XML

Loreto Bravo

Contexto

- **SGML (Standard Generalized Markup Language)**
  - Es el estándar internacional para la definición de la estructura y el contenido de diferentes tipos de documentos.
- **HTML (HyperText Markup Language)**
  - Estándar para publicación de información con formato, entregable por HTTP.
  - Diseñado principalmente para *visualización* de datos y se centra en cómo aparece la información, no en su estructura.
  - Es sólo un tipo de documento **SGML**

Se requiere algo nuevo...

- Barato, veloz y sencillo:
  - Para crear documentos.
  - Para procesar documentos.
  - Para presentar documentos.
- Extensible:
  - Un conjunto de reglas, no un conjunto de etiquetas.
- Compatible con el HTML:
  - Debe tener una manera sencilla de convertir de HTML.
- Compatible con el SGML:
  - Debe de conservar su potencia sin contener complejidades no necesarias.

Metas de diseño.

- XML debe ser utilizable a través de Internet.
- XML debe soportar muchos **escenarios** de aplicación.
- XML debe ser compatible con el SGML.
- Los programas que procesen documentos XML deben ser fáciles de crear.
- Los documentos en XML deben de ser legibles por humanos y razonablemente claros.

El XML es...

- El Lenguaje de Marcaje Extensible (*Extensible Markup Languaje*, XML).
  - Un metalenguaje de marcaje
    - Sirve para definir otros lenguajes de propósito específico (e.g., XHTML, OpenOffice, XSL, RDF, etc.)
      - http://en.wikipedia.org/wiki/List_of_XML_markup_languages
- Una recomendación técnica del W3C.
  - Es un estándar del W3C, no de alguna compañía.
- Multiplataforma, simple, fácil de aprender.
  - Es fácil construir herramientas para XML.
  - Optimizado para usarse en Internet.
- Libre (y gratuito).

El XML **no** es...

- Un lenguaje de marcaje (*markup*).
  - No. Es un estándar que especifica una sintaxis para crear lenguajes de marcaje.
- Solo para Web.
  - No. Puede ser usado para describir y comunicar cualquier información estructurada.
- Un superconjunto del HTML.
  - No. Aunque el HTML puede ser definido con sintaxis de XML.
- Un invento de [x compañía].
  - No. XML es un estándar creado por el W3C y soportado por compañías e instituciones de todo el mundo.

Lenguajes Específicos

- HTML — ejemplo

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 FINAL//EN">
<HTML>
  <HEAD>
    <TITLE>Memo</TITLE>
  </HEAD>
  <BODY>
    <FONT FACE="Times New Roman" SIZE="2">
      <P>
        <B>To: </B>Pedro<BR>
        <B>From: </B>Juana<BR>
        <B>Cc: </B>Nacho<BR>
        <B>Subject: </B> Capítulo 1
      </P>
      <P> Qué opinas del formato? </P>
    </FONT>
  </BODY>
</HTML>
```

Ejemplo en XML

- XML — ejemplo

```
<?xml version="1.0"?>
<MEMO>
  <TO>Juana</TO>
  <FROM>Pedro</FROM>
  <CC>Nacho</CC>
  <SUBJECT>Capítulo 1</SUBJECT>
  <BODY>Qué opinas del formato? </BODY>
</MEMO>
```

Sintaxis simple

Legible por personas y máquinas

Muy parecido al HTML

El XML sirve para...

- Formato de intercambio de datos
  - Sistemas heredados
  - Integración de sistemas heterogéneos
- Publicación de datos
  - En diversos formatos (HTML, WML, PDF, etc.) a través de transformaciones XSLT
- Repositorios de datos
  - Bases de datos nativas XML
  - Lenguajes de consulta y actualización: XQuery, XQL, XUpdate, etc.
- Archivos de configuración y log
  - Aplicaciones, servidores Web, motores de Servlets, descripción de componentes EJB, etc.
  - Sistema operativo

El documento XML bien formado (*well-formed XML*)

Documentos XML bien formados (*well-formed XML*)

- XML es mas relajado que HTML en cuanto a las etiquetas (*tags*) que se pueden usar
- XML es mas estricto sobre donde y como se ponen estas etiquetas.
- XML *bien formado* requiere (entre otros):
  - Las etiquetas de inicio y final coinciden.
  - Los elementos vacíos tienen una forma especial.
  - Hay elementos anidados pero no traslapados.
  - Los atributos van en comillas.
  - Un elemento no pueden tener dos atributos con el mismo nombre
- Si un documento esta bien formado un parser puede leerlo y entenderlo.

Elemento

- Un elemento va desde su etiqueta de inicio hasta la respectiva etiqueta de termino

```
<Curso>
   XML
</Curso>
```

```
<Curso>
 <Nombre>XML</Nombre>
</Curso>
```

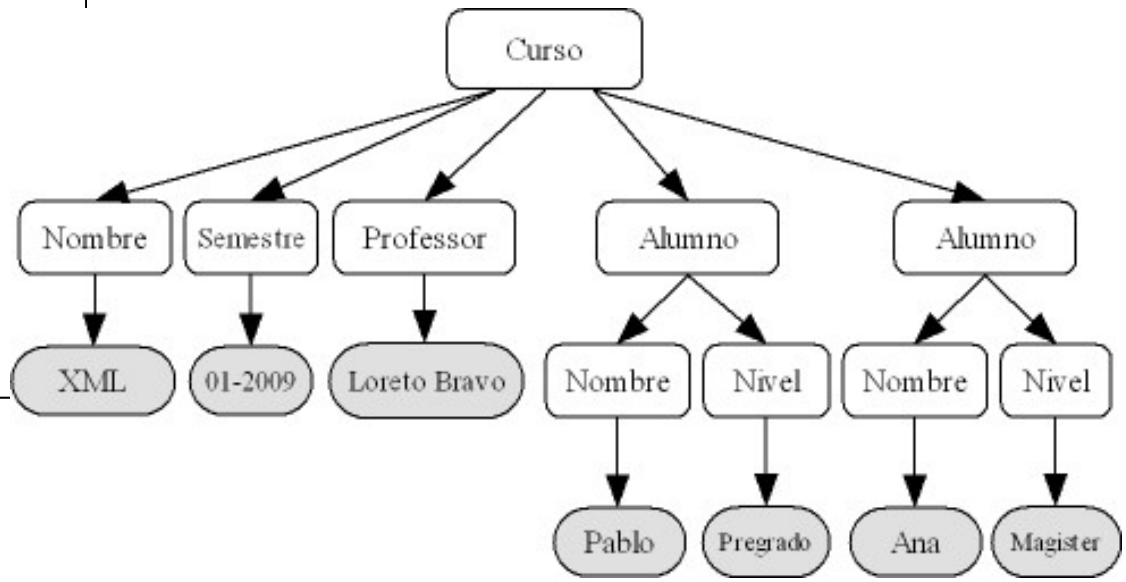
- Cinco cosas necesarias:
  - Nombre elemento.
  - Dónde se inicia el elemento.
  - Dónde termina el elemento.
  - Contenido del elemento.

Elementos

- Los elementos no se deben traslapar.
- Todo elemento tiene una etiqueta (*tag*) de inicio y una de final.
  - Inicia con **`<Nombre_elemento>`**
  - Termina con **`</Nombre_elemento>`**
- Elementos vacíos:
  - **`<Nombre_elemento></Nombre_elemento>`**
  - **`<Nombre_elemento/>`**
- El XML diferencía entre mayúsculas y minúsculas.
  - **`<Libro>`**, **`<libro>`**, **`<LIBRO>`** y **`<LiBrO>`** son etiquetas que se refieren a diferentes elementos.

Arboles XML

```
<?xml version="1.0"?>
<curso>
  <nombre>XML</nombre>
  <semestre>01-2009</semestre>
  <profesor>Loreto Bravo</profesor>
  <alumno>
    <nombre>Pablo</nombre>
    <nivel>pregrado</nivel>
  </alumno>
  <alumno>
    <nombre>Ana</nombre>
    <nivel>magister</nivel>
  </alumno>
</curso>
```



- Un único elemento raíz (*root*).
- Los elementos en la raíz aparecen secuencialmente o anidados.
- Hay orden entre los hijos

Bien formado?

```
<?xml version="1.0"?>
<Configuracion>
      <Impresora>
            <Nombre>Xerox 3119</Nombre>
            <Controlador>xe3119.dll</Controlador>
            <Sitio>\\sol\xe3119</Sitio>
            <Opciones>
                  <AlimentadorSobres/>
                  <Scanner/>
                  <Fotocopiadora/>
            </Opciones>
      </Impresora>
</Configuracion>
```

Si!

Bien formado?

```
<?xml version="1.0"?>
<Configuracion>
      <Impresora>
            <Nombre>Xerox 3119</Nombre>
      <Controlador>xe3119.dll</Controlador>
      <Sitio>\\sol\xe3119</Sitio>
      <Opciones>
            <AlimentadorSobres></AlimentadorSobres>
            <Scanner></Scanner>
            <Fotocopiadora/>
      </Opciones>
  </Impresora>
</Configuracion>
```

Si!

Bien Formado?

```
<?xml version="1.0"?>
<Configuracion>
        <Impresora>
        <Nombre>Xerox
   3119<Controlador>xe3119.dll</Nombre>
        </Controlador>
        <Sitio>\\sol\xe3119</Sitio>
        <Opciones>
                <AlimentadorSobres>
                <Scanner/>
                <Fotocopiadora/>
        </Opciones
   </Impresora>
</configuracion>
</Configuracion>
```

No!

Atributos.

- Propiedades de los elementos
- Contienen información acerca del elemento.
  - Unidades de medida.
  - Fechas, colores, etc.
  - Idioma
  - identificador
- Aparecen en la etiqueta de inicio:

```
<Nombre_elemento Nombre_atributo=“valor”>
                 ó
<Nombre_elemento Nombre_atributo=‘valor’>
```

Ejemplo con atributos.

```
<?xml version="1.0"?>
<Configuracion>
      <Impresora local="si">
            <Nombre>Xerox 3119</Nombre>
            <Controlador instalado="si">
                  xe3119.dll</Controlador>
            <Sitio>\\sol\xe3119</Sitio>
            <Opciones>
                  <AlimentadorSobres/>
                  <Scanner/>
                  <Fotocopiadora/>
            </Opciones>
      </Impresora>
</Configuracion>
```

Atributo o elemento?

```
<alumno>
   <nombre>Pedro</nombre>
   <apellido>Perez</apellido>
</alumno>
```

```
<alumno nombre='Pedro' apellido='Perez'/>
```

- Mucha discucion al respecto!
  - Hay casos en los que es claro:
    - Multiples valores -> Elemento
    - IDs -> atributo
  - En general es decision del diseñador
    - Ser consistente!

La declaración XML

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
```

- Dice "¡Soy un documento XML!"
- Tiene partes específicas:

`<?xml`	apertura
`version="1.0"`	versión
`encoding="ISO-8859-1"`	codificación de caracteres
`standalone="yes"`	doc. independ. (yes/no)
`?>`	fin

- XML tiene dos versiones:
  - Version 1.0: nombres de elementos y atributos en Unicode 2.0
  - Version 1.1: nombres de elementos y atributos en Unicode 3.0
    - Agrega letras de Burmese, Mongolian, Thaana, Cambodian, Yi y Amharic

La declaración XML

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
```

- Encoding:
  - Se refiere a la codificacion del texto en el documento XML
  - El default es UTF-8
  - Si no esta en UTF-8 debe contener la declaración de codificación (*encoding*).
  - Para tener texto en español: `ISO-8859-1`

- Standalone:
  - Esta toda la informacion necesaria en este documento?
  - Si se hace a referencia a otros documentos o estructuras que no esta en este documento:
    - `standalone="no"`

Ejemplos de declaraciones XML.

