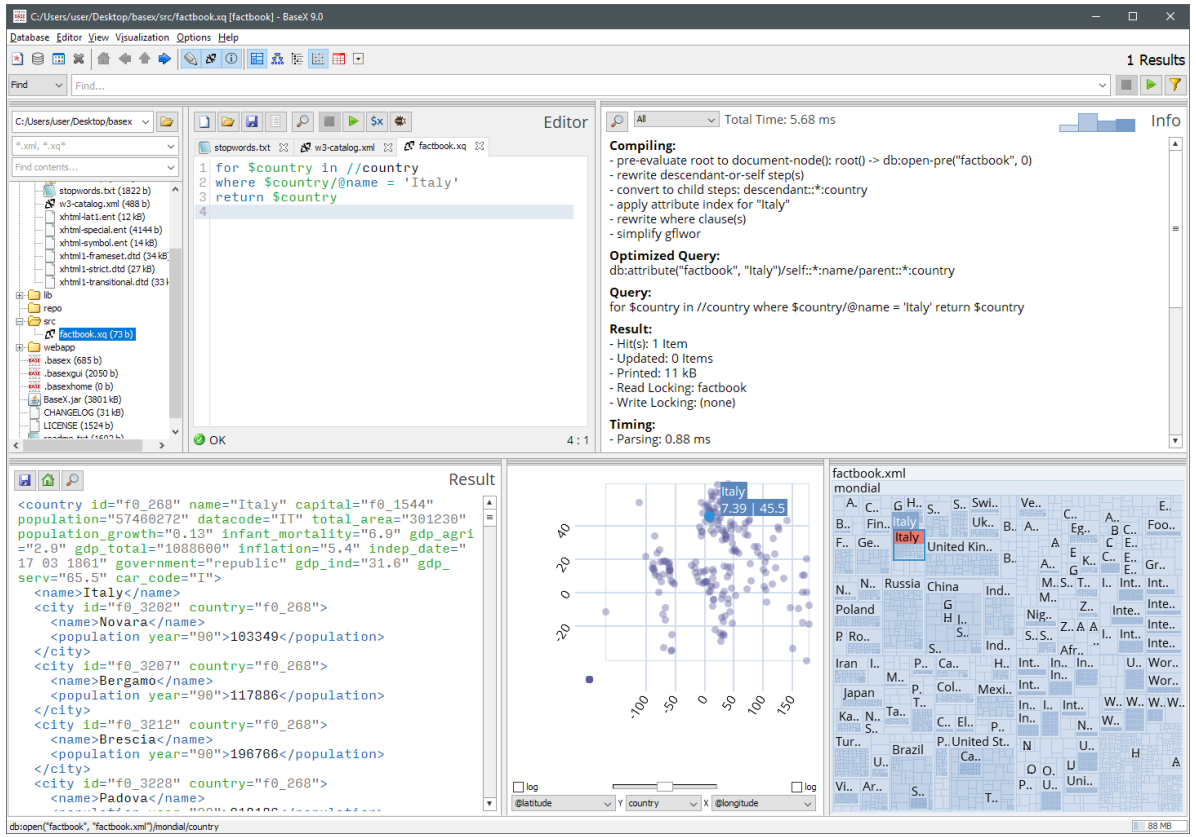
[illustration © C. Türker]

XQuery

- Vamos a trabajar con la siguiente aplicación:

http://basex.org/

- Requiere tener Java instalado



XQuery

- **Archivo `xquery_ejemplo1.xml`**

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
<book category="COOKING">
  <title lang="en">Everyday Italian</title>
  <author>Giada De Laurentiis</author>
  <year>2005</year>
  <price>30.00</price>
</book>
<book category="CHILDREN">
  <title lang="en">Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>
<book category="WEB">
  <title lang="en">XQuery Kick Start</title>
  <author>James McGovern</author>
  <author>Per Bothner</author>
  <author>Kurt Cagle</author>
  <author>James Linn</author>
  <author>Vaidyanathan Nagarajan</author>
  <year>2003</year>
  <price>49.99</price>
</book>
<book category="WEB">
  <title lang="en">Learning XML</title>
  <author>Erik T. Ray</author>
  <year>2003</year>
  <price>39.95</price>
</book>
</bookstore>
```

XQuery

- Función para cargar el XML:

