TP1: xslt prog

Detalles del trabajo

Objetivos

- Analizar el problema planteado correctamente.
- Diseñar una solución utilizando correctamente los conceptos y lineamientos de **programación funcional**.
- Seguir las buenas prácticas de programación.

Indicaciones del trabajo

Grupos de trabajo

El trabajo se desarrollará de forma grupal, en grupos de 4 alumnos. El trabajo práctico debe realizarse utilizando Git y Github como herramienta de colaboración.
Todos los alumnos integrantes del grupo **deben** participar activamente del desarrollo del trabajo de forma equitativa. Mientras la participación sea equitativa, queda a criterio de cada grupo la forma de trabajo y organización.

Tecnología

El trabajo debe realizarse en *Scala 3*, utilizando cualquier versión estable de esta versión.

Tiempo de trabajo y entregas parciales

Se contarán con aproximadamente 4 semanas completas para llevar a cabo el desarrollo del TP. El trabajo no cuenta con entregas parciales: transcurrido el tiempo de trabajo, debe realizarse la **entrega**.

Modalidad de entrega

Para realizar la entrega se debe:

- Github:
  - El repositorio tiene que ser privado y debe estar su corrector invitado al mismo.
  - Para realizar la entrega se debe crear un branch llamado `tp-1`, con el código correspondiente a la entrega. No se debe modificar más luego de la fecha de entrega.
  - Se debe contar con un `README.md` que explique cómo correr el programa “desde cero”, cómo instalar las dependencias y otras explicaciones pertinentes. El código debe poder correrse con un comando que compile el

código y lo ejecute, **no es válido subir un ejecutable y que el comando simplemente lo corra**.

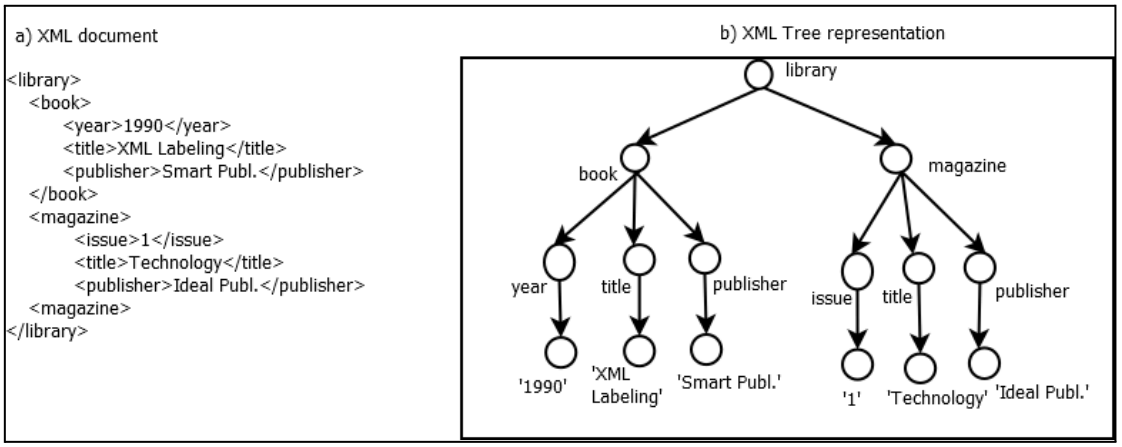
- Mail: Mandar un mail a algoritmos3.fiuba@gmail.com indicando número de grupo y sus integrantes (nombre, apellido y padrón), aclarando cuál es la branch y el repositorio de la entrega. El mail debe llevar adjunto el informe en formato PDF y asunto debe `TP1 - <nombre del grupo>`.

Fecha de entrega

La entrega definitiva, que debe alinearse a lo especificado en la Modalidad de entrega, debe realizarse con anterioridad a las 23:59hs del viernes 25/04/2025.

