# Ontologías y Web Semántica

## XML avanzado

Pablo R. Fillottrani

Depto. Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur

marzo 2019



## XML avanzado





## URI

- un URI (Uniform Resource Identifier) es un elemento de los protocolos de Internet consistente en una secuencia de caracteres, y que representa un recurso
- puede ser entendido como un nombre, como una dirección, o ambos
- está compuesto de un nombre de esquema, como `http`, `ftp`, `urn`, etc, seguido de `:`, y de una parte específica del esquema. Ejemplos:

  `http://www.google.com`
  `urn:ISBN:0-111-1111-1`
- la sintaxis y la semántica de la parte específica del esquema es determinada por el esquema



## URL y URN

- cuando un URI se interpreta como una dirección, entonces también se conoce como URL (locator)
- en este caso, el esquema indica cómo es posible obtener el recurso, y la parte dependiente indica un parámetro para este método
- un URN (name) es un URI que identifica al recurso, sin necesidad de implicar ubicación o dereferencia

## IRI

- sólo un pequeño subconjunto de caracteres ASCII es permitido en la cadena que identifica un URI
- como el objetivo de un URI es también que sirva a los humanos identificar el recurso, es conveniente que se puedan definir URIs con caracteres internacionales
- se define entonces un IRI (international resource identifier) como una secuencia de caracteres Unicode, juntamente con un mapeo IRI a URI



## Referencias a URIs

- una referencia a un URI es otra secuencia de caracteres que representa un URI, y a su vez también, el recurso correspondiente
- una referencia puede tomar la forma de un URI completo, o de solamente la parte dependiente del esquema, o aun una subcadena de la misma
- puede también contener un identificador de fragmento precedido por un #
- para obtener el URI de una referencia a un URI, esta se convierte a la forma “absoluta” mediante la composición con un URI base, dependiente del contexto



## Namespaces: objetivos

- la introducción de espacios de nombres en XML tiene los siguientes objetivos
  - distinguir entre elementos y atributos provenientes de distintos vocabularios que casualmente tengan el mismo nombre
  - agrupar elementos y atributos de la misma aplicación XML de manera que sean fácilmente reconocidos
- si bien el primer propósito es más fácil de entender, en la práctica el segundo es el más importante



- los espacios de nombres se implementan separando el nombre de un elemento/atributo en: un prefijo, el caracter `:`, y el nombre local
- cada prefijo es mapeado a un URI, mediante un atributo `xmlns:prefijo` definido en algún elemento que contenga a todas las referencias
- se puede también asignar URI por default a los nombres sin prefijos
- elementos y atributos que estan asociados al mismo URI se consideran en el mismo espacio de nombres

- dado que URIs contienen caracteres que no se permiten en los nombres XML, es necesario la definición de prefijos que los abrevien
- los URIs particionan el conjunto de nombres de aributos y elementos
- es necesario destacar que los prefijos son sólo abreviaturas
- nombres con prefijo contienen exactamente un caracter `:`



- todo lo que aparece después del `:` se denomina parte local
- el nombre completo, prefijo, dos puntos, y parte local, se llama nombre calificado
- para evitar ambigüedades en el parsing, ni el prefijo ni la parte local pueden contener `:`



- la asociación entre `prefijo` y un URI se hace mediante un atributo `xmlns:prefijo` en un elemento
- esta asociación es válida en el contenido de ese elemento, y en los de todos sus descendientes
- ejemplo:

```
<pref:x1 xlmns:pref="http://www.miURI.org">
      <pref:vacio/>
      <sinEspNombres/>
</pref:x1>
```



- si es necesario, un mismo elemento puede contener varias declaraciones de prefijos
- los elementos sin prefijos no pertenecen a ningún espacio de nombres
- puede definirse en un dado elemento un valor por default para los prefijos, con el atributo `xmlns`
- esta declaración es válida para todos los nombres de elementos sin prefijos en el contenido del elemento de la declaración
- no afecta los nombres de atributos