```
(ninguna)

<?xml version="1.0"?>

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

<?xml version="1.0" standalone="yes"?>

<?xml version="1.0" encoding="ASCII" standalone="no"?>
```

Nombres XML (*XML names*)

- Los nombres de elementos y atributos:
  - Pueden contener letras (incluso con acento!), numeros, y algunos simbolos.
  - Deben de iniciar con una letra, underscore (_) o dos puntos (:)
  - Los caracteres siguientes pueden ser letras, números, puntos, guiones (-), subrayados (_) o dos puntos (:).
  - El nombre “XML” y sus variaciones están reservadas.
- Ejemplos
  - <dia-mes-año>30/11/1975</dia-mes-año>
  - <_2-pistas/>
- Ejemplos no permitidos:
  - <dia/mes/año>30/11/1975</dia/mes/año>
  - <2-pistas/>
  - <dos pistas>
  - <Driver’s_Licence>

Entidades carácter

- Para documentos bien formados:

>	`>`	(greater than)
<	`<`	(less than)
&	`&`	(ampersand)
’	`&apos;`	(apóstrofe)
”	`&quot;`	(double quote)

- Ejemplos:
  - `<optica>Rotter & Krauss</optica>`
  - `<Nombre>Marcelo &apos;Chino&apos; Rios</Nombre>`

Comentarios e Instrucciones

- Comentarios:

```
<!-- Revisar los valores -->
```

  - Los parsers se los pueden saltar
  - Pueden estar dentro del root, pero no dentro del resto de los elementos

- Instrucciones:

```
<?robots index="yes" follow="no"?>
<?php mysql_connect()...       ?>
<?xml-stylesheet href="curso.css" type="text/css"?>
<?xml version="1.0"?>
```

Orden

- El orden importa en elementos

```
<alumnos>
   <nombre>Pedro</nombre>
  <nombre>Juan</nombre>
</alumnos>
```

distinto a

```
<alumnos>
  <nombre>Juan</nombre>
  <nombre>Pedro</nombre>
</alumnos>
```

- El orden no importa en atributos

```
<persona nacimiento='10-11-1910' muerte='02-05-1998'/>
```

igual a

```
<persona muerte='02-05-1998' nacimiento='10-11-1910'/>
```

Contenido Mixto

- Hasta el momento, un elemento contiene:
  - Texto sin etiquetas
  - Elementos
- Esto es común en un documento de datos, pero no en un documento narrativo:

```
<biografia>
  <parrafo> Lucila de María del Perpetuo Socorro Godoy
  Alcayaga, llamada <nombre>Gabriela Mistral</nombre>
  nacio <nacimiento>7 de abril de 1889</nacimiento> y
  murio el <muerte>10 de enero de 1957</muerte>.
  </parrafo>
</biografia>
```

- Contenido mixto es comun en articulos, ensayos, historias, paginas web.

Datos Estructurados o no?

```
Lucila de María del Perpetuo
Socorro Godoy Alcayaga, llamada
Gabriela Mistral nació el 7 de
abril de 1889 y murió el 10 de
enero de 1957.
```

Sin estructura

Lucila de María del Perpetuo Socorro Godoy Alcayaga, llamada *Gabriela Mistral* nació el 7 de abril de 1889 y murió el 10 de enero de 1957.

Sin estructura

```
<?xml version="1.0"?>
<biografia>
  <parrafo> Lucila de María del
   Perpetuo Socorro Godoy
   Alcayaga, llamada
   <nombre>Gabriela
   Mistral</nombre> nacio
   <nacimiento>7 de abril de
   1889</nacimiento> y murio el
   <muerte>10 de enero de
   1957</muerte>.
  </parrafo>
</biografia>
```

Semi- estructurado

- Documentos narrativos pueden ser considerados sin estructura o semi-estructurados

Datos Estructurados o no?

```
<?xml version="1.0"?>
<curso>
  <nombre>XML</nombre>
  <semestre>01-2009</semestre>
  <profesor>Loreto Bravo</profesor>
  <alumno>
    <nombre>Pablo</nombre>
    <nivel>pregrado</nivel>
  </alumno>
  <alumno>
    <nombre>Ana</nombre>
    <nivel>magister</nivel>
  </alumno>
</curso>
```

```
<?xml version="1.0"?>
<curso>
  <nombre>XML</nombre>
  <semestre>01-2009</semestre>
  <profesor>Loreto Bravo</profesor>
  <alumno>
    <nombre>Pablo</nombre>
    <nivel>pregrado</nivel>
  </alumno>
  <sala>B01</sala>
</curso>
```

- Documentos sin contenidos mixtos pueden ser considerados semi-estructurados o estructurados
- En la presencia de un esquema (DTD, XSD) son estructurados

¿Qué editor puedo usar?

- Requisitos mínimos:
  - No debe generar caracteres EOF al final del archivo.
  - No debe generar tabulaciones (si se usan deben de expandirse a espacios al grabar).
- Sugerencias:
  - Cualquier editor de texto o procesador de palabras.
  - Editores especiales para XML.
    - Cooktop
    - Oxygen
    - libxml

Ejercicio

- Crear un documento XML
- Visualizar en algun navegador (firefox, chrome,...) para chequear si está bien formado

DTD

Document Type Definition (DTD)

- Define un lenguaje especifico de etiquetas para alguna aplicación
- Define los tipos de ***elementos***, ***atributos*** y ***entidades*** permitidas, y puede expresar algunas limitaciones para combinarlos
- Puede estar:
  - Definido en un archivo externo y ser compartido por varios documentos XML, o
  - Contenido en el propio documento XML como parte del prólogo.
- Un documento XML es ***valido*** si se se ajusta a su DTD

Elemento raiz

```
<?xml version="1.0"?>
<!DOCTYPE direccion [
<!ELEMENT direccion (nombre, calle, ciudad, pais, codigo)>
<!ELEMENT nombre (#PCDATA)>
<!ELEMENT calle (#PCDATA)>
<!ELEMENT ciudad (#PCDATA)>
<!ELEMENT pais (#PCDATA)>
<!ELEMENT codigo (#PCDATA)>
]>
<direccion>
  <nombre>Juan Perez</nombre>
  <calle>Barros Arana 454</calle>
  <ciudad>Concepcion</ciudad>
  <pais>Chile</pais>
  <codigo>4030000</codigo>
</direccion>
```

```
<!DOCTYPE course SYSTEM "http://www.unitime.org/interface/CourseOfferingExport.dtd">
<course>
  <class>...</class>
  ...
</course>
```

Declaraciones de tipo Elemento

- Deben empezar con **“<!ELEMENT**” seguidas por el identificador genérico del elemento que se declara
- A continuación tienen una especificación del contenido
- Ejemplo:

```
<!ELEMENT receta (titulo, ingredientes, procedimiento)>
```

Documento XML no válido

```
<receta>
  <parrafo>La siguiente receta me la pasó Alvaro</parrafo>
  <titulo>Arroz cocido</titulo>
  <ingredientes>Arroz</ingredientes>
  <procedimiento>Cocer el arroz</procedimiento>
</receta>
```

Elementos

- `<!ELEMENT objeto #PCDATA>`

- `<!ELEMENT linea-horizontal EMPTY>`
  - linea-horizontal no tiene contenido.
- `<!ELEMENT objeto ANY>`
  - El elemento objeto puede tener cualquier contenido. Es mejor no usarla y estructurar adecuadamente los documentos!
- `<!ELEMENT aviso (parrafo)>`
  - <aviso> sólo puede contener un elemento <parrafo>

Elementos

- **<!ELEMENT aviso (titulo, parrafo)>**
  - <aviso> debe contener un elemento <titulo> seguido de un elemento <parrafo>
- **<!ELEMENT aviso (parrafo | grafico)>**
  - La barra vertical "|" indica opción. El número de opciones no está limitado y se pueden agrupar usando paréntesis
- **<!ELEMENT aviso (titulo, (parrafo | grafico))>**
  - <aviso> debe contener un <titulo> seguido de un <parrafo> o un <grafico>
- **<!ELEMENT aviso (titulo?, (parrafo+, grafico)*)>**
  - ? → 0 ó 1 vez (opcional)
  - * → 0 ó más veces
  - + → 1 ó más veces

Ejemplo

DTD

```
<!ELEMENT curso (nombreCurso, semestre, profesor, alumno*)>
<!ELEMENT nombreCurso #PCDATA>
<!ELEMENT semestre #PCDATA>
<!ELEMENT profesor #PCDATA>
<!ELEMENT alumno (nombre,nivel,email?)>
<!ELEMENT nombre #PCDATA>
<!ELEMENT nivel #PCDATA>
<!ELEMENT email #PCDATA>
```

Documento XML

```
<?xml version="1.0"?>
<curso>
  <nombreCurso>XML</nombreCurso>
  <semestre>01-2009</semestre>
  <profesor>Loreto
Bravo</profesor>
  <alumno>
    <nombre>Pablo</nombre>
    <nivel>pregrado</nivel>
  </alumno>
  <alumno>
    <nombre>Ana</nombre>
    <nivel>magister</nivel>
    <email>ana@udec.cl</email>
  </alumno>
</curso>
```

Ejemplo

- Existe un DTD que valide estos documentos?

DTD

```
<!ELEMENT curso (nombreCurso|
   semestre| profesor)*>
<!ELEMENT nombreCurso #PCDATA>
<!ELEMENT semestre #PCDATA>
<!ELEMENT profesor #PCDATA>
```

Documento XML 1

```
<?xml version="1.0"?>
<curso>
  <nombreCurso>XML</nombreCurso>
  <semestre>01-2009</semestre>
  <profesor>Loreto</profesor>
</curso>
```

Documento XML 2

```
<?xml version="1.0"?>
<curso>
  <semestre>01-2009</semestre>
  <nombreCurso>XML</nombreCurso>
 <profesor>Loreto</profesor>
</curso>
```

Documento XML 3

```
<?xml version="1.0"?>
<curso>
  <profesor>Loreto</profesor>
  <semestre>01-2009</semestre>
  <nombreCurso>XML</nombreCurso>
</curso>
```

Ejemplo: contenido mixto

Documento XML

```
<biografia>
  <parrafo> Lucila de María del Perpetuo Socorro
   Godoy Alcayaga, llamada <nombre>Gabriela
   Mistral</nombre> nacio <nacimiento>7 de abril de
   1889</nacimiento> y murio el <muerte>10 de enero
   de 1957</muerte>.
  </parrafo>
</biografia>
```

DTD?

- Cuando un elemento tiene contenido mixto (texto y elementos) se puede representar de una sola forma:

```
<!ELEMENT parrafo(#PCDATA|nombre|nacimiento|muerte)*>
…
```

Declaración de lista de Atributos

- Atributos
  - Añaden información adicional a los elementos
  - Sólo se pueden especificar una vez y en cualquier orden
  - No pueden contener sub-atributos
- Declaración

```
<!ATTLIST elemento atributo tipo porDefecto>
```

  - `elemento:` Identificador del elemento al que se aplica
  - `atributo:` nombre del atributo
  - `tipo:` tipo del atributo
  - `porDefecto:` declaración por defecto

Ejemplo

<!ATTLIST elemento atributo tipo porDefecto>

```
<!ELEMENT mensaje (de, a, texto)>
<!ATTLIST mensaje prioridad (normal | urgente) normal>
<!ELEMENT texto (#PCDATA)>
<!ATTLIST texto idioma CDATA #REQUIRED>
```

```
<mensaje prioridad="urgente">
  <de>Topacio Jade</de>
  <a>Esmeralda Turquesa</a>
  <texto idioma="español">
  Hay que preparar los informes de junio
  </texto>
</mensaje>
```

Declaración por defecto

- **#REQUIRED**: Es obligatorio especificar el atributo. No tiene valor por defecto.
- **#IMPLIED**: attributo opcional. Se puede omitir el atributo, sin que se adopte automáticamente un valor por defecto
- Literal: string que contiene el valor por defecto

```
<!ATTLIST IMG URL CDATA #REQUIRED
              ALT CDATA #IMPLIED>

<!ATTLIST mensaje prioridad (normal | urgente) normal>
```

ID, IDREF, IDREFS

- El tipo **ID** permite que un atributo determinado tenga un nombre único en el documento que podrá ser referenciado por un atributo de otro elemento que sea de tipo **IDREF**
- Contenido de estos atributos debe cumplir con los requisitos de *XML names*
- Permite implementar un sistema de hipervínculos en un documento XML

```
<!ATTLIST link        destino       IDREF   #REQUIRED>
<!ATTLIST capitulo    referencia    ID      #IMPLIED>
```