Enunciado: Transformaciones de XML

Dominio

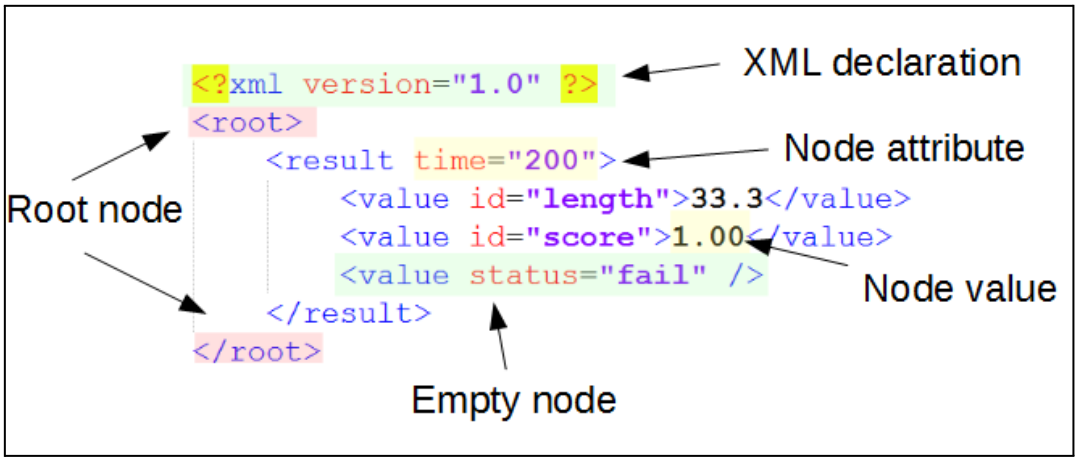


Ejemplo de un documento XML y su representación de árbol. Ver también acá.

¿Qué es XML?

El formato de texto **XML** (Extensible Markup Language) es un estándar de intercambio de información que supo ser el estándar en la comunicación de servicios web (en el auge del protocolo SOAP), además de ser un lenguaje muy utilizado en la especificación del layout de sitios web (por su familiaridad con HTML), como lenguaje de reports, de almacenamiento de datos y mucho más.

El formato XML se compone, al igual que HTML, de secciones llamadas elementos o nodos XML, que se delimitan por la apertura y clausura de etiquetas. Dichos elementos están compuestos de un nombre, atributos, y tienen como valor ya sea un valor específico (texto) y/o otros elementos XML (nodos hijos), y en algunos casos ningún valor.



Ejemplo de un documento XML y sus etiquetas.

La estructura de un nodo se compone de los siguiente manera:

```
<nombreElemento atributo1=valor1 atributo2=valor2 …>
  <subElemento>
    <campo1 atributo11=valor11>
      valorDelNodo
    </campo1>
  </subElemento>
</nombreElemento>
```

Donde cada línea especifica lo siguiente:

1. Se da inicio al primer elemento XML de la jerarquía, llamado “nombreElemento”, que cuenta con 2 atributos.
2. Se específica la etiqueta de inicio del nodo “subElemento”, cuyo nodo padre es “nombreElemento”. Este nodo no posee atributos.
3. Se especifica un nodo hijo de “subElemento”, cuyo nombre es “campo1”. Cuenta con un atributo.
4. Es el valor del nodo “campo1”.
5. Se delimita el cierre del nodo “campo1”.
6. Lo mismo sucede con “subElemento”.
7. Finalmente, se especifica la etiqueta de cierre del nodo abuelo “nombreElemento”.

¿Qué es xslt, xsltproc y para qué sirven?

El XML puede albergar data cruda, que especificada de esa manera quizás no tiene mucho valor, por lo cual muchas veces se desea partir de dicha data y transformarla para generar uno o más XMLs nuevos con valor más específico.

Posiblemente la herramienta más popular para el procesamiento y transformación de documentos XML es **xsltproc**, una herramienta que en base a una especificación escrita en el lenguaje **XSLT** aplica transformaciones sobre un XML y genera un documento resultante (que no necesariamente es otro XML).

El lenguaje XSLT (Extensible Stylesheet Language Transformation) es un lenguaje que está pensado para transformar archivos XML. Para ello, define reglas de transformación que se definen en un esquema, denominado *stylesheet*, que se vuelva en un archivo `.xslt`.

Objetivo del trabajo

El objetivo del trabajo es realizar una herramienta que contemple un subset de las mismas o similares funcionalidades de *xsltproc*, teniendo un alcance acotado de la especificación tanto de XML como xslt. La herramienta debe ser realizada en Scala y basada en los conceptos de la programación funcional. Para realizar este trabajo, se deberá familiarizar con el formato XML y con la funcionalidad de *xsltproc*.

Requerimientos

Importante: En caso de tomar una decisión respecto a una definición ambigua del dominio o los requerimientos, volcarlo en la correspondiente sección de hipótesis del informe.