- es posible redefinir los prefijos con distintos URIs
- no existe asociación entre los espacios de nombres nombrados con URIs que son URLs, y el contenido de la página correspondiente



## Parser y los espacios de nombres

- los espacios de nombres son posteriores a XML 1.0
- sin embargo, la sintaxis es compatible. Un parser que no reconoce espacios de nombres no debería tener problemas en reconocer un documento que los usa.
- un parser que reconoce espacios de nombres debe sólo realizar dos controles adicionales a las reglas de documento bien formado:
  1. que ningún nombre contenga dos `:`
  2. que todos los prefijos esten mapeados a URIs
- varios parsers permiten al usuario seleccionar o no el modo de control de espacios de nombres



## Entidades parámetro y espacios de nombres

- los espacios de nombres son totalmente ortogonales con los DTDs
- los nombres con elementos con prefijo por default deben ser declarados en los DTD sin prefijo
- es posible usar entidades parámetros para definir texto de reemplazo para prefijos, de forma de construir documentos independientes del prefijo



## Objetivos

- XPath es un lenguaje no-XML para identificar partes de un documento XML
- XPath es un lenguaje de expresiones, cuyos resultados pueden ser conjuntos de nodos, valores enteros, cadenas o booleanos
- se puede ubicar nodos por posición absoluta y relativa, contenido, tipo, y otros varios criterios
- XPath es usado en otros lenguajes como XSLT, XML Schemas, XLink y XPointer

## Modelo del documento

- para XPath, un documento XML está formado por un árbol de nodos
- existe exactamente un nodo raíz, que contiene a todos los demás nodos en el documento
- el nodo raíz no corresponde con el elemento raíz del documento XML
- XPath es un lenguaje de expresiones para seleccionar algunos nodos de este árbol



Tipos de nodos:
- raíz
- elemento
- texto
- atributo
- comentario
- instrucciones de procesamiento (pi's)
- espacios de nombres

- se puede observar que no existen nodos CDATA, entidades, o DTD. Esto se debe a que XPath trabaja sobre el documento XML ya normalizado



- el nodo raíz es distinto al nodo elemento documento
- los atributos `xlmns:prefijo` y `xlmns` no generan nodos atributos, sino nodos espacios de nombres
- las expresiones XPath que evaluan en conjuntos de nodos se denominan caminos de evaluación



## Pasos de ubicación

- un camino está formado por varios pasos
- cada paso de ubicación identifica un conjunto de nodos
- los pasos pueden ser
  - pasos simples que indican la posición relativa de los nodos con respecto a un nodo contexto determinado por la aplicación
  - pasos filtrados donde además se especifican propiedades que los nodos seleccionados deben cumplir

- son pasos simples:
  - la selección del nodo raíz `/` (no depende del nodo contexto)
  - la selección de los elementos hijos de acuerdo a su nombre `nombre`
  - la selección de atributos de acuerdo a su nombre `@nombre`
  - la selección de comentarios, partes de texto contiguas, o instrucciones de procesamiento mediante `comment()`, `text()` y `processing_instructions()`
  - la selección de todos los nodos elementos hijos con `*`
  - la selección de todos los atributos con `@*`
  - la selección de todos los nodos hijos con `node()`



- los pasos filtrados agregan a continuación de un paso simple una condición encerrada entre [ y ]
- ejemplo: `nombre[apellido="perez" or @id="123"]`
- la condición puede contener comparaciones entre contenido de elementos, atributos y cadenas textuales, combinadas con los operados booleanos tradicionales



- si el resultado en lugar de ser booleano es entero, entonces indica posición en el nodo contexto
- ejemplo: `nombre[last()-3]`
- si el resultado es una cadena de texto, entonces se seleccionan todos los nodos para los cuales el resultado es distinto de la cadena vacía
- ejemplo: `nombre[inicial]`