```
<!ATTLIST empleado RUT         ID #REQUIRED>
<!ATTLIST proyecto idProyecto ID #REQUIRED
                   equipo     IDREFS #REQUIRED>
```

Tipos de Atributos

- 10 tipos de Atributos:
  - CDATA: character data
  - NMTOKEN
  - NMTOKENS
  - Enumerados
  - ENTITY
  - ENTITIES
  - ID
  - IDREF
  - IDREFS
  - NOTATION

Ejemplo

Lista.dtd

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

<!ELEMENT lista (persona)+>
<!ELEMENT persona (nombre, email*, relacion?)>
<!ATTLIST persona id ID #REQUIRED>
<!ATTLIST persona sexo (hombre | mujer) #IMPLIED>
<!ELEMENT nombre (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT relacion EMPTY>
<!ATTLIST relacion amigo-de IDREFS #IMPLIED
                   enemigo-de IDREFS #IMPLIED>
```

```
<?xml version="1.0"?>

<!DOCTYPE lista SYSTEM "Lista.dtd">
<lista>
  <persona sexo="hombre" id="Pedro">
    <nombre>Pedro Perez</nombre>
    <email>pedro@udec.com</email>
    <relacion amigo-de="Ana Marta">
  </persona>
  <persona sexo="mujer" id="Ana">
    <nombre>Ana Andrade</nombre>
    <email>ana@gmail.com</email>
  </persona>
 <persona sexo="mujer" id="Marta">
    <nombre>Marta Ruiz</nombre>
    <email>mruiz@gmail.com</email>
    <email>mruiz@udec.cl</email>
    <relacion enemigo-de="Ana" amigo-de="Pedro">
  </persona>
</lista>
```

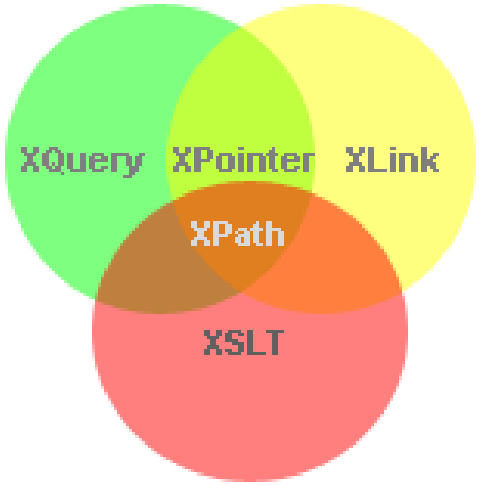
Validando un documento XML

- Los navegadores chequean que un documento este bien formado, no que sea valido
- Opciones:
  - Parseador
    - Cooktop
    - xmllint
  - En linea:
    - Brown University http://www.stg.brown.edu/service/xmlvalid/
    - Richard Tobin's XML validator http://www.cogsci.ed.ac.uk/~richard/xml-check.html

DTDs

- Applicaciones de XML pueden (y muchas veces son) definidas con DTDs:
  - SVG
  - RDF
  - MathML
  - RSS
  - XHTML
  - OpenOffice
- Donde se pueden algunos DTDs estandares:
  - http://xml.coverpages.org/xml-rfc.html
  - www.w3c.org/TR/

XML Path Language (XPath)



Loreto Bravo

lbravo@gmail.com

Introducción

- Es un lenguaje para encontrar información en un documento XML.
- Es utilizado para navegar a través de elementos y atributos para localizar nodos.
- Esta diseñado para procesar árboles y facilitar la referencia de los datos en la jerarquía.
- Es el mayor componente en el estándar XSLT, XQuery y XPointer utilizan las expresiones XPath.
- Se convirtió en un estándar recomendado por la W3C en 1999

XPath

```
/book/chapter[@author="Marcela Paz"]
```

- Expresiones:
  - Se utilizan expresiones para seleccionar nodos o conjuntos de nodos.
  - Son expresiones similares a las utilizadas en el sistema de archivos tradicional
- Funciones:
  - Incluye cerca de 100 funciones.
  - Existen funciones para valores de cadenas, numéricas, fechas, comparación de hora, nodos, manipulación del QName, secuencias, valores booleanos, etc.

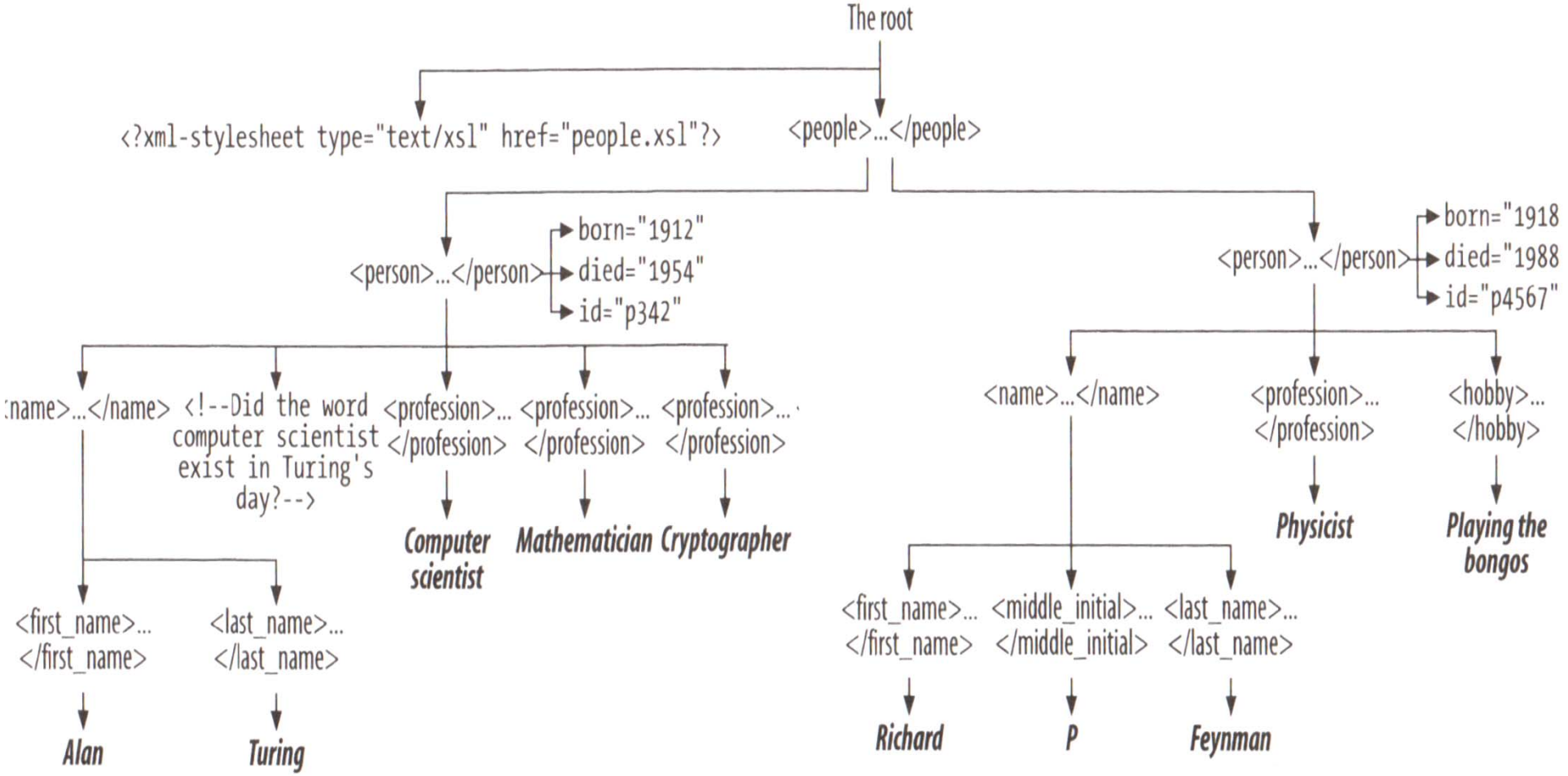
Estructura de arbol para XPath

- Documento es un arbol compuesto de nodos de distintos tipos:
  - raiz (*root*)
    - contiene todo el documento
    - No es un elemento del documento XML
    - Contiene como hijo al elemento raiz
  - elementos
  - texto
  - atributos
  - comentarios
  - Instrucciones de procesamiento
  - Namespaces

Ejemplo: people.xml

```
<?xml version="1.0"?>
<?xml-stylesheet type="application/xml" href="people.xsl"?>
<people>
  <person born="1912" died="1954" id="p342">
    <name>
      <first_name>Alan</first_name>
      <last_name>Turing</last_name>
    </name>
    <!-- Did the word computer scientist exist in Turing's day? -->
    <profession>computer scientist</profession>
    <profession>mathematician</profession>
    <profession>cryptographer</profession>
    <homepage xlink:href="http://www.turing.org.uk/"/>
  </person>
  <person born="1918" died="1988" id="p4567">
    <name>
      <first_name>Richard</first_name>
      <middle_initial>&#x50;</middle_initial>
      <last_name>Feynman</last_name>
    </name>
    <profession>physicist</profession>
    <hobby>Playing the bongoes</hobby>
  </person>
</people>
```

people.xml como arbol



Trayectorias

- XPath utiliza expresiones de trayectorias para seleccionado nodos o conjuntos de nodos en un documento XML.
  - Una trayectoria esta compuesta de uno o más pasos
- Se pueden especificar trayectorias:
  - Absolutas: se parte desde la *raíz* del documento.
    - Las consultas parte con: **/**
  - Relativas: se parte desde el *nodo de contexto*
    - Nodo de contexto: nodo desde el cual se hace la consulta
    - XSLT y XPointer definen en su lenguaje el nodo contexto

Trayectorias

- Las trayectorias se componen a partir de uno o más *pasos*
- Cada paso se evalua de acuerdo a su nodo de contexto

Pasos

`/`	Selecciona desde el nodo raíz del documento (absoluto)
`nombre_elemento`	Selecciona todos los elementos con nombre `nombre_elemento` hijos del nodo contexto
`@nombreAt`	Selecciona los atributos con nombre `nombreAt` hijos del nodo contexto
`comment()`	Selecciona los comentarios hijos del nodo contexto
`text()`	Selecciona los nodos de texto hijos del nodo contexto
`Processing-instruction()`	Selecciona los nodos con instrucciones hijos del nodo contexto. Puede tener un parametro
`*`	Cualquier nodo elemento
`@*`	Cualquier nodo atributo
`node()`	Cualquier nodo
`A|B`	nodos A o B

Ejemplos people.xml

/	devuelve el documento completo
profession (context node= Alan Turing **person**)	devuelve los tres nodos de profession bajo el nodo de contexto
profession (context node= Richard Feynman **person**)	devuelve el unico nodo de profession bajo el nodo de contexto
profession (context node= **people**)	devuelve null ya que `people` no tiene ningun hijo `profession`
@born (context node= Alan Turing **person**)	`@born="1912"`
processing-instruction() (context node= the root)	devuelve el nodo de instruccion para `xml-stylesheet`
processing-instruction('xml-stylesheet') (context node= the root)	devuelve el nodo de instruccion para `xml-stylesheet`

Trayectorias Compuestas (secuencias de pasos)

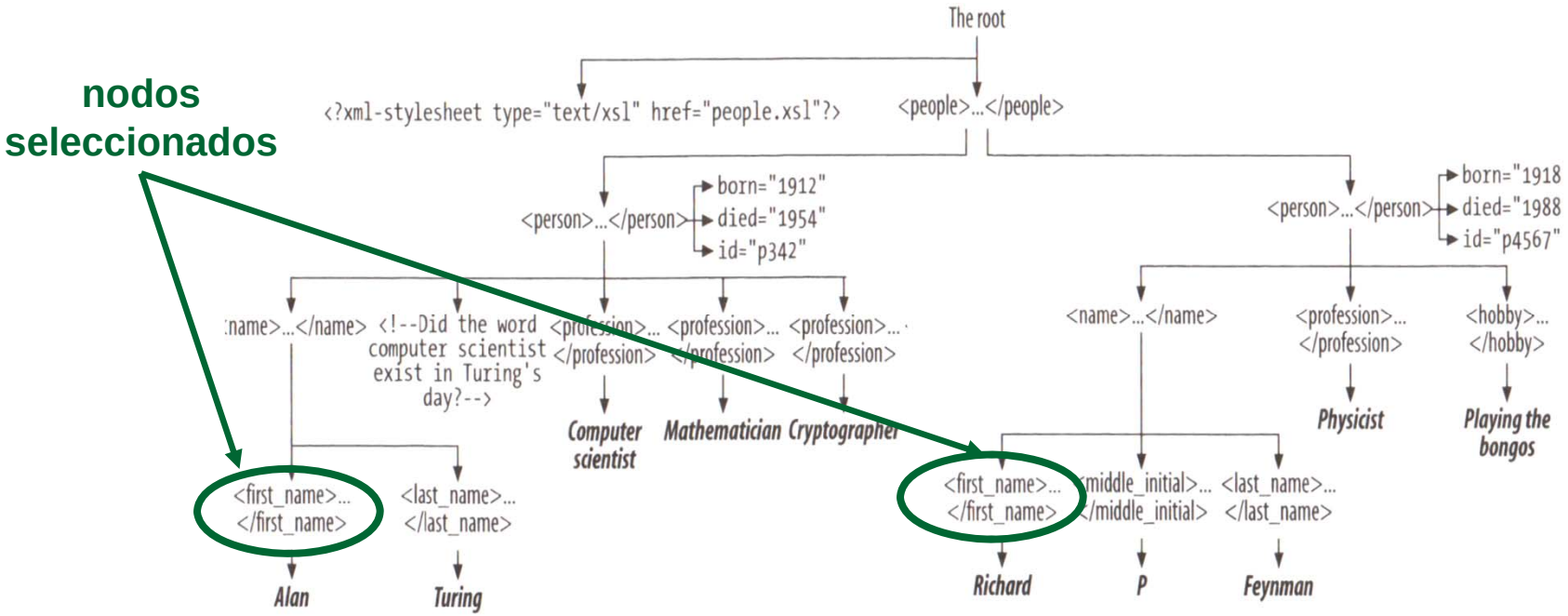
/	Separa dos pasos. El segundo es relativo al anterior
//	Selecciona todos los descendientes del nodo contexto y el nodo mismo. Si // aparece al comienzo de la expression XPath, es absoluta y se refiere a todos los descendintes de la raiz y a la raiz
.	Selecciona el nodo actual
..	Selecciona el padre del nodo actual

