XSLT
Extensible Stylesheet Language Transformations

Mauro Gaio

Université de Pau et des Pays de l'Adour
Département d'Informatique
mauro.gaio@univ-pau.fr

24 octobre 2007

Plan

Avant-propos

XSLT est une technologie puissante !

- XSLT permet de transformer des documents XML
- XSLT utilise des règles *(templates)* pour transformer
- à la fois aux frontières et au cœur du dev. de logiciels actuels
- Son apprentissage passe par des recettes

Plan

Structuration avec XML

- représenter des contenus indépendamment de l'application
- combinaison de plusieurs principes simples et généraux

```
<?xml version=''1.0'' encoding=''utf-8''?>
<cinema>
   <nom> Le Méliès</nom>
   <adresse>6, Rue Bargoin.</adresse>
   <ville>Pau</ville>
   <departement>64000</departement>
   <salle num='1' places='200'>
                <titre>Un secret</titre>
                <seances><seance>21 :00</seance></seances>
   </salle>
</cinema>
```

Structuration avec XML

- représenter des contenus indépendamment de l'application
- combinaison de plusieurs principes simples et généraux

- la structurartion avec XML permet : de **désigner** des parties d'un contenu (chaîne de caractère) avec des noms et de **hiérarchiser** ces noms
- une information structurée avec XML consiste à considérer que celle-ci est un **arbre**

Structuration avec XML

- représenter des contenus indépendamment de l'application
- combinaison de plusieurs principes simples et généraux

- la structurartion avec XML permet : de **désigner** des parties d'un contenu (chaîne de caractère) avec des noms et de **hiérarchiser** ces noms
- une information structurée avec XML consiste à considérer que celle-ci est un **arbre**

Structuration avec XML

- la structurartion avec XML permet : de **désigner** des parties d'un contenu (chaîne de caractère) avec des noms et de **hiérarchiser** ces noms
- une information structurée avec XML consiste à considérer que celle-ci est un **arbre**

La désignation est réalisée par des balises :
`<cinema>,<nom>,</nom>,<adresse>,</adresse>,<ville>,</ville>, <departement>,</departement> ...et </cinema>`