## Caminos de ubicación

- cada paso de ubicación puede ser compuesto con otros para formar los caminos
- son operadores de composición de pasos:
  - `/` para indicar los elementos descendientes directos
  - `//` para indicar todos los elementos descendientes
  - `..` para indicar el elemento padre
  - `.` para indicar el nodo contexto
- ejemplo: `nombre//inicial/../apellido`

- los caminos de ubicación también se pueden combinar con el operador | que significa opción
- ejemplo: `nombre[@apellido] |profesion`



## Notación no abreviada

- las expresiones XPath vistas se denominan expresiones abreviadas
- en general, toda expresión XPath tiene dos componentes: el componente eje y el componente nodo
- en la notación abreviada estos componentes están mezclados
- por ejemplo la expresión `nombre` tiene componente eje `child` y componente nodo `nombre`
- por ejemplo la expresión `@id` tiene componente eje `attribute` y componente nodo `id`



- en la notación no abreviada, las dos partes componentes son separadas por un `::`
- ejemplo: `child::nombre` y `attribute::id` son equivalentes a las anteriores



- existen diversos ejes que no tienen notación abreviada correspondiente. Por ejemplo:
  - `ancestor` todos los elementos ancestros del nodo contexto
  - `following_sibling` todos los nodos que siguen al nodo contexto en el contenido del nodo padre, y son descendientes del mismo padre
  - `following` todos los nodos que siguen al nodo contexto en el documento
  - `preceeding` y `preceeding_sibling` son definidos en forma similar

## Expresiones generales

- XPath no se limita a que el resultado de las expresiones sea un conjunto de nodos
- también pueden ser números, booleanos o cadenas, con el objetivo de permitir a la aplicación disponer de un limitado lenguaje de procesamiento
- ejemplo: `56 * 87`, `3.14 + 5` o `position()=last()`
- existen varias funciones predefinidas permitidas en estas expresiones



## Objetivos

- aunque DTDs pueden especificar restricciones estructurales sobre los documentos, muchas aplicaciones necesitan un método de validación más poderoso y expresivo
- la recomendación W3C XML Schema cumple este objetivo, especificando relaciones complejas entre elementos y atributos de un documento XML
- un esquema XML es un documento XML que contiene la descripción formal de lo que constituye un documento XML válido
- W3C XML Schema Language es sólo otro de los distintos lenguajes para esquemas XML, como RELAX y Schematron



## Comparación con DTD

- los DTDs pueden ser usados para validar:
  - anidado de elementos
  - atributos permitidos
  - tipos y valores por default de los atributos
- XML Schema Language puede ser usado para definir:
  - tipos de datos simples y complejos
  - derivación de nuevos tipos y herencia
  - restriciones en la ocurrencia de elementos
  - elementos y atributos dependientes de espacios de nombres



- DTDs no están escritos en XML; XML Schemas sí
- DTDs sólo permiten un limitado conjunto de tipos simples; XML Schemas extienden esos tipos simples y proveen constructores para tipos complejos
- DTDs es suficiente para documentos narrativos, orientados al documento; XML Schemas está orientado a aplicaciones orientadas a datos, con estructura tipo registro
- XML Schemas puede expresar restricciones más finas sobre número y secuencia de elementos
- DTDs permite definir entidades generales, cosa que no es posible con XML Schemas
- en síntesis, ambos pueden coexistir

# Asociación documento-esquema

- un documento XML descripto por un esquema se denomina documento instancia
- si el documento satisface todas las restricciones especificadas en el esquema, entonces se lo considera esquema-válido
- el esquema asociado a un documento instancia es especificado en una de las siguientes formas:
  - con el atributo `xsi:schemaLocation` en un elemento, conteniendo el URI del archivo con el esquema que valida el elemento
  - con el atributo `xsi:noNamespaceSchemaLocation` para validar elementos que no están en ningún espacio de nombres
  - una indicación explícita del usuario al parser, externa al documento XML a validar