Ejemplos people.xml

/people/person/name /first_name	parte de la raiz, selecciona todos los nodos `people`, luego todos los hijos `person` de estos nodos, luego todos los nodos `name` de estos y finalmente selecciona los hijos `first_name`.
/people/person/name /first_name/text()	devuelve el texto dentro de los elementos del ejemplo anterior
person/@id	primero seleccion los nodos `person` que son hijos del nodo de contexto y luego devuelve los atributos `id` de ellos
//name	devuelve todos los nodos `name` que estan en *cualquier* parte del documento
person//name	primero selecciona hijo `person` del nodo de contexto y luego retorna todos sus descendientes `name`.
people//@id	selecciona todos los atributos `id` que sean descendientes de un hijo `people`.
//middle_initial/.. /first_name	devuelve todos los `first_name` que son hermanos de un elemento `middle_initial`

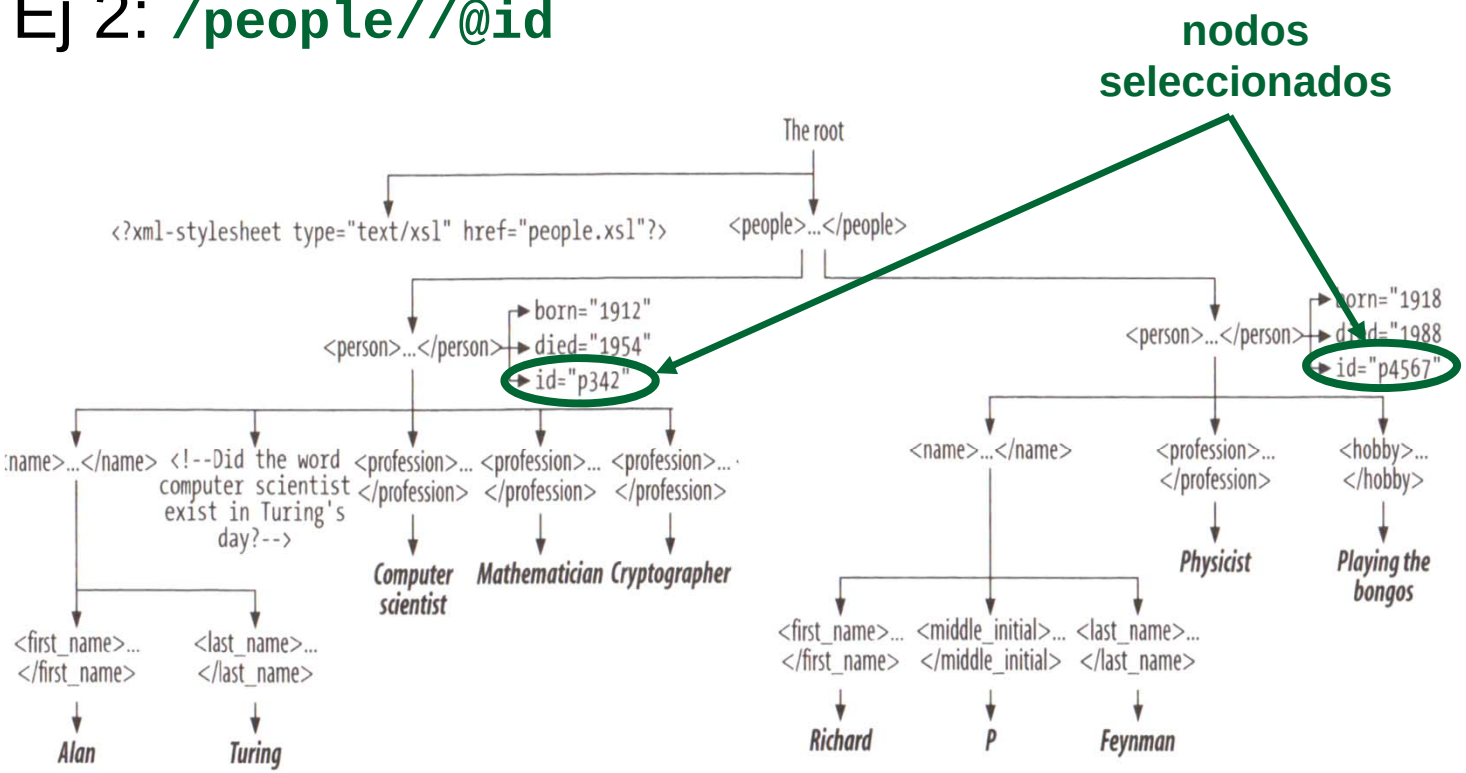
Xpath

- Las expresiones de trayectoria en XPath devulven set de *nodos*
  - Ej 1: **`/people/person/name/first_name`**



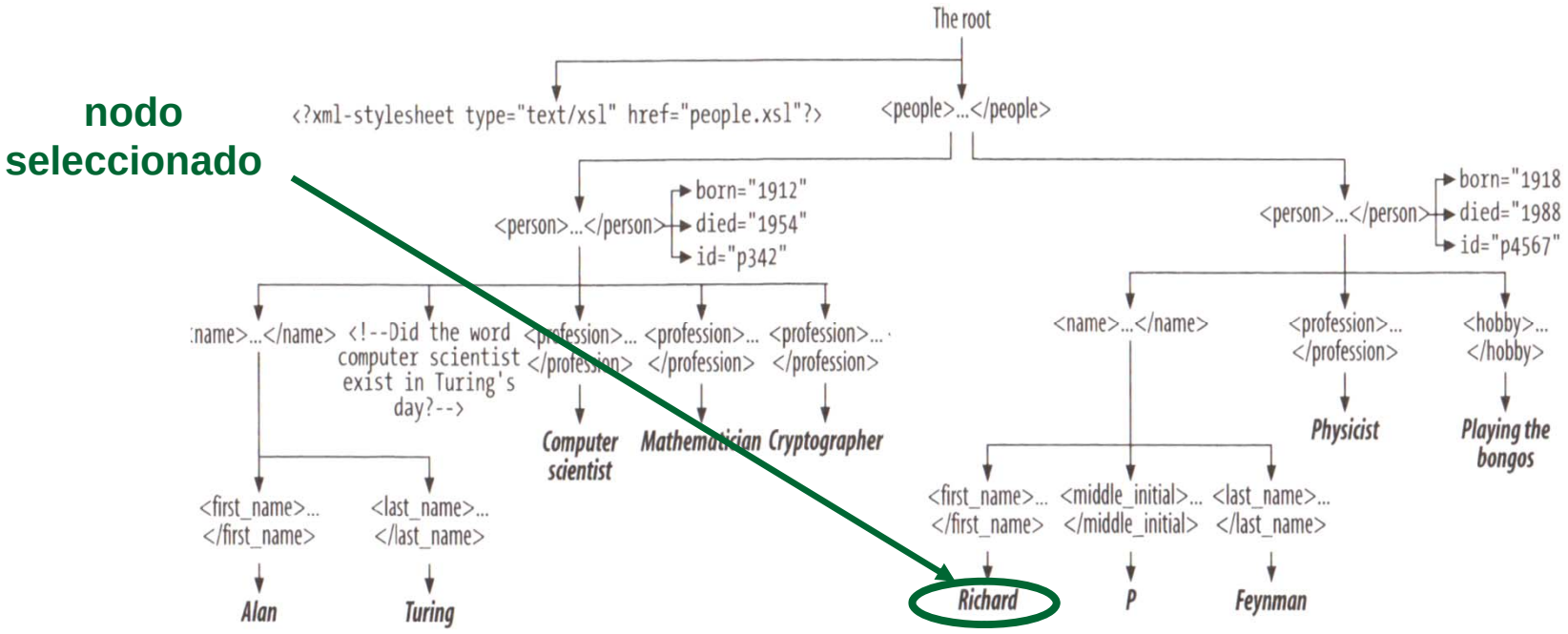
Xpath

- Las expresiones de trayectoria en XPath devulven set de *nodos*
  - Ej 2: **`/people//@id`**



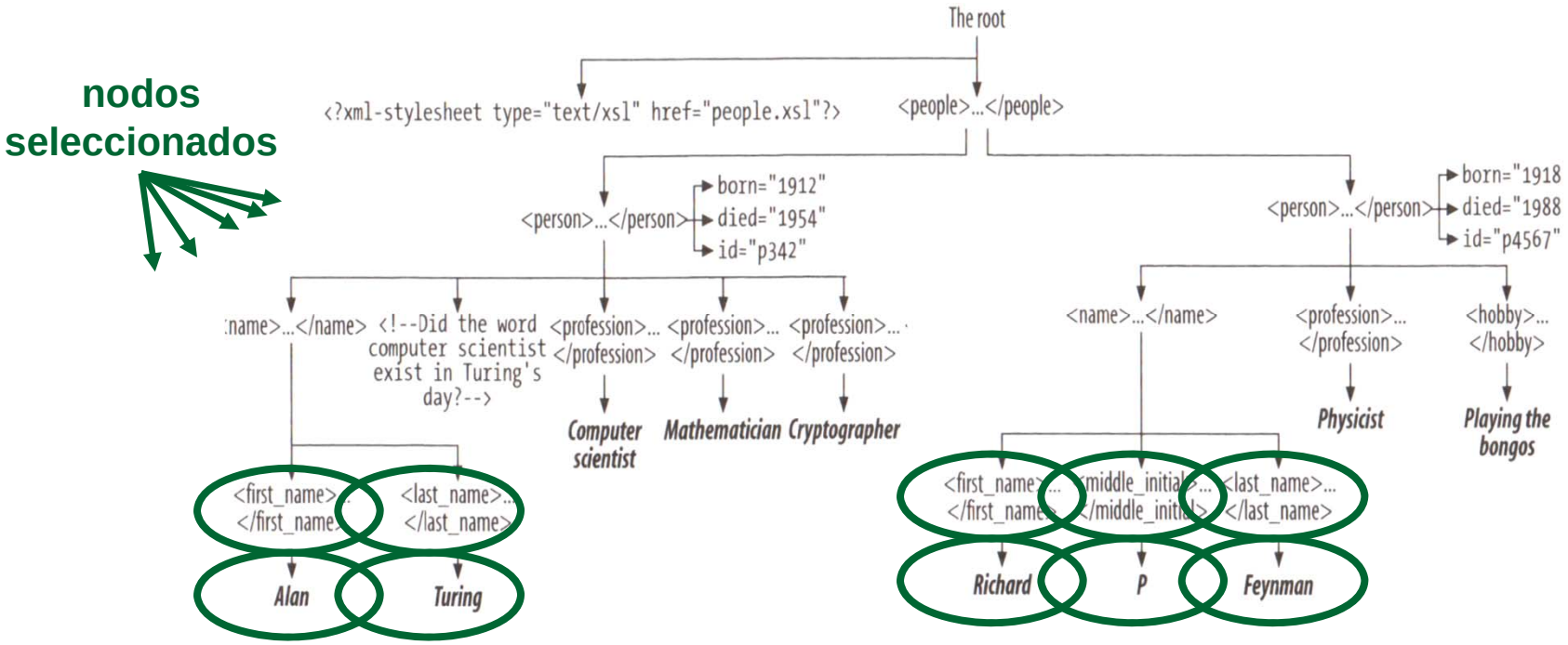
Xpath

- Las expresiones de trayectoria en XPath devulven set de *nodos*
  - Ej 3: **`//middle_initial/../first_name/text()`**