Entrada y argumentos

El programa al invocarlo debe recibir como argumentos el path (la ruta) al archivo del *stylesheet* y el path al archivo con el XML original. Adicionalmente, también puede recibir parámetros si el stylesheet lo requiere:

```
$ xslt-prog <path_al_xslt> <path_al_xml> [parametros]
```

Ejemplo:

```
$ xslt-prog esquemas/stylesheet.xslt doc.xml --param1=juan
>> <output de la ejecución>
```

El comando debe procesar los argumentos, determinando tanto si el stylesheet como el archivo original son válidos.
Se deben interpretar y modelar internamente las reglas del esquema XSLT, de manera que sepan cómo deben aplicarse según el XML.

Se exige únicamente que el output se muestre por salida estándar. Con ellos, se puede redirigir la salida del programa hacia un archivo, con el operador clásico de la shell:

```
$ xslt-prog <path_al_xslt> <path_al_xml> [parametros] > result.out
```

Parser y sintaxis XML

El XML recibido indefectiblemente debe poder procesarse durante la ejecución del programa, con lo cual el programa debe implementar un parser de archivos XML y lógicamente determinar si un XML es válido o no. La validez solamente se determina por si todas las etiquetas están abiertas y cerradas en el orden correcto.

Para este trabajo práctico, a fin de simplificar la variabilidad de formatos que dificulten el parseo, proponemos las siguientes restricciones sobre la sintaxis XML y considerar como hipótesis que todos los documentos utilizados cumplirán con ellas:

- Cada línea cuenta únicamente con una etiqueta de inicio, una etiqueta de fin o un valor.
- No es necesario permitir las etiquetas mixtas (de inicio y fin a la vez).
    - Por ejemplo, `<algunaEtiqueta id="pepe"/>`
- No es necesario contemplar etiquetas de comentarios (i.e `<!-- comentario -->`).

Parser y sintaxis de XSLT

Paths

Una de las primeras cosas que querríamos en nuestro procesador de XSLT, es el de poder referirnos y acceder a ciertas partes de una estructura XML, ya sea por nombre de los elementos o por ubicación en el árbol XML. Dichas referencias, que llamaremos ***paths***, dependerán tanto del contexto global como del actual y de los elementos posibles.

La especificación de XSLT define un sistema de expresiones XPath, el cual es muy extenso y excede el alcance esperado de este trabajo, pero que en líneas generales nos ofrece la posibilidad de referirnos a 1 o más componentes de un árbol XML a través de una ruta (path), que pueden ser absolutas (partiendo desde el root del XML) o relativas (si uno se "encontrase" en una parte particular del documento, sería el path relativo a ese lugar).

Particularmente, nosotros optaremos por redefinir XPath y usar una versión muy simplificada, que contemple los siguientes paths:

Descripción	Path expression
Refiere al elemento root	/
Refiere al nodo actual	.
Refiere a los nodos <elem> presentes en el contexto actual	<elem>
Refiere a elementos a través de rutas compuestas en el árbol	<elem1>/<elem2> /<elem>/<subElem>
Refiere a los elementos presentes que cumplen cierta condición	<elem>[<condicion>]

Por ejemplo, tomando el siguiente documento XML:

```
<fiuba>
  <nombre>
    Facultad de Ingenieria UBA
  </nombre>
  <materias>
    <materia codigo="TB025">
       Paradigmas de Programación
    </materia>
    <materia codigo="9502">
       Algoritmos y Programación III
    </materia>
  </materias>
</fiuba>
<exactas>
  <nombre>
    Facultad de Ciencias Exactas y Naturales UBA
  </nombre>
</exactas>
```

Estos serían los resultados de las distintas expresiones, dados los contextos:

Contexto actual	Path expression	Refiere a
Nodo root	`/`	El nodo “root” del documento
	`fiuba`	El nodo “fiuba”
	`/exactas/nombre`	El nodo “nombre” dentro del elemento “exactas”
Nodo “fiuba”	`/`	El nodo root del documento
	`./nombre`	El nodo “nombre” dentro del elemento “fiuba”
	`materias/materia`	Cada uno de los nodos “materia” hijos del nodo “materias”
Nodo “materias”	`/exactas`	El nodo “exactas”
	`materia[codigo=”9502”]`	El nodo “materia” de “Algoritmos y Programación III”
	`materia[0]`	El primero de los nodos “materia”
	`nombre`	No refiere a ningún nodo

(Opcional) Apartado de variables y parámetros

Algunas reglas opcionales a implementar, contempla la creación de variables que pueden utilizarse dentro de paths o como valores en los cuerpos de los nodos. Por lo tanto, hay que contemplar la existencia de dichas variables en el scope, sus ocurrencias y sus correctos reemplazos por el valor de la variable.