- ejemplo:

```
<elemento
  xlmns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="schema.dtd">
        ...
</elemento>
```

- se puede observar que XML Schemas depende fuertemente de los espacios de nombres
- para validar un documento con XML Schemas, es necesario asociar el URI correcto, no el prefijo `xsi`



- el documento esquema consiste de una única declaración de elementos `xs:schema` conteniendo declaraciones para todos los elementos y atributos que pueden aparecer en una instancia válida
- el prefijo `xs:` es generalmente asociado al URI
  `http://www.w3.org/2001/XMLSchema`
- los elementos `xs:element` declarados como hijos del elementos `xs:schema` se consideran elementos globales
- cualquier elemento instancia de una declaración global puede ser el elemento raíz del documento XML



- cada elemento `xs:element` debe tener atributos `name` y `type` con su nombre y tipo
- el tipo puede ser un nombre de un tipo predefinido, o un nombre de un tipo definido por el esquema en otra parte
- si el atributo `type` no está presente, entonces se entiende que el contendio de ese elemento `xs:element` es la definición del tipo

# Anotaciones

- las anotaciones constituyen documentación extra a la definiciíon propia del esquema
- pueden definirse asociadas a elementos con los nombres `xs:annotation`
- estos a su vez pueden contener elementos `xs:documentation` y `xs:appinfo` para información destinada a humanos y aplicaciones respectivamente



# Declaración de elementos y atributos

- el marcado `xs:element` se usa para indicar un elemento, en el esquema, si se trata de un elemento global, o en el elemento que lo contiene
- cada elemento puede ser de contenido
  - vacío
  - contenido simple (con `xs:simpleType`)
  - contenido complejo (con `xs:complexType`)
- si el contenido es de tipo simple, entonces puede ser tipo de un atributo o de un elemento
- si el contenido es de vacío o de tipo complejo, entonces sólo puede tratarse de un tipo de elemento



- el tipo de contenido puede estar definido explícitamente con un tipo anónimo, o mediante un nombre global
- ejemplo:

```
<xs:elemento  name="name">
    <xs:simpleType><xs:restriction base="xs:string">
    </xs:simpleType>
</elemento>
<xs:elemento name="name" type="xs:string">
</elemento>
```



- los atributos se declaran con el marcado `xs:attribute`, y siempre se declaran de un tipo simple
- deben tener siempre un nombre, definido con el atributo `name` y un tipo, nombrado con el atributo `type` o anónimo con la definición explícita
- los atributos pueden ser declarados globalmente, o en forma local
- se pueden agrupar atributos con `xs:attributeGroup`, y referirse a ellos en forma grupal

- ejemplos:

```
<xs:attributeGroup name="address">
  <xs:attribute name="street" type="xs:string"/>
  <xs:attribute name="number" type="xs:integer"/>
  <xs:attribute name="city" type="xs:string"/>
  <xs:attribute name="zipCode" type="xs:string"/>
</xs:attributeGroup>
<xs:element name="person">
  <xs:attribute name="language" type="xs:language"/>
  <xs:attributeGroup ref="address"/>
  ...
</xs:element>
```



## Contenido vacío

- si se declara un elemento de tipo vacío, entonces este elemento sólo contribuye al documento con sus atributos, o con su posición en relación con otros elementos
- la sintaxis es declarar un tipo complejo sin elementos
- ejemplo:

```
<xs:element name="phone">
 <xs:complexType>
   <xs:attribute name="number" type="xs:integer"/>
 </xs:complexType>
</xs:element>
```



## Contenido de tipo simple

- si es de contenido simple con uno de los varios tipos simples especificados: `xs:string`, `xs:anyURI`, `xs:boolean`, `xs:byte`, `xs:dateTime`, `xs:integer`, etc
- también se permiten los tipos predefinidos en DTD: `ID`, `IDREF`, `ENTITY`, `NMTOKEN`, etc
- es posible definir nuevos tipos simples que restrinjan (`xs:restriction`) los valores de un tipo simple predefinido (permitiendo enumerados, rangos de enteros, restricciones en la longitud de cadenas, etc)
- listas y uniones de tipos simples