Xpath

- Las expresiones de trayectoria en XPath devulven set de *nodos*
  - Ej 4: **`//name//node()`**

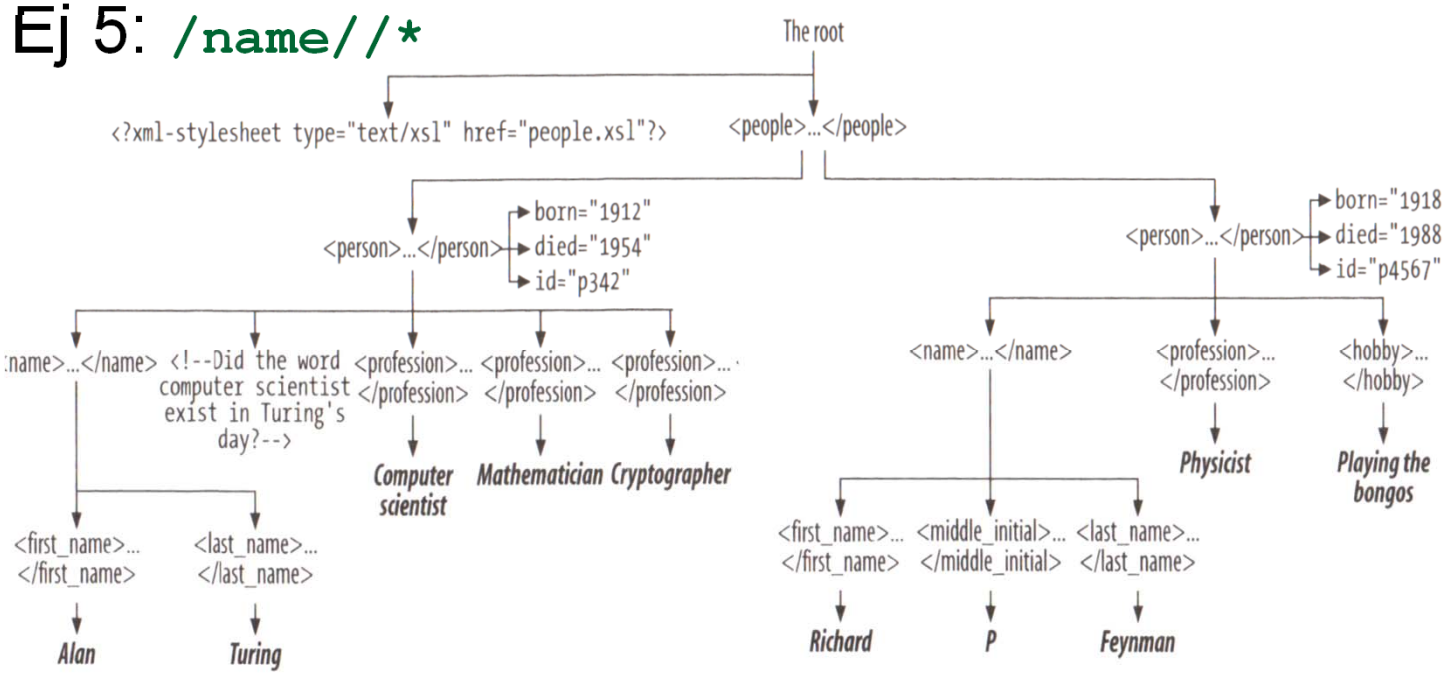


Xpath

- Las expresiones de trayectoria en XPath devulven set de *nodos*

  Null

  - Ej 5: **`/name//*`**



Predicados

- Las trayectorias que hemos definido hasta el momento
  - Nos permiten seleccionar nodos dependiendo de su posicion en el documento
  - No permiten poner condiciones sobre los contenidos
- Los predicados permiten poner condiciones extras sobre los nodos a seleccionar
  - Se ponen en paréntesis cuadrados [ ] al final de cada paso.
  - Puede haber mas de uno por paso
  - Contienen:
    - Condicion booleana: si es falsa el nodo se elimina de la solucion
    - Numero *n*: devuelvo el nodo en la posicion *n*
    - Sets de nodos: se elimina el nodo contexto si el set de nodos es vacio

Ejemplos people.xml

`//profession[.="physicist"]`	selecciona los nodos `profession` con valor physicist
`//person[profession="physicist"]`	selecciona los nodos `person` que tiene un hijo de `profession` physicist
`//person[@id="p4567"]`	devuelve las `personas` que tiene un atributo `id` con valor p4567
`//person[@born<=1976]`	devuelve las `personas` que tiene un atributo `born` con valor menor o igual a 1976
`//person[@born<=1976 and @born> 1900]`	devuelve las `personas` que tiene un atributo `born` con valor menor o igual a 1976 y mayor que 1900
`//name[2]`	selecciona el segundo elemento `name` del documento. En este caso seria Richard
`//name[middle_initial]`	selecciona los `name` que tienen al menos un `middle_initial`
`//name[not(middle_initial)]`	selecciona los `name` que no tienen ningun elemento `middle_initial`
`/people/person[@born<1950]/name[first_name=Alan]`	primero selecciona los nodos `people` bajo la raiz. Luego, elige los hijos `person` que nacieron antes de 1950. Entre ellos retorna los `name` que tiene como `first_name` Alan

Ejemplo bookstore

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- The file is still incomplete -->
<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>
```

Ejemplo bookstore

/bookstore	Selecciona el elemento raíz `bookstore`
bookstore/book	Selecciona todos los elementos `book` que son hijos de `bookstore`
//book	Selecciona todos los elementos `book` no importando donde estén.
bookstore//book	Selecciona todos los elementos `book` descendientes de `bookstore` no importando donde estén debajo de este.
//@lang	Selecciona todos los atributos llamados `lang`
/bookstore/*	Todos los elementos hijos de `bookstore`
//*	Todos los elementos del documento
//title[@*]	Todos los nodos `title` que tengan atributos

Ejemplo bookstore

`bookstore/book[1]`	Selecciona el primer elemento book
`bookstore/book[last()]`	Selecciona el ultimo elemento book
`bookstore/book[position()<3]`	Selecciona los primeros dos elementos book
`bookstore/book[price>35.00]`	Selecciona todos los book con precio mayor a 35.0
`//title[@lang='eng']`	Selecciona todos los elementos titulo que tienen atributo `lang` con valor ‘eng’

Otro ejemplo: libro

- DTD Libro:

Ojo con esto que no vimos antes!