```
doc("xquery_ejemplo1.xml")
```

- Navegamos con XPath:

```
doc("xquery_ejemplo1.xml")/bookstore/book/title
```

- Podemos usar predicados:

```
doc("xquery_ejemplo1.xml")/bookstore/book[price<30]
```

XQuery – FLOWR

- `for:` selecciona una secuencia de nodos
- `let:` enlaza la secuencia a una variable
- `where:` filtra los nodos
- `order by:` ordena los nodos
- `return:` lo que devolvemos (evaluado una vez por cada nodo)

XQuery

- Los siguientes bloques son equivalentes:

```
doc("xquery_ejemplo1.xml")/bookstore/book[price>30]/title
```

```
for $x in doc("xquery_ejemplo1.xml")/bookstore/book
where $x/price>30
return $x/title
```

XQuery

- Con XQuery tenemos más opciones
- Podemos ordenar el resultado (archivo xquery_ejemplo1a.xq):

```
for $x in doc("xquery_ejemplo1.xml")/bookstore/book
where $x/price>30
order by $x/title
return $x/title
```

- Podemos incluir texto / XML / HTML (archivo xquery_ejemplo1b.xq):

```
<ul>
{
  for $x in doc("xquery_ejemplo1.xml")/bookstore/book/title
  order by $x
  return <li>{data($x)}</li>
}
</ul>
```

XQuery

- Podemos aplicar condiciones (archivo xquery_ejemplo1c.xq):

```
for $x in doc("xquery_ejemplo1.xml")/bookstore/book
return if ($x/@category="CHILDREN")
then <child>{data($x/title)}</child>
else <adult>{data($x/title)}</adult>
```

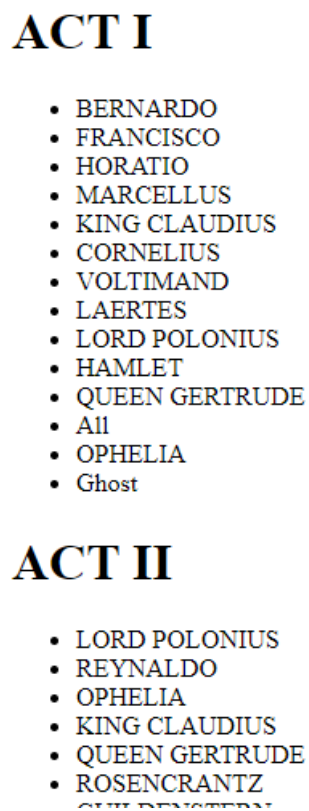
- Podemos contar con la palabra clave at (archivo xquery_ejemplo1d.xq):

```
for $x at $i in
doc("xquery_ejemplo1.xml")/bookstore/book/title
return <book>{$i}. {data($x)}</book>
```

XQuery – Ejemplo

- Mostrar los personajes distintos que aparecen en cada acto de Hamlet, devolviendo los datos en formato HTML (archivo xquery_ejemplo2.xq):

```
<html><body>
{
  for $act in doc("hamlet.xml")//ACT
  let $speakers := distinct-values($act//SPEAKER)
  return
    <div>
      <h1>{ string($act/TITLE) }</h1>
      <ul>
        {
          for $speaker in $speakers
          return <li>{ $speaker }</li>
        }
      </ul>
    </div>
}
</body></html>
```



XQuery – Ejercicio 1

- Dado el siguiente XML:

```
<?xml version="1.0"?>
<bib>
  <book year="1994">
    <title>TCP/IP Illustrated</title>
    <author>Stevens</author>
    <publisher>Addison-Wesley</publisher>
    <price>65.95</price>
  </book>
  <book year="1994">
    <title>Principles of Databases</title>
    <author>Abiteboul</author>
    <publisher>Addison-Wesley</publisher>
    <price>35.89</price>
  </book>
  <book year="1992">
    <title>Advanced Programming in the Unix environment</title>
    <author>Stevens</author>
    <publisher>Addison-Wesley</publisher>
    <price>65.95</price>
  </book>
  <book year="2000">
    <title>Data on the Web</title>
    <author>Abiteboul</author>
    <author>Buneman</author>
    <author>Suciu</author>
    <publisher>Morgan Kaufmann Publishers</publisher>
    <price> 39.95</price>
  </book>
  <book year="1992">
    <title>The Economics of Technology and Content for Digital
TV</title>
    <editor>
    Gerbarg
    <affiliation>CITI</affiliation>
    </editor>
    <publisher>Kluwer Academic Publishers</publisher>
    <price>129.95</price>
  </book>
</bib>
```

XQuery – Ejercicio 1

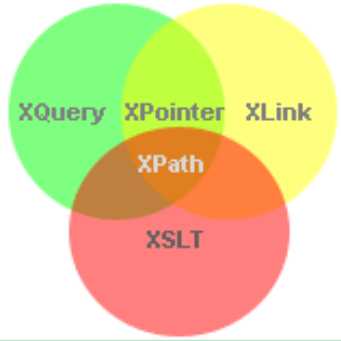
1. Devuelve los títulos de todos los libros ordenados por precio de mayor a menor
2. ¿Cuántos libros ha escrito Abiteboul?
3. Haz un XML que indique para cada autor, cuántos libros ha escrito

XQuery – Ejercicio 1

- Ejemplo de la solución 3:

```
<authors>
  <author>
    <name>Stevens</name>
    <count>2</count>
  </author>
  <author>
    <name>Abiteboul</name>
    <count>2</count>
  </author>
  <author>
    <name>Buneman</name>
    <count>1</count>
  </author>
  <author>
    <name>Suciu</name>
    <count>1</count>
  </author>
</authors>
```

XQuery vs XPath



XQuery	XPath
Functional programming and query language that is used to query a group of XML data	XML path language that is used to select nodes from an xml document using queries
Used to extract and manipulate data from either xml documents or relational databases and ms office documents that support an xml data source	Used to compute values like strings, numbers and boolean types from another xml documents
Represented in the form of a tree model with seven nodes, namely processing instructions, elements, document nodes, attributes, namespaces, text nodes, and comments	Represented as tree structure, navigate it by selecting different nodes
Supports Xpath and extended relational models	A component of query language
Helps to create syntax for new xml documents	Created to define a common syntax and behavior model for Xpointer and XSLT

XSLT

- XSL: Extensible Stylesheet Language: Versión inicial
- XSLT: XSL (with) Transformations: Versión mejorada
- XSLT se escribe utilizando XML
- Versión 3.0 en 2017 (v 4.0 en propuesta)
- Estructura el documento en nodos: elementos, atributos, texto, comentarios…
- Sirve para encontrar partes de un documento XML (usando XPath) y reemplazarlas por otras
- Busca conforme a una plantilla y reemplaza el resultado entero

XSLT

- Pueden aplicarse recursivamente
- Usa construcciones típicas de los lenguajes de programación:
  - Condicionales: (if-else)
  - Iteraciones: (for-each)
- Al usarlo se debe tener cuidado con:
  - Comportamientos extraños con los espacios en blanco
  - Prioridad implícita de las plantillas

XSLT – Demo

- https://www.w3schools.com/xml/tryxslt.asp?xmlfile=cdcatalog&xsltfile=cdcatalog

XML Code:

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog>
  <cd>
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <country>USA</country>
    <company>Columbia</company>
    <price>10.90</price>
    <year>1985</year>
  </cd>
  <cd>
    <title>Hide your heart</title>
    <artist>Bonnie Tyler</artist>
    <country>UK</country>
```

XSLT Code:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
<body>
  <h2>My CD Collection</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th style="text-align:left">Title</th>
      <th style="text-align:left">Artist</th>
    </tr>
    <xsl:for-each select="catalog/cd">
    <tr>
```

Edit the XML or XSLT code above and Click Me »

My CD Collection

Title	Artist
Empire Burlesque	Bob Dylan
Hide your heart	Bonnie Tyler
Greatest Hits	Dolly Parton
Still got the blues	Gary Moore
Eros	Eros Ramazzotti
One night only	Bee Gees

XSLT – Declaración

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

- Hay que incluirlo al principio del documento (después del XML prolog, si existe)
- En la URL está definido el estandar
- Con `xmlns:xsl` definimos xsl como el namespace que vamos a utilizar para tener acceso a las etiquetas definidas
- También podemos usar (totalmente equivalentes):

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

XSLT – Elementos