Dichas variables se especifican en los paths y en los cuerpos de elementos con la simbología `$nombreVariable`. Básicamente, cualquier ocurrencia de dicho símbolo, debe ser reemplazado al momento de renderizarlo.

Las variables (tanto definidas por reglas o asignadas por parámetros) son inmutables por diseño, con lo cual una vez seteada, no serán redefinidas más adelante. Si se intentase, debe resultar en un error.

Reglas y transformaciones

La idea del trabajo es implementar la mayor cantidad de reglas XSLT posibles. Las identificadas como más útiles para una primera versión de la herramienta son las que se

mencionan a continuación. Aquellas que están marcadas por un signo de exclamación rojo son consideradas ***indispensables*** para el uso de la herramienta.

Nota: Todas las reglas empezaran con una etiqueta de prefijo "`pxls:`".

Etiqueta de la regla	Descripción de la regla

! `<pxsl:template match=[path]>`
`  {cuerpo del template}`
`</pxsl:template>`
	Descripción: Operación base de toda transformación XSLT. Dado un contexto actual, identifica cada uno de los elementos referidos por el path y opera sobre cada uno de ellos, estableciendo generando un output
	
	Parámetros:
	- match (path): indica el path por los objetos a los que debe referir.
	
	Output generado:
	Para cada elemento referido, genera el output resultante del cuerpo del template, separados por un salto de línea.

! `<pxsl:value-of`
`  select=[path]`
`/>`
`</pxsl:value-of>`
 	
o
 	
`<pxsl:value-of select=[path]/>`
	Descripción: Genera el valor del o de los nodos a los que apunta path. Ejemplos:
	- `select="nombre"` → refiere a los nodos "nombre"
	- `select="."` → refiere al nodo actual
	
	Parámetros:
	- select (path): indica el path al objeto al cual referir.
	
	Output generado:
	Para cada elemento referido, escupe el valor del elemento. Se encadenan consecutivamente, sin espacios ni saltos de linea.

! `<pxsl:copy-of select=[path]>`
`</pxsl:copy-of>`
 	
o
 	
`<pxsl:copy-of select=[path]/>`
	Descripción: Toma a los elementos XML a los que apunta el path, y los genera en la salida tal cual estaban especificados en el XML original (incluyendo todos sus descendientes).
	
	Parámetros:
	- select (path): indica el path para los elementos a los que debe referir.
	
	Output generado:
	Para cada elemento referido, escupe su valor XML tal cual como era originalmente. Los valores se encadenan con saltos de línea de por medio.

! `<pxsl:copy select=[path]>`
`  {cuerpo}`
`</pxsl:copy>`
	Descripción: Igual que el copy-of, pero sin incluir a los descendientes en la copia. Además, deja definir nuevos elementos hijos en el cuerpo de la etiqueta.
	
	Parámetros:
	- select (path): indica el path para los elementos a los que debe referir.
	
	Output generado:
	Para cada elemento referido, escupe su valor XML con sus atributos y valor, pero sin sus descendientes, y con el adicional del cuerpo especificado. Se encadenan con saltos de línea de por medio.

`<pxsl:param id=[nombreParam]>`
`</pxsl:param>`
 	
o
 	
`<pxsl:param id=[nombreParam]/>`
	Descripción: Hace un binding del valor pasado por parámetro al programa (con `--param`) a una variable “nombreParam” que luego puedan ser utilizadas luego en las transformaciones y los paths.
	
	Parámetros:
	- id (string): nombre de uno parámetros pasados al programa.
	
	Output generado:
	No genera output.

`<pxsl:variable`
`  name=[nombreVar]`
`  value=[valor]`
`>`
`</pxsl:param>`
 	
o
 	
`<pxsl:param id=[nombreParam]/>`
	Descripción: Define una variable con un determinado valor, que luego pueda ser utilizada luego en las transformaciones y los paths.
	
	Parámetros:
	- name (string): nombre de la variable.
	- value (string): valor de la variable.
	
	Output generado:
	No genera output.

`<pxsl:attribute-of`
`  select=[path]`
`  name=[attributeName]`
`/>`
`</pxsl:attribute-of>`
	Descripción: Igual que el value-of, pero en lugar del valor del nodo, genera el valor del atributo especificado.
	
	Parámetros:
	- select (path): indica el path al nodo al cual referir.
	- name (string): nombre del atributo a obtener.
	
	Output generado:
	Para cada elemento referido, escupe el valor del atributo correspondiente del elemento. Se encadenan consecutivamente, sin espacios ni saltos de línea.