```
<?xml encoding="UTF-8"?>
<!ENTITY % attSec "id CDATA #IMPLIED
                   author CDATA #IMPLIED
                   fecha CDATA #IMPLIED">
<!ELEMENT libro (capitulo)*>
<!ELEMENT capitulo (titulo?,intro?,seccion*)>
<!ELEMENT seccion (grafico|parrafo)*>
<!ELEMENT intro (grafico|parrafo)*>
<!ELEMENT figura (#PCDATA)>
<!ELEMENT parrafo (#PCDATA)>
<!ELEMENT titulo (#PCDATA)>
<!ATTLIST capitulo %attSec;>
<!ATTLIST intro %attSec;>
<!ATTLIST seccion %attSec;>
<!ATTLIST figura id CDATA #IMPLIED
                 img ENTITY #REQUIRED>
```

Ejemplo: Libro

libro	todos los elementos `libro` hijos del nodo contexto (`root`)
libro/*	todos los elementos nietos del nodo contexto e hijos de `libro`.
/libro/capitulo/seccion/grafico/text()	selecciona todos los nodos texto del nodo `grafico` bajo que son hijos de `seccion`, nietos de `capitulo` y bisnietos de `libro`
//seccion/@fecha	selecciona los atributos `fecha` de `seccion`
/libro/capitulo/seccion/@*	todos los atributos de nodos `seccion`
/libros/capitulos/seccion/parrafo[1]	primer `parrafo` de cada `seccion`
/libro/capitulo/seccion/parrafo[last()]	último `parrafo` de cada `seccion`
libro/*/titulo	todos los elementos `titulo` nietos de `libro`
libro//titulo	todos los elementos `titulo` descendientes de `libro`

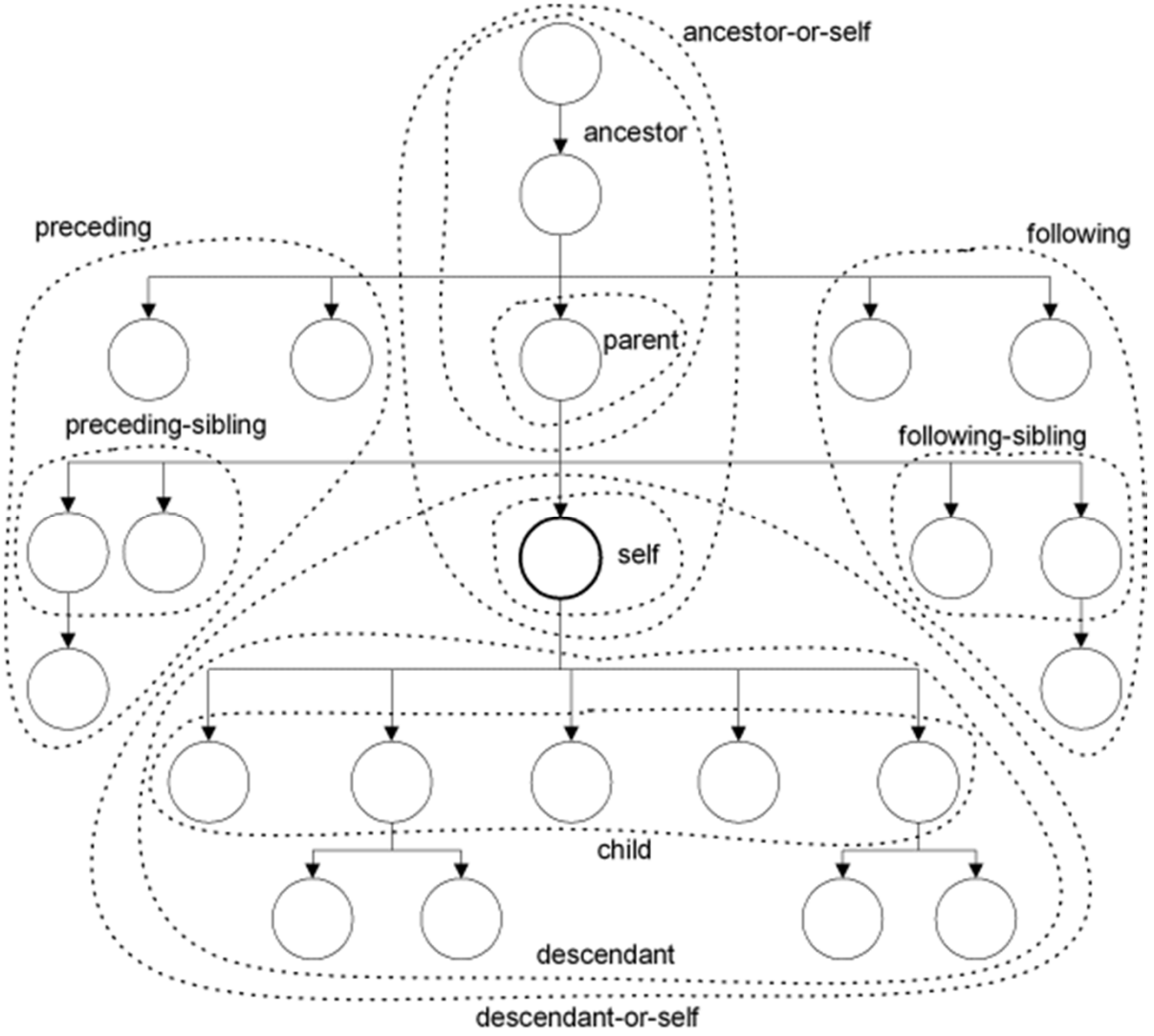
Ejemplo

`//titulo`	todos los elementos `titulo` descendientes del nodo contexto
`/libro/capitulo[1]/seccion[3]`	La tercera `seccion` del `capitulo` 1
`/libro/capitulo/comment()`	Los nodos comentario del elemento `capitulo`
`libro/capitulo/titulo/../@autor`	selecciona al atributo `autor` del elemento padre de `titulo`, es decir, el atributo de `capitulo` si es que este tiene un hijo `titulo`
`/libro/capitulo[@autor="Marcela"]`	el `capitulo` con `autor` Marcela
`/libro/capitulo/seccion[@fecha="10-10-2001"][2]`	la segunda `seccion` de cada `capitulo` entre aquellas que tenga por `fecha` "10-10-2001"
`libro/capitulo[intro]`	selecciona los nodos `capitulo` que tenga un elemento hijo llamado `intro`
`/libro/capitulo/seccion[@id][@fecha]`	selecciona los nodos `seccion` que tienen ambos atributos
`libro/capitulo/seccion[@id]|libro/capitulo/seccion[@fecha]`	Los elementos `seccion` que tengan el atributo `id` ó el atributo `fecha`

Trayectorias no abreviadas

- Hasta el momento hemos utilizado la version abreviada de XPath
  - Facil de usar
  - Simple
  - "Intuitiva"
- Existe una extendida:
  - Da más control
  - Es más explicita

Ejes


ancestor-or-self
ancestor
preceding
following
parent
preceding-sibling
following-sibling
self
child
descendant
descendant-or-self

Ejes

ancestor	Selecciona todos los ancestros del nodo actual (padre, abuelo, etc.)
ancestor-or-self	Selecciona todos los ancestros del nodo actual (padre, abuelo, etc.) y el nodo actual
attribute	Selecciona todos los atributos del nodo actual
child	Selecciona todos los hijos del nodo actual. No selecciona atributos ni namespace
descendant	Selecciona los descendientes del nodo actual. No selecciona atributos ni namespace
descendant-or-self	Selecciona los descendientes del nodo actual y el nodo actual. No selecciona atributos ni namespace
following	Selecciona todo en el documento que esta a partir del nodo actual. No selecciona atributos ni namespace
following-sibling	Selecciona los hermanos posteriores del nodo actual. No selecciona atributos ni namespace. Si el nodo de contexto es un atributo o namespace, retorna vacio
namespace	Selecciona todos los nodos con el namespace del nodo actual
parent	Selecciona el padre del nodo actual
preceding	Selecciona todo en el documento que esta antes del nodo actual. No selecciona atributos ni namespace
preceding-sibling	Selecciona todos los hermanos anteriores al nodo actual. Si el nodo de contexto es un atributo o namespace, retorna vacio
self	Selecciona el nodo actual

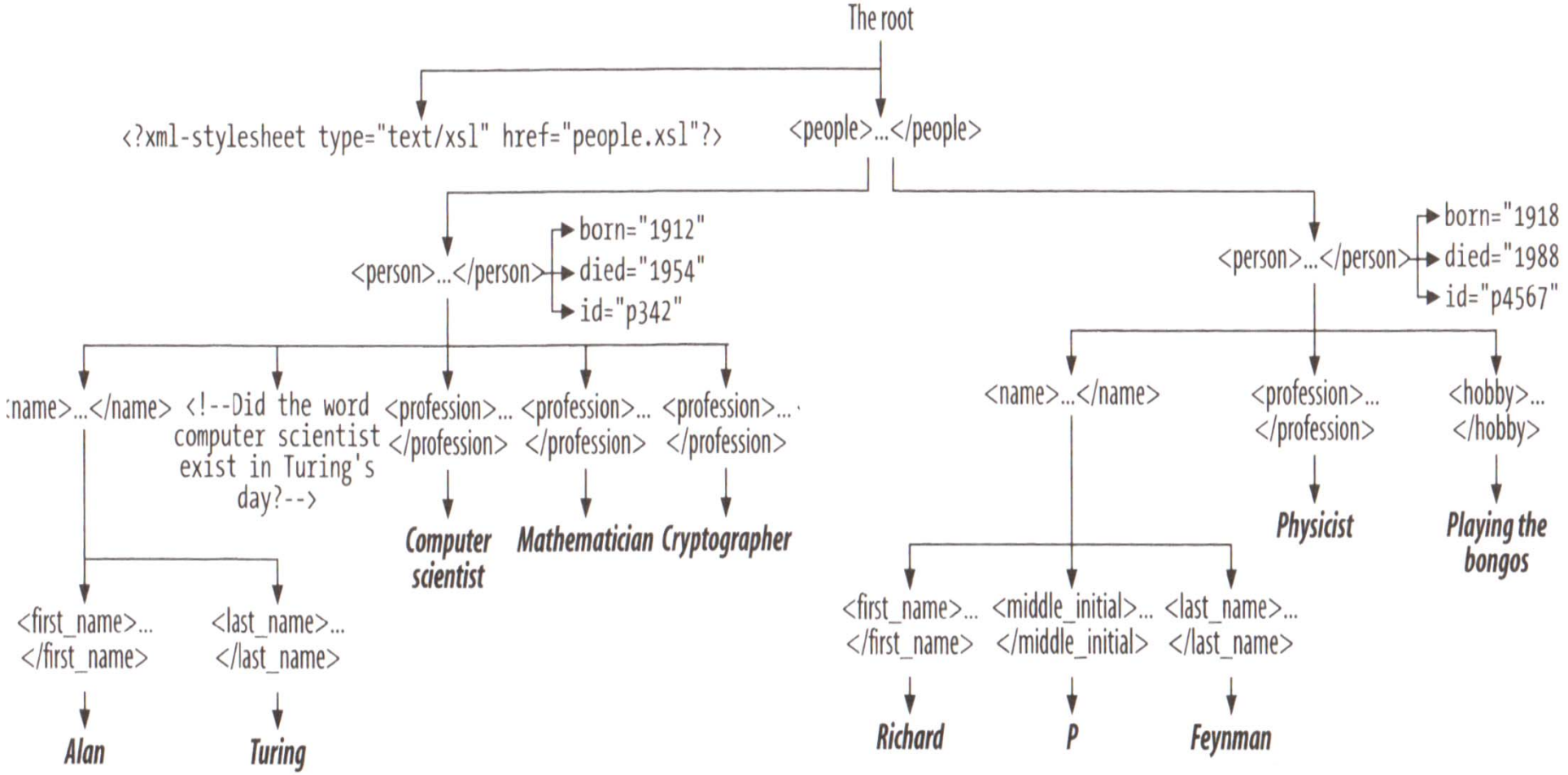
Sintaxis no abreviada

- Pasos:

  ***eje*::*test*[*predicado*][*predicado*]...**

  - eje: el tipo de seleccion que se quiere realizar
  - test: testea el tipo de nodo que se quiere seleccionar
  - predicado: una o mas condiciones a imponer sobre los nodos a seleccionar
- Trayectoria:
  - Secuencia de pasos separados por **/**

Ejemplo people.xml



Ejemplos people.xml

`/people/*`	`/child::people/child::*`
`/people/person/name/first_name/text()`	`/child::people/child::person/child::name/child::first_name/child::text()`
`person/@id`	`child::person/attribute::id`
`//name`	`/descendant-or-self::node()/child::name`
`person//name`	`child::person/descendant-or-self::node()/child::name`
`//middle_initial/../first_name`	`descendant-or-self::node()/child::middle_initial/parent::node()/child::first_name`
`.//name`	`self::node()/descendant-or-self::node()/child::name`
`person[@id="p342"]`	`child::person[attribute::id="p342"]`

Ejemplo people.xml

- Resultados de ejecutar en //person[2] las siguientes expresiones?
  - `self::*`
  - `child::*`
  - `child::hobby`
  - `following::*`
  - `attribute::born`
  - `preceding-sibling::node()`
  - `preceding::comment()`
  - `descendant::first_name`
  - `preceding-sibling::node()/child::profession`
  - `/child::people/child::person[attribute::born < 1950]/child::name[child::first_nam ="Alan"]`

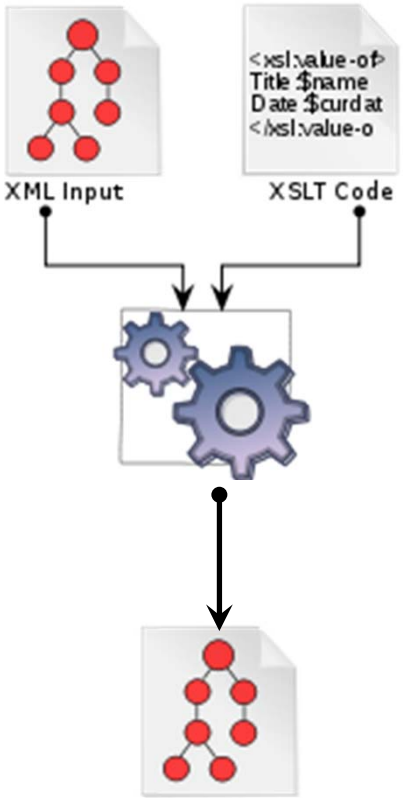
Links

- W3C Standard XPath 1.0
  - http://www.w3.org/TR/xpath
- W3C Standard XPath 2.0
  - http://www.w3.org/TR/xpath20/
- Varios buenos ejemplos (22 páginas):
  - http://www.zvon.org/xxl/XPathTutorial/Output/example1.html
- W3C school tutorial:
  - http://www.w3schools.com/Xpath/

XSL Transformations (XSLT)

Loreto Bravo

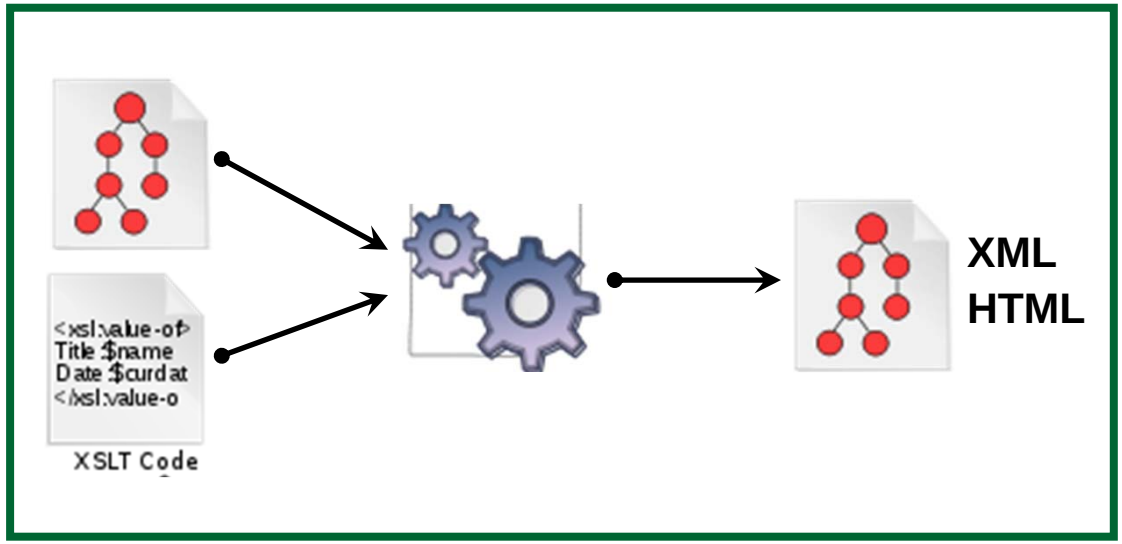
lbravo@gmail.com

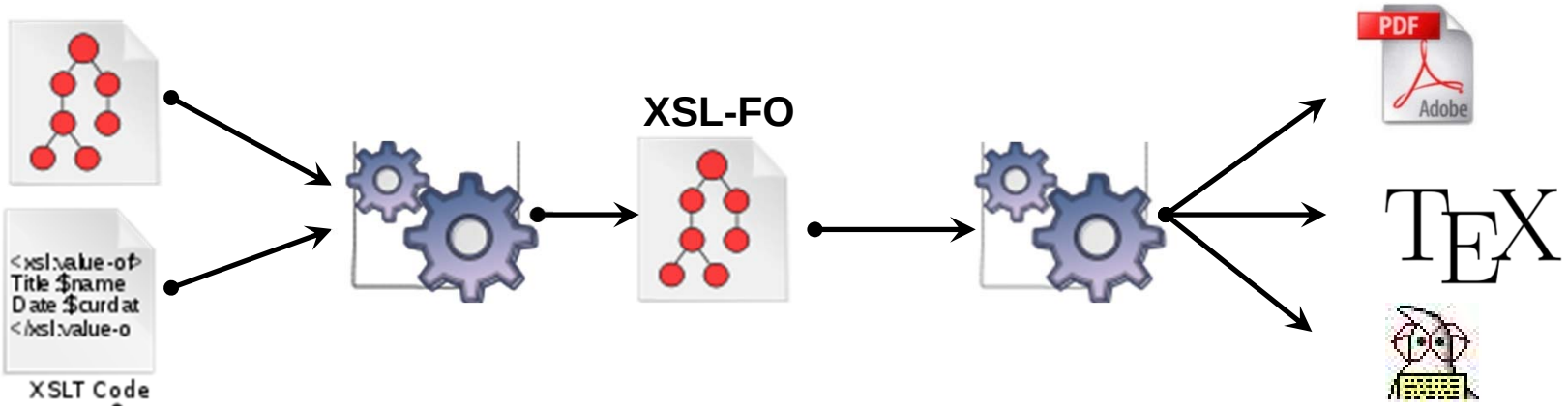


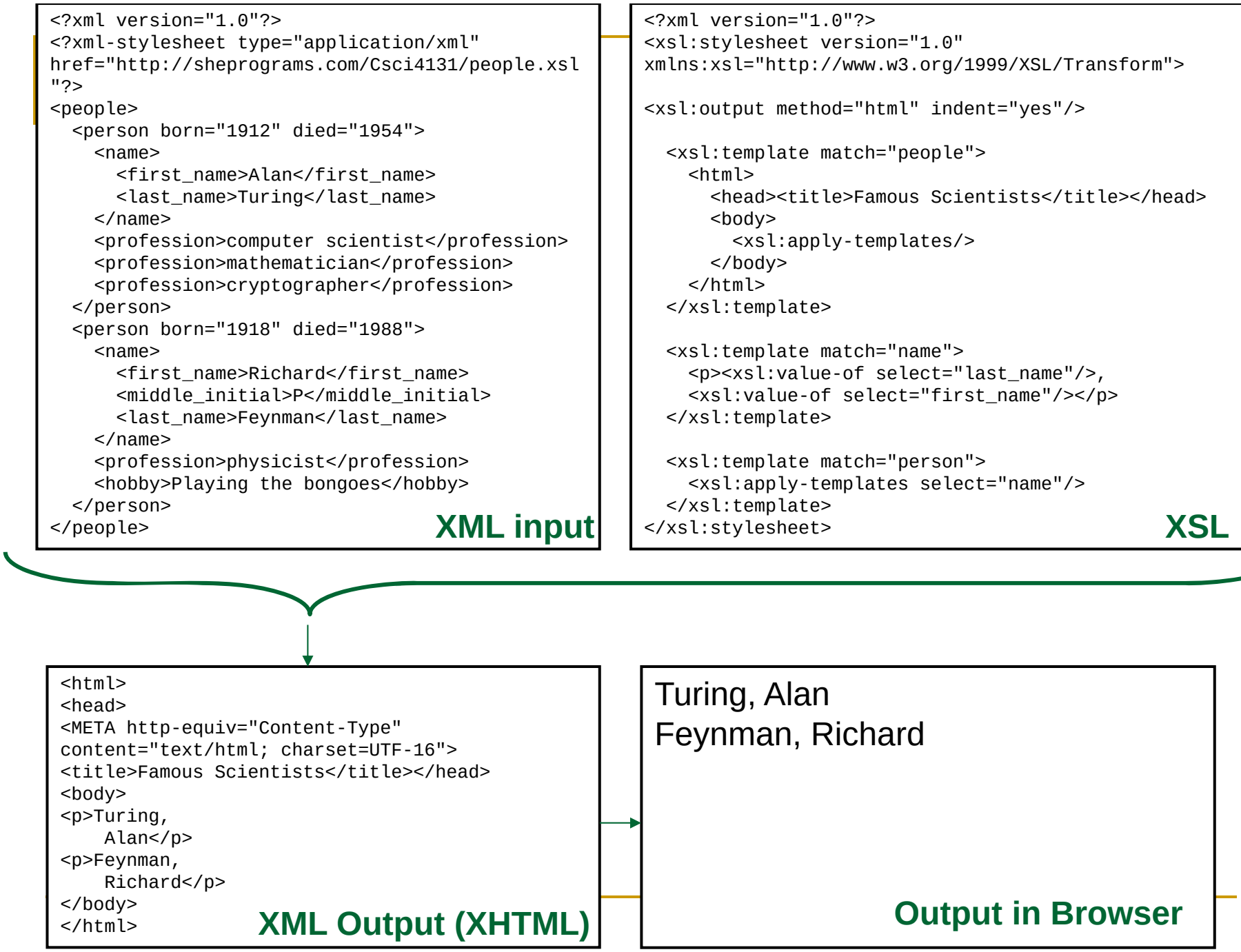
XSL

- Extensible Stylesheet Language (XSL):
  - XSL Transformations:
    - aplicación XML
    - especifica como transformar documentos XML en documentos XML
  - XSL Formatting Objects
    - aplicación XML
    - describe en forma precisa el diseño del texto en la página

XSL


<xsl:value-of
Title $name
Date $curdat
</xsl:value-o
XSLT Code
XML
HTML


<xsl:value-of
Title $name
Date $curdat
</xsl:value-o
XSLT Code
XSL-FO
PDF
Adobe
TEX


<?xml version="1.0"?>
<?xml-stylesheet type="application/xml"
href="http://sheprograms.com/Csci4131/people.xsl
"?>
<people>
<person born="1912" died="1954">
<name>
<first_name>Alan</first_name>
<last_name>Turing</last_name>
</name>
<profession>computer scientist</profession>
<profession>mathematician</profession>
<profession>cryptographer</profession>
</person>
<person born="1918" died="1988">
<name>
<first_name>Richard</first_name>
<middle_initial>P</middle_initial>
<last_name>Feynman</last_name>
</name>
<profession>physicist</profession>
<hobby>Playing the bongoes</hobby>
</person>
</people>
XML input
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" indent="yes"/>
<xsl:template match="people">
<html>
<head><title>Famous Scientists</title></head>
<body>
<xsl:apply-templates/>
</body>
</html>
</xsl:template>
<xsl:template match="name">
<xsl:value-of select="last_name"/>,
<xsl:value-of select="first_name"/>
</xsl:template>
<xsl:template match="person">
<xsl:apply-templates select="name"/>
</xsl:template>
</xsl:stylesheet>
XSL
<html>
<head>
<META http-equiv="Content-Type"
content="text/html; charset=UTF-16">
<title>Famous Scientists</title></head>
<body>
Turing,
Alan
Feynman,
Richard
</body>
</html>
XML Output (XHTML)
Turing, Alan
Feynman, Richard
Output in Browser

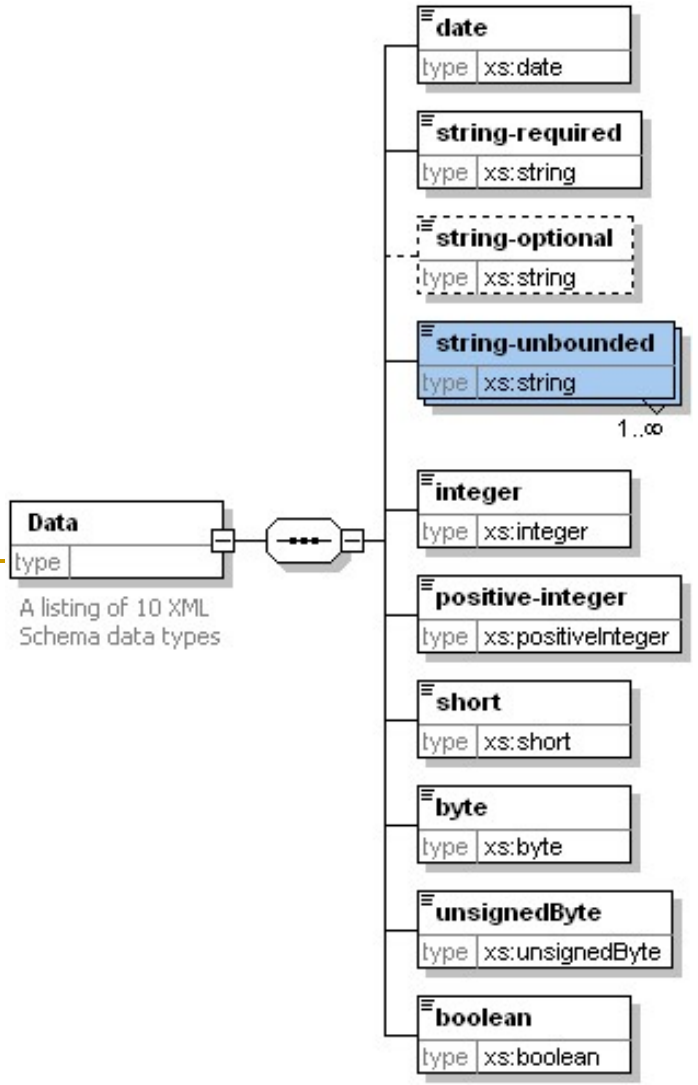
Procesadores XSLT

- Procesador
  - Toma un XSLT stylesheet y un documento XML
  - Genera un documento aplicando las reglas del XSLT sobre el documento
- Ejemplos
  - SAXON (linea de comando)
  - Apache XML project Xalan (linea de comando)
  - Cooktop
  - MSXML (en internet explorer)
  - Apache XML project Cocoon (en un servidor web)

Esquema XML
(XML Schema, XSD)

Loreto Bravo

lbravo@gmail.com



Esquema XML (XSD)

- W3C XML Schema Language
  - http://www.w3.org/XML/Schema
- Hay varios lenguages para definir esquemas para XML:
  - DTDs
  - XML Schemas
  - RELAX NG
- ¿Por qué los DTDs no son suficientes?
  - Los DTDs no están escritos en XML
    - Se necesita un parser especial
  - No se pueden definir restricciones
  - No hay variedad en cuanto a tipo de datos
    - Todo es texto!

Esquemas XML

- El elemento *schema* es el elemento raíz del documento en el que se define el esquema:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
...
</xs:schema>
```