```
<xsl:template match="pattern">...</ xsl:template>
```

- Para construir plantillas
- El contenido es lo que se escribe a la salida
- También se puede usar para descartar
- El atributo match la asocia a un nodo o nodos del XML
- El valor de match es una expresión XPath
- `match="/"` cubre todo el documento
- `match="text()"` cubre el texto, ni las etiquetas ni los atributos
- `match="*|@*|text()"` cubre todo el documento, pero cada ente por separado
- Cuidado con el orden de las plantillas
- Más información

XSLT – Elementos

```
<xsl:value-of select="expression" />
```

- Extrae el valor de un nodo
- El atributo select especifica lo que se extrae
- El valor de select es una expresión XPath
- Más información

```
<xsl:for-each select="expression">...</xsl:for-each>
```

- Extrae todos los valores de un nivel
- El atributo select especifica lo que se extrae
- El valor de select es una expresión XPath
- Más información

XSLT – Elementos

```
<xsl:sort select="expression" />
```

- Ordena los valores
- En orden ascendente por defecto
  - Podemos añadir el atributo `order="descending"`
- Se usa dentro de un for-each
- El atributo select especifica lo que se ordena
- El valor de select es una expresión XPath
- Más información

XSLT – Elementos

```
<xsl:if test="expression"> ... </xsl:if>
```

- Impone una condición
- No permite else
  - Usar choose, when, otherwise (más información)
- Se usa dentro de un for-each
- El atributo test especifica lo que se comprueba
- Más información
- Ejemplo:

```
<xsl:if test="price &gt; 10"> ... </xsl:if>
```

XSLT – Elementos

```
<xsl:attribute name="value">
  <xsl:value-of select="…" />
</xsl:attribute>
```

- Para añadir un atributo a un elemento

XSLT – Elementos

```
<xsl:variable name="nombreVariable" select="…"/>
```

- Para definir una variable
- Luego se utiliza con $nombreVariable

XSLT – Ejemplo

Archivo xslt_ejemplo1.xslt

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html><body>
  <h2>My CD Collection</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th style="text-align:left">Title</th>
      <th style="text-align:left">Artist</th>
    </tr>
    <xsl:for-each select="catalog/cd">
    <tr>
      <td><xsl:value-of select="title"/></td>
      <td><xsl:value-of select="artist"/></td>
    </tr>
    </xsl:for-each>
  </table>
</body></html>
</xsl:template>
</xsl:stylesheet>
```

Archivo xslt_ejemplo.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog>
  <cd>
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <country>USA</country>
    <company>Columbia</company>
    <price>10.90</price>
    <year>1985</year>
  </cd>
  <!-- ... -->
</catalog>
```

XSLT – Ejemplo

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html>
  <body>
    <h2>My CD Collection</h2>
    <table border="1">
      <tr bgcolor="#9acd32">
      <th>Title</th>
      <th>Artist</th>
      <th>Price</th>
    </tr>
    <xsl:for-each select="catalog/cd">
    <xsl:sort select="artist"/>
    <xsl:if test="price>10">
      <tr>
        <td><xsl:value-of select="title"/></td>
        <td><xsl:value-of select="artist"/></td>
        <td><xsl:value-of select="price"/></td>
      </tr>
    </xsl:if>
    </xsl:for-each>
    </table>
  </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

XSLT

```
<xsl:apply-templates select="expression"/>
```

- Permite encadenar templates
- En el select podemos especificar los elementos sobre los que se aplica
- Sirve para reutilizar código
- Más información

XSLT – Ejemplo

Archivo xslt_ejemplo2.xslt

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html>
  <body>
  <h2>My CD Collection</h2>
  <xsl:apply-templates/>
  </body>
  </html>
</xsl:template>

<xsl:template match="cd">
  <p>
  <xsl:apply-templates select="title"/>
  <xsl:apply-templates select="artist"/>
  </p>
</xsl:template>

<xsl:template match="title">
  Title: <span style="color:#ff0000">
  <xsl:value-of select="."/></span>
  <br />
</xsl:template>

<xsl:template match="artist">
  Artist: <span style="color:#00ff00">
  <xsl:value-of select="."/></span>
  <br />
</xsl:template>

</xsl:stylesheet>
```