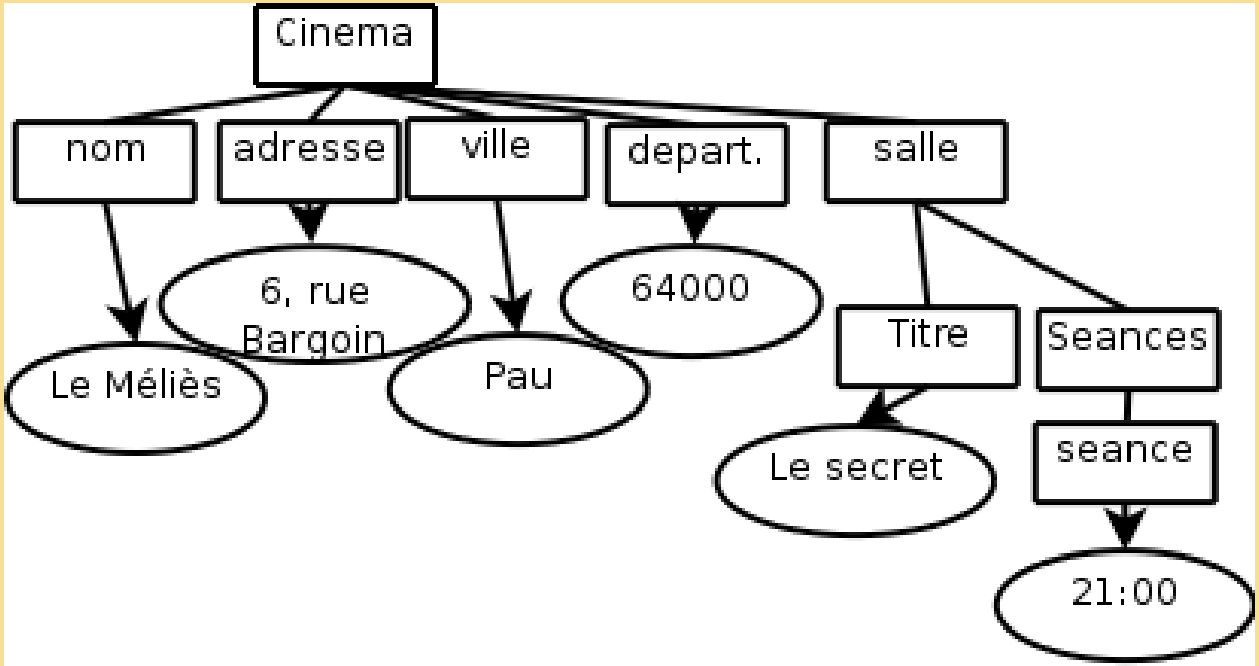
Structuration avec XML

- la structurartion avec XML permet : de **désigner** des parties d'un contenu (chaîne de caractère) avec des noms et de **hiérarchiser** ces noms
- une information structurée avec XML consiste à considérer que celle-ci est un **arbre**

- Une paire **désigne** (type) une partie du contenu : *l'élémént XML*
- La hiérarchie entre les différentes paires **structure** le contenu

Structuration avec XML

- la structurartion avec XML permet : de **désigner** des parties d'un contenu (chaîne de caractère) avec des noms et de **hiérarchiser** ces noms
- une information structurée avec XML consiste à considérer que celle-ci est un **arbre**



Structuration avec XML

- la structurartion avec XML permet : de **désigner** des parties d'un contenu (chaîne de caractère) avec des noms et de **hiérarchiser** ces noms
- une information structurée avec XML consiste à considérer que celle-ci est un **arbre**

à retenir :

- un élément est caractérisé à la fois par son nom et par sa place dans l'arbre XML
- tout traitement se fonde sur des outils permettant de choisir des éléments par :
  - leur nom
  - ou leur position
  - ou par les 2 dimensions

Structuration avec XML

- la structurartion avec XML permet : de **désigner** des parties d'un contenu (chaîne de caractère) avec des noms et de **hiérarchiser** ces noms
- une information structurée avec XML consiste à considérer que celle-ci est un **arbre**

à retenir :

- un élément est caractérisé à la fois par son nom et par sa place dans l'arbre XML
- tout traitement se fonde sur des outils permettant de choisir des éléments par :
  - leur nom
  - ou leur position
  - ou par les 2 dimensions

Documents XML

- la structure d'un document XML est définissable et validable par un **schéma**
- un document XML est **entièrement transformable** dans un autre document XML.

- *Document Type Definition* DTD
- XMLSchema

Documents XML

- la structure d'un document XML est définissable et validable par un **schéma**
- un document XML est **entièrement transformable** dans un autre document XML.

```
<!ELEMENT cinema(nom, adresse, ville, departement, (salle)+)
<!ELEMENT nom(#PCDATA)>
...
<!ELEMENT salle (titre, (seances)+)>
<!ATTLIST salle num CDATA #REQUIRED
                 place CDATA #IMPLIED>
<!ELEMENT seances ((seance)+)>
<!ELEMENT seance(#PCDATA)>
```

Documents XML

- la structure d'un document XML est définissable et validable par un **schéma**
- un document XML est **entièrement transformable** dans un autre document XML.

La même chose mais en xml-schema

d'abord les éléments de type simple :

```
<xs :schema xmlns :xs=''http ://www.w3.org/2001/XMLSchema''>
.../...
<xs :element name=''nom'' type=''xs :string''/>
<xs :element name=''departement'' type=''xs :integer''/>
.../...
<xs :element name=''seance'' type=''xs :date''/>
```

Documents XML

- la structure d'un document XML est définissable et validable par un **schéma**
- un document XML est **entièrement transformable** dans un autre document XML.

ensuite les éléments de type complexe :

```
<xs :element name=''cinema''>
   <xs :complexType>
     <xs :sequence>
         <xs :element ref=''nom''/> .../...
         <xs :element ref=''seances''/>
      </xs :sequence>
    </xs :complexType>
</xs :element>
```

Documents XML

- la structure d'un document XML est définissable et validable par un **schéma**
- un document XML est **entièrement transformable** dans un autre document XML.

```
<xs :element name=''seances''>
   <xs :complexType>
    <xs :sequence>
      <xs :element ref=''seance'' maxOccurs=''unbounded''/>
    </xs :sequence>
   </xs :complexType>
</xs :element>
```

Les attrib. minOccurs et maxOccurs sont par défaut à 1, donc lorsque omis : l'élément doit apparaître une et une seule fois

Documents XML