- Contiene:
  - Tipos de datos
    - simple type
    - complex type: tipos que contienen elementos o atributos
  - Elementos
    - Estan asociados a algun tipo de dato

Ejemplos Elementos Simple

- Elemento que no contiene a otros elementos ni atributos
  - `<xs:element name="xxx" type="yyy"/>`
- Ejemplos:
  - `<xs:element name="apellido" type="xs:string"/>`
  - `<xs:element name="edad" type="xs:integer"/>`
  - `<xs:element name="fechaNacimiento" type="xs:date"/>`
  - `<xs:element name="clave" type="password"/>`
  - `<xs:element name="edadAlumno" type="edad"/>`

Ejemplos Simples con restricciones

```
<xs:simpleType name="edad">
    <xs:restriction base="xsd:integer">
          <xs:minInclusive value="0"/>
          <xs:maxInclusive value="130"/>
    </xs:restriction>
</xs:simpleType>


<xs:simpleType name="letra">
      <xs:restriction base="xsd:string">
            <xs:pattern value="[a-z]"/>
      </xs:restriction>
</xs:simpleType>
```

Ejemplos Simples con restricciones

```
<xs:simpleType name="iniciales">
        <xs:restriction base="xsd:string">
                <xs:pattern value="[a-zA-Z][a-zA-Z][a-zA-Z]"/>
        </xs:restriction>
</xs:simpleType>

<xs:simpleType name="eleccion">
        <xs:restriction base="xsd:string">
                <xs:pattern value="[xyz]"/>
        </xs:restriction>
</xs:simpleType>
```

Ejemplos Simples con restricciones

```
<xs:simpleType name="IDalumno">
     <xs:restriction base="xsd:integer">
     <xs:pattern value="[0-9][0-9][0-9][0-9][0-9]"/>
     </xs:restriction>
</xs:simpleType>


<xs:simpleType name="letras">
     <xs:restriction base="xsd:string">
            <xs:pattern value="([a-z])*"/>
     </xs:restriction>
</xs:simpleType>
```