XSLT – Ejercicio

- Descárguese la información de población de esta web en formato XML
- Trabaje con un subset representativo de los datos
  - Las herramientas online no soportan archivos XML tan grandes
- Utilice XSLT para generar una página HTML con las siguientes características:
  - Un selector que permita elegir el país
  - Para el país seleccionado se muestra una tabla con la información de población de cada año

XSLT – Ejercicio

- Ejemplo de resultado:

Choose a country: Aruba

Aruba
Africa Eastern and Southern
Afghanistan
Africa Western and Central
Angola
Albania
Andorra
Arab World
United Arab Emirates
Argentina
Armenia
American Samoa
Antigua and Barbuda
Australia
Austria
Azerbaijan
Burundi
Belgium
Benin
Burkina Faso

Aruba

Year	Population
1960	54208
1961	55434
1962	56234
1963	56699
1964	57029
1965	57357
1966	57702
1967	58044
1968	58377
1969	58734
1970	59070
1971	59442
1972	59849
1973	60236

XSLT

- Tutorial

https://www.w3schools.com/xml/xsl_intro.asp

- Demo online

https://www.w3schools.com/xml/tryxslt.asp?xmlfile=cdcatalog&xsltfile=cdcatalog

- Referencia

https://www.w3schools.com/xml/xsl_elementref.asp

- Editor online

https://www.freeformatter.com/xsl-transformer.html

XQuery vs XSLT

XQuery	XSLT
Program driven	Function Driven
Declarative	Functional
Not XML based	XML based
Simple transformations	Complex transformations
Designed for retrieving and interpreting information	Mainly designed for transforming
Easier	Difficult
Shorter and faster	Hard to mantain

NODE.JS

Node.js

- Node.js no está diseñado para trabajar con XML
  - Java tiene librerías pensadas para eso
- Pero puede ser necesario tener que usar XML
- Afortunadamente hay muchas librerías para trabajar con XML
  - En general parsean XML ↔ JSON
  - En general no validan
  - En general no permiten trabajar con XPath, XQuery ni con XSLT

Node.js

node-xml2js

- 20M+ de descargas semanales
- Última actualización de hace 7 meses
- 2 dependencias

Node.js

xmlbuilder-js

- 25M+ de descargas semanales
- Última actualización de hace 4 años
  - Nueva versión xmlbuilder2
- 0 dependencias

Node.js

sax

- 34M+ de descargas semanales
- Última actualización de hace 5 meses
- 0 dependencias

Node.js

fast-xml-parser

- 18M+ de descargas semanales
- Última actualización reciente
- 1 dependencia
- Demo

RSS

- RDF Site Summary
- Really Simple Syndication
- RSS 2.0 en 2009
- Feed web:
  - Información que cambie a menudo
- Formato XML
- Alternativa: Atom (también en XML)
- XSD

RSS

- News / Feed aggregators
  - Aplicaciones para procesar los RSS
  - Permiten seleccionar las feeds
  - Presentan al usuario la información de las feeds
  - Ej:
    - Feedly
    - Inoreader
    - Google Reader (RIP)

RSS – Ejemplo

```
<rss version="2.0">
<channel>
  <title>Liftoff News</title>
  <link>http://liftoff.msfc.nasa.gov/</link>
  <description>Liftoff to Space Exploration.</description>
  <language>en-us</language>
  <pubDate>Tue, 10 Jun 2003 04:00:00 GMT</pubDate>
  <lastBuildDate>Tue, 10 Jun 2003 09:41:01 GMT</lastBuildDate>
  <item>
    <title>Star City</title>
    <link>http://liftoff.msfc.nasa.gov/news/2003/news-starcity.asp</link>
    <description>How do Americans get ready to work with Russians(…)</description>
    <pubDate>Tue, 03 Jun 2003 09:39:21 GMT</pubDate>
    <guid>http://liftoff.msfc.nasa.gov/2003/06/03.html#item573</guid>
  </item>
  <item>
    <description>Sky watchers in Europe, Asia, and parts of Alaska and Canada (…)</description>
    <pubDate>Fri, 30 May 2003 11:06:42 GMT</pubDate>
    <guid>http://liftoff.msfc.nasa.gov/2003/05/30.html#item572</guid>
  </item>
</channel>
</rss>
```