`<pxsl:sort-by-attr`
`  name=[attributeName]`
`  order=[desc|asc]`
`>`
`  {cuerpo}`
`</pxsl:sort>`
	Descripción: Esta operación toma el resultado generado por el cuerpo de la etiqueta, y lo retorna pero con cada uno de los elementos (del primer nivel) ordenados por un atributo.
	
	Parámetros:
	- name (string): nombre del atributo por el cual ordenar.
	- order (string): `asc` para ordenar de menor a mayor, y `desc` para lo contrario.
	
	Output generado:
	El output generado por el cuerpo, con sus elementos ordenados por el atributo pasado, en el orden especificado. En caso que algún elemento no tenga el atributo, se considera menor a los que sí lo tienen.

`<pxsl:sort-by-elem`
`  elem=[path]`
`  order=[desc|asc]`
`>`
`  {cuerpo}`
`</pxsl:sort>`
	Descripción: Esta operación toma el resultado generado por el cuerpo de la etiqueta, y lo retorna pero con cada uno de los elementos ordenados por el valor de uno de los subelementos en su estructura.
	
	Parámetros:
	- elem (path): path a los subelementos de cada elemento en el primer nivel.
	- order (string): `asc` para ordenar de menor a mayor, y `desc` para lo contrario.
	
	Output generado:
	El output generado por el cuerpo, con sus elementos ordenados por el valor de los subelementos elegidos, en el orden especificado. En caso que algún elemento no contenga dicho subelemento, se considera menor a los que sí lo tienen.

Aclaraciones adicionales y diferencias con XSLT estándar

En las stylesheets XSLT, generalmente el cuerpo arranca con un encabezado `<xls:stylesheet>`. En nuestro caso, no será necesario.

Cabe recalcar que la funcionalidad de nuestras reglas y esquema de paths es bastante acotado respecto a XSLT, con lo cual el gap funcional entre una herramienta y otra será considerable y de ninguna manera PXSLT será un reemplazo de XSLT.
Esto es lo esperado, ya que solamente se utiliza el dominio de la herramienta para modelar un concepto de trabajo práctico. Lo que se hizo fue reducir la generalización de las reglas y se introdujeron reglas nuevas (como `attribute-of`) para realizar la misma funcionalidad de una manera más sencilla.

Informe

Se espera la realización de un informe que contenga:

- Cómo se compila y corre el programa.
- Qué funcionalidades fueron implementadas, con uno más ejemplos que hayan utilizado para validar su correcta ejecución.
- Decisiones tomadas respecto al procesamiento, modelado en ejecución y manipulación de los documentos XML y stylesheets XSLT.
- Detección de los casos y fragmentos del código donde se vieron en la necesidad de salirse de los conceptos de la programación funcional, justificando el por qué debieron hacerlo.
- Una sección de las hipótesis tomadas durante la realización del trabajo.
- Conclusiones.

Criterios de aprobación

Para que un TP se considere aprobado deberá al menos cumplir con los siguientes criterios mínimos:

- Sintaxis: se espera que se contemplen valores de atributos de 1 sola palabra. No es necesario contemplar valores con espacios en el medio.
- Debe funcionar con los argumentos de path al XML y path al XSLT, aunque no es necesario que funcione con parámetros.
- Es necesario contemplar los paths expressions por root, por nodo actual y por elemento simple. No así los paths de elementos compuestos, o por condición.
- Deben estar implementadas todas las reglas XSLT indispensables, denotadas con el símbolo “ **!** ” en el enunciado.
- No es necesario contemplar manejo de errores exhaustivo. Únicamente la validez de los documentos XML según lo especificado en dicha sección.
- El código debe respetar completamente los lineamientos del paradigma funcional.

Si bien estas son las funcionalidades mínimas requeridas para la aprobación, un TP se considerará completo solo si cumple con todos los requerimientos descritos en las secciones anteriores. Para alcanzar la nota máxima del trabajo, se espera que esté implementada toda la funcionalidad descrita en el enunciado, con su correcto manejo de errores.

Criterios de promoción

Para la promoción, se deberán implementar al menos 7 de las reglas especificadas (es decir, las 4 obligatorias y 3 elegidas por el grupo) y debe contemplar los paths compuestos y/o por condición (al menos uno de los 2). Al menos uno de los sortings y uno de las reglas de variables (`param` o `variable`) deben ser implementadas.
Además, debe manejar agraciadamente la mayoría de errores posibles, no fallando con errores inesperados por el programa.