Ejemplos Simples con restricciones

```
<xs:simpleType name="password">
        <xs:restriction base="xs:string">
                <xs:pattern value="[a-zA-Z0-9]{8}"/>
        </xs:restriction>
</xs:simpleType>
```

```
<xs:simpleType name="password">
        <xs:restriction base="xsd:string">
                <xs:length value="8"/>
        </xs:restriction>
</xs:simpleType>
```

Ejemplos Simples con restricciones

```
<xs:element name="Aeropuerto"
            type="CodigoAeropuerto">

<xs:simpleType name="CodigoAeropuerto">
  <xs:restriction base="xsd:string">
    <xs:enumeration value="SCL"/>
    <xs:enumeration value="CCP"/>
    <xs:enumeration value="YOW"/>
    <xs:enumeration value="YUL"/>
    ...
  </xs:restriction>
</xs:simpleType>
```

```
<xs:element name="Aeropuerto">
   <xs:simpleType>
     <xs:restriction base="xsd:string">
        <xs:enumeration value="SCL"/>
        <xs:enumeration value="CCP"/>
        <xs:enumeration value="YOW"/>
        <xs:enumeration value="YUL"/>
        ...
     </xs:restriction>
   </xs:simpleType>
</xs:element>
```

Tipo anonimo

Elemento Simple

- Elemento que no contiene a otros elementos ni atributos
  - `<xs:element name="xxx" type="yyy"/>`
- Ejemplos:
  - `<xs:element name="apellido" type="xs:string"/>`
  - `<xs:element name="edad" type="xs:integer"/>`
  - `<xs:element name="fechaNacimiento" type="xs:date"/>`
  - `<xs:element name="clave" type="password"/>`
  - `<xs:element name="edadAlumno" type="edad"/>`

Ejemplos

```
<xs:element name="Aeropuerto"
            type="CodigoAeropuerto">

<xs:simpleType name="CodigoAeropuerto">
  <xs:restriction base="xsd:string">
    <xs:enumeration value="SCL"/>
    <xs:enumeration value="CCP"/>
    <xs:enumeration value="YOW"/>
    <xs:enumeration value="YUL"/>
    ...
  </xs:restriction>
</xs:simpleType>
```

```
<xs:element name="Aeropuerto">
   <xs:simpleType>
     <xs:restriction base="xsd:string">
        <xs:enumeration value="SCL"/>
        <xs:enumeration value="CCP"/>
        <xs:enumeration value="YOW"/>
        <xs:enumeration value="YUL"/>
        ...
     </xs:restriction>
   </xs:simpleType>
</xs:element>
```

Tipo anonimo

Elementos complejos

- Elementos que contienen
  - atributos y/o
  - elementos
- Ejemplo:

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

Elementos Complejos

- Se pueden clasificar en cuatro tipos:
  1. Elementos que contienen solo elementos

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

  2. Elementos vacios

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

  3. Elementos que contienen solo texto

```
<food type="dessert">Ice cream</food>
```

  4. Elementos que contienen texto y elementos

```
<description>
It happened on <date lang="norwegian">03.03.99</date> ....
</description>
```

  - Todos ellos pueden contener atributos también!

Atributos

- Se declaran de forma similar a los elementos simples
- Los elementos que contiene atributos son complejos
- Un atributo se declara de la siguiente forma:

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

Ejemplo:

```
<xs:attribute name="lang" type="xs:string"/>
```

- Los atributos pueden tener valores por defecto y valores fijos:

```
<xs:attribute name="lang" type="xsd:string" default="EN"/>
```

- Si se quiere definir un atributo como obligatorio:

```
<xs:attribute name="lang" type="xsd:string" use="required"/>
```

  - Por defecto, los atributos son opcionales.

Elementos que contienen solo elementos

```
<xs:element name="employee">
  <xs:complexType>
    <xs:sequence>
    <xs:element name="firstname" type="xsd:string"/>
    <xs:element name="lastname" type="xsd:string"/>
    </xs:sequence>
    <xs:attribute name="RUT" type="xsd:string"/>
  </xs:complexType>
</xs:element>
```

- Elemento:

```
<employee RUT="15.035.161-K">
  <firstname>John</firstname>
  <lastname>Smith</lastname>
</employee>
```

Elemento Complejo vacío

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

- Elemento:

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

Elementos que contienen solo texto

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

- Ejemplo:
  `<shoesize country="UK">6</shoesize>`

Elementos que contienen texto y elementos

- Es necesario agregar atributo “mixed”

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

- Ejemplo:

```
<letter>Estimado cliente: <name>Juan Perez</name>. Su pedido número
<orderid>1032</orderid> se enviará el día <shipdate>2001-07-
13</shipdate>. </letter>
```

Elementos Complejos

- Además de **`<xs:sequence>`** existen varios otros constructores:
  - **`<xs:choice>`**
  - **`<xs:all>`**
  - **`maxOcurrs -- minOcurrs`** (para reemplazar y extender ? * +)
  - **`<xs:any>`**

<xs:all>

- Indica que los elementos aparecen exactamente una vez en cualquier orden

```
<xs:element name="persona">
   <xs:complexType>
     <xs:all>
            <xs:element name="nombre" type="xsd:string"/>
            <xs:element name="fechaNac" type="xsd:date"/>
            <xs:element name="telefono" type="xsd:integer"/>
     </xs:all>
   </xs:complexType>
</xs:element>
```

<xs:choice>

- Elige solo uno de los elementos contenidos

```
<xs:element name="Area">
   <xs:complexType>
        <xs:choice>
                <xs:element name="Provincia" type="xsd:string"/>
                <xs:element name="Estado" type="xsd:string"/>
        </xs:choice>
   </xs:complexType>
</xs:element>
```

maxOccurs y minOccurs

- Indican el número máximo y mínimo de veces que puede aparecer un elemento hijo de un elemento complejo
  - maxOccurs puede tomar el valor “unbounded”, que indica que no existe ningún límite

```
<xs:element name="person">
    <xs:complexType>
          <xs:sequence>
          <xs:element name="full_name" type="xsd:string"/>
          <xs:element name="child_name" type="xsd:string" minOccurs ="0"
                                                          maxOccurs="10"/>
          </xs:sequence>
    </xs:complexType>
</xs:element>
```

- Default value of both is 1

<xs:any>

- Permite incluir elementos no declarados inicialmente en el esquema

```
<xs:element name="person">
   <xs:complexType>
        <xs:sequence>
                <xs:element name="firstname" type="xsd:string"/>
                <xs:element name="lastname" type="xsd:string"/>
                <xs:any minOccurs="0"/>
        </xs:sequence>
   </xs:complexType>
</xs:element>

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

Ejemplo más complejo

```
<xs:element name="Items"><xs:complexType>
 <xs:sequence>
  <xs:element name="item" minOccurs="0" maxOccurs="unbounded">
   <xs:complexType>
    <xs:sequence>
     <xs:element name="productName" type="xsd:string"/>
     <xs:element name="quantity">
      <xs:simpleType>
       <xs:restriction base="xsd:positiveInteger">
        <xs:maxExclusive value="100"/>
       </xs:restriction>
      </xs:simpleType>
     </xs:element>
     <xs:element name="USPrice" type="xsd:decimal"/>
     <xs:element ref="comment" minOccurs="0"/>
     <xs:element name="shipDate" type="xsd:date" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="partNum" type="SKU" use="required"/>
   </xs:complexType>
  </xs:element>
 </xs:sequence>
</xs:complexType> </xsd:element>
```

Tipos

- Es posible tambien definir tipos para utilizar en otras partes del esquema

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

<xs:element name="employee" type="personinfo"/>
<xs:element name="student" type="personinfo"/>
```

Ejercicio

Escribir el siguiente DTD utilizando XML Schema:

```
<!ELEMENT paper (title,author*,year, (journal|conference))>
```

Criticas a XSD

- Las especificaciones son muy largas!
  - Dificiles de enteder e implementar
- Validar un esquema es mas costoso y complejo especialmente para grandes volumenes de datos

DTD

```
<!ELEMENT order (item)+>
<!ELEMENT item (name,price)>
<!ATTLIST item code NMTOKEN #REQUIRED>
<!ELEMENT name (#PCDATA)>
<!ELEMENT price (#PCDATA)>
<!ATTLIST price currency NMTOKEN 'USD'>
```

XML Schema

```
<xsd:schema xmlns:xsd='http://www.w3.org/2001/XMLSchema'>

 <xsd:element name='order' type='Order'/>
 <xsd:element name='item'  type='Item'/>
 <xsd:element name='name'  type='Name'/>
 <xsd:element name='price' type='Price'/>

 <!-- <!ELEMENT order (item)+> -->
 <xsd:complexType name='Order'>
  <xsd:sequence>
   <xsd:element ref='item' minOccurs='1' maxOccurs='unbounded'/>
  </xsd:sequence>
 </xsd:complexType>

 <!-- <!ELEMENT item (name,price)> -->
 <xsd:complexType name='Item'>
  <xsd:sequence>
   <xsd:element ref='name'/>
   <xsd:element ref='price'/>
  </xsd:sequence>
  <!-- <!ATTLIST item code NMTOKEN #REQUIRED> -->
  <xsd:attribute name='code'>
   <xsd:simpleType>
    <xsd:restriction base='xsd:string'>
     <xsd:pattern value='[A-Z]{2}\d{3}'/>
    </xsd:restriction>
   </xsd:simpleType>
  </xsd:attribute>
 </xsd:complexType>

 <!-- <!ELEMENT name (#PCDATA)> -->
 <xsd:simpleType name='Name'>
  <xsd:restriction base='xsd:string'/>
 </xsd:simpleType>

 <!-- <!ELEMENT price (#PCDATA)> -->
 <xsd:complexType name='Price'>
  <xsd:simpleContent>
   <xsd:extension base='NonNegativeDouble'>
    <!-- <!ATTLIST price currency NMTOKEN 'USD'> -->
    <xsd:attribute name='currency' default='USD'>
     <xsd:simpleType>
      <xsd:restriction base='xsd:string'>
       <xsd:pattern value='[A-Z]{3}'/>
      </xsd:restriction>
     </xsd:simpleType>
    </xsd:attribute>
   </xsd:extension>
  </xsd:simpleContent>
 </xsd:complexType>

 <xsd:simpleType name='NonNegativeDouble'>
  <xsd:restriction base='xsd:double'>
   <xsd:minInclusive value='0.00'/>
  </xsd:restriction>
 </xsd:simpleType>

</xsd:schema>
```

Otra tecnología XML

Otras tecnologías XML

- XPointer:
  - Lenguaje de punteros XML
  - Estándar del World Wide Web Consortium (W3C)
  - Para identificar, de forma única, fragmentos de un documento XML para realizar vínculos
- XLink:
  - Lenguaje de vínculos XML
  - Recomendación de W3C
  - Permite crear elementos de XML que describen relaciones cruzadas entre documentos, imágenes y archivos de Internet u otras redes
  - Usa XPointer

XLink and XPointer Example

```
?xml version="1.0" encoding="ISO-8859-1"?>

<mydogs xmlns:xlink="http://www.w3.org/1999/xlink">

<mydog xlink:type="simple"
  xlink:href="http://dog.com/dogbreeds.xml#Rottweiler">
  <description xlink:type="simple"
  xlink:href="http://myweb.com/mydogs/anton.gif">
  Anton is my favorite dog. He has won a lot of.....
  </description>
</mydog>

<mydog xlink:type="simple"
  xlink:href="http://dog.com/dogbreeds.xml#FCRetriever">
  <description xlink:type="simple"
  xlink:href="http://myweb.com/mydogs/pluto.gif">
  Pluto is the sweetest dog on earth......
  </description>
</mydog>

</mydogs>
```

Otras tecnologías XML

- XQuery:
  - lenguaje de consulta diseñado para consultar colecciones de datos XML
  - Semánticamente similar a SQL, pero incluye algunas capacidades de programación (turing complete)
  - Utiliza XPath

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

Otras tecnologías XML

- SAX y DOM
  - Parseadores de XML
  - Estan definidos independiente del lenguaje
    - Existen implementaciones disponibles en varios lenguajes: Java, C++, Perl, Python,....
  - SAX
    - Define interfaz dirigido por eventos (*event-driven*) para el procesamiento de un documento XML
    - http://www.megginson.com/SAX
  - DOM
    - Provee una representación de un documento XML en forma de un árbol
    - http://www.w3.org/DOM