Node.js – Demo

- Vamos a hacer una web que procese noticias RSS con node.js

Node.js – Demo

- Lector de RSS con node.js

```
express -v ejs rssReader
cd rssReader
npm install
npm install axios
npm install fast-xml-parser
```

Node.js – Demo

- Modificamos app.js para configurar la ruta /rss:

```
const axios = require('axios');
(…)
var rssRouter = require('./routes/rss');
(…)
app.use('/rss', fetchRSS, rssRouter);
```

Node.js – Demo

- Configuramos el middleware en app.js para obtener el RSS:

```
async function fetchRSS(req, res, next){
  if(req.query.url){
    const url = req.query.url;
    //TODO parsear URL
    console.log(url);
    const resData = await axios.get(url);
    req.data = resData.data;
  }
  next();
}
```

Node.js – Demo

- Configuramos `routes/rss.js`:

```
var express = require('express');
var router = express.Router();
const {XMLParser, XMLBuilder, XMLValidator} = require('fast-xml-parser');
const options = { ignoreAttributes : true};
const parser = new XMLParser(options);
router.get('/', function(req, res, next) {
  let jsonObj = '';
  if(req.data){
    jsonObj = parser.parse(req.data);
    console.log(jsonObj.rss.channel.item);
  }
  res.render('rss', { title:'RSS Reader', data:jsonObj});
});
module.exports = router;
```

Node.js – Demo

- Configuramos el template `views/rss.ejs`:

```
<html><body>
    <h1><%= title %></h1>
    <p>Welcome to <%= title %></p>
    <form method="get" action="/rss">
      <label for="url">RSS URL:</label>
      <input type="text" name="url">
      <input type="submit" name=""/>
    </form>
    <% if(data != ''){ %>
    <div >
      <% data.rss.channel.item.forEach(function(item){ %>
        <div class="container-fluid clear">
          <h2><%= item.title %></h2>
          <div><%- item.description %></div>
          <div><a href="<%= item.link %>">Seguir leyendo...</a></div>
        </div>
      <% }); %>
    </div>
    <% } %>
</body></html>
```

Node.js – Demo

- Ejecutamos la demo:

Node.js – Ejercicio

- Haz una app en node.js que procese un XML de algún lado (que no sea RSS) y una página web que lo muestre
- Listado de APIs
- Otra lista

Abreviaturas

CSS: Cascading Style Sheets

DOM: Document Object Model

DTD: Document Type Descriptor

HTML: Hypertext Markup Language

ISO: International Organization for Standardization

SAX: Simple API for XML

SGML: Standard Generalized Markup Language

XML: Extensible Markup Language

XSD: XML Schema Definition

XSL: Extensible Stylesheet Language

XSLT: Extensible Stylesheet Language Transformations

Enlaces

- Norma ISO SGML

https://www.iso.org/obp/ui/#iso:std:iso:8879:ed-1:v1:en

- XML W3C Recommendation

https://www.w3.org/TR/xml/

- Tutorial XML en W3C

https://www.w3schools.com/xml/

- Diferencias entre XML y SGML

https://www.w3.org/TR/NOTE-sgml-xml-971215/

- DTD

https://www.w3schools.com/xml/xml_dtd_intro.asp

Enlaces

- Validador XML con DTD

https://www.xmlvalidation.com/

- XSD

https://www.w3schools.com/xml/schema_intro.asp

- Validador XML con XSD

https://www.freeformatter.com/xml-validator-xsd.html

- XPath tester

https://www.freeformatter.com/xpath-tester.html

Bibliografía

- Learning XML (O'Reilly) [978-0596004200]
- XML in a Nutshell (O'Reilly) [978-0596007645]
- XML Pocket Reference (O'Reilly) [978-0596100506]
- Definitive XML Schema (Prentice Hall) [978-0132886727]
- XML (For Dummies) [978-0764588457]
- Beginning XML (Wrox) [978-1118162132]