- para tipos simples se pueden definir distintas facetas que limitan los valores que pueden asignarse
- las posible facetas son:
  - `xs:length`, `xs:minlength`, `xs:maxlength`
  - `xs:pattern`
  - `xs:enumeration`
  - `xs:whiteSpace`
  - `xs:maxInclusive`, `xs:maxExclusive`
  - `xs:minInclusive`, `xs:maxExclusive`
  - `xs:totalDigits`
  - `xs:fractionDigits`

- ejemplos:

```
<xs:simpleType name=''location''>
 <xs:restriction base=''xs:string''>
   <xs:enumeration value=''work''/>
   <xs:enumeration value=''home''/>
   <xs:enumeration value=''mobile''/>
 </xs:restriction>
</xs:simpleType>
<xs:simpleType name=''number''>
 <xs:restriction base=''xs:string''>
  <xs:pattern value=''+\d\d-\d\d\d-\d\d\d\d\d\d\d''/>
 </xs:restriction>
</xs:simpleType>
```



- así como DTDs pueden definir tipos `IDREFS`, `ENTITIES`, etc, XML Schemas también permite definir tipos simples compuestos de valores de tipos simples
- el constructor `xs:list` define un tipo simple como una lista de valores de un determinado tipo base, también simple, separados por espacios
- el constructor `xs:union` define un tipo simple como una alternativa entre valores de una serie de tipos simples



- ejemplos:

```
<xs:simpleType name="nameList">
 <xs:list itemType="xs:string">
</xs:simpleType>
<xs:attribute name="location">
 <xs:simpleType>
  <xs:union memberType="location xs:NMTOKEN">
 </xs:simpleType>
</xs:attribute>
```



## Contenido Complejo

- si los elementos son de contenido complejo entonces se deben definir nuevos elementos `xs:complexType` con la especificación del tipo
- si el elemento `xs:complexType` tiene el atributo `mixed="true"`, entonces todas las instancias de ese tipo deben contener tanto texto como elementos
- los tipos complejos pueden contener secuencias o alternativas de elementos en distinto orden, declaradas con el marcado `xs:sequence`, `xs:choice`, `xs:all`
- todos los elementos de tipo complejo pueden tener attributos `xs:minOccurs` y `xs:maxOccurs` que limitan la repetición de los elementos

- el lenguaje también permite derivar nuevos tipos complejos mediante la extensión y la restricción de tipos existentes
- el tipo `xs:anyType` es un tipo predefinido que permite una secuencia sin restricciones de marcado y texto
- los marcados `xs:simpleContent` y `xs:complexContent` limitan el contenido de los tipos a datos simples o complejos
- estos tipos derivados pueden ser usados en cualquier lugar especificado con el tipo original



- ejemplos:

```
<xs:element name="phone" minOccurs="0">
 <xs:complexType>
  <xs:complexContent>
   <xs:restriction base="xs:any"/>
    <xs:attribute name="number" type="xs:string"/>
   </xs:restriction>
  </xs:complexContent>
 </xs:complexType>
</xs:element>
```



- es posible definir que un conjunto de elementos tienen valor único (determina una clave) mediante los marcados `xs:unique` y `xs:key`
- estos elementos deben contener como hijos elementos `xs:selector` y `xs:field`, que a su vez contienen atributos `xpath` con valor expresiones XPath. El primero indica el ámbito de la restricción; el segundo cuál es el elemento restringido
- también se pueden asociar nombres a grupos de elementos con `xs:group`, y referenciar a todos esos elementos en diversas ocasiones sin necesidad de constituir un nuevo tipo



- XQuery es un lenguaje de consulta para documentos XML, el análogo a SQL para bases de datos relacionales
- permite ubicar y extraer elementos y atributos de los documentos XML
- es una recomendación W3C de enero 2007

- XQuery 1.0 y XPath 2.0 comparten el mismo modelo de datos, y soportan las mismas funciones y operadores
- XQuery permite además
  - extraer información para ser usada en un servicio web
  - generar reportes resúmenes
  - transformar datos XML en XHTML



```
<?xml version="1.0" encoding="ISO-8859-1"?>
<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 usa expresiones de caminos de ubicación para navegar a través de los elementos de un docuemento XML
- ejemplo:
  `doc("books.xml")/bookstore/book/title` puede devolver los siguientes datos

```
<title lang="en">Everyday Italian</title>
<title lang="en">Harry Potter</title>
<title lang="en">XQuery Kick Start</title>
<title lang="en">Learning XML</title>
```



- el mismo resultado puede obtenerse con la expresión FLWOR

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

- FLWOR significa "for, let, where, order by, return", y son expresiones similares a SQL presentes en XQuery
- es posible ordenar el resultado agregando `order by $x/title`

- los resultados pueden ordenarse usando la clausula `order by`
- también es posible generar código HTML, por ejemplo, agregando literales a la orden FLWOR
- ejemplo:

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



## Tipos de aplicaciones

- la mayoría de los lenguajes de programación proveen soporte para XML, como por ejemplo C++, Perl y Java.
- este soporte de programación puede ser
  1. basado en eventos un programa responde a los eventos generados por un parser de XML a medida que procesa el documento XML. Ejemplo: SAX
  2. basado en árboles: un programa atraviesa y modifica localmente un modelo del documento, generado por el parser. Ejemplo: DOM
  3. basado en reglas: un programa ejecuta reglas de transformación recursivas causando modificaciones globales al árbol del documento (XSLT)
- generalmente el soporte es en forma de librerías



## Interfaces basada en eventos

- es un enfoque más orientado a cómo trabaja un parser, de bajo nivel de abstracción
- la librería va leyendo el documento, y genera eventos como por ejemplo
  - `start document` y `end document`
  - `start element` y `end element`
  - `start attribute` y `end attribute`
  - `characters` para datos textuales



- la aplicación debe negociar trámite un manejador de eventos
- el documento original permanece sin cambios
- muchas veces es necesario mantener una estructura auxiliar, para futuros procesamientos
- ejemplo: SAX para Java, Xerces para Java del proyecto apache

## Interfaces basadas en modelo: DOM

- el parser construye explícitamente un árbol de objetos que contiene toda la información del documento XML
- se puede consultar y modificar este árbol mediante operaciones en la librería
- posteriormente el árbol modificado puede sobreescribir el documento original, implicando modificaciones en el mismo
- ejemplo: JDOM para Java, Xerces para Java del proyecto apache



## Comparación

- es preferible usar DOM cuando
  - se necesita modificar estructuralmente el documento XML
  - se comparte el documento con otras aplicaciones
  - se necesita acceso aleatorio a distintas partes del documento
- es preferible usar SAX cuando
  - se necesita alta performance
  - no se tiene mucha memoria, o el documento a procesar es muy grande
  - se tiene un flujo continuo de datos



## Procesamiento basado en reglas: XSLT

- una hoja de estilo XSLT es un documento XML que especifica cómo transformar un documento XML de entrada en otro documento XML de salida
- es decir, hace un transformación en el documento
- el control de esta transformación se hace mediante la especificación de reglas



- las reglas se definen con un elemento `xsl:template` que tiene un atributo `match` que especifica cuándo se aplica la regla, y el contenido del elemento es lo que aparece en el documento salida
- el atributo `match` usa expresiones XPath para determinar a cuáles nodos aplicar las reglas
- XSLT no es una librería, sino una aplicación independiente
- ejemplo Xalan del proyecto apache, escrito en Java

# Ejercicio práctico

- Realizar una comparación entre XML, Jason, YAML y LJS
- Informe escrito y presentación oral (17/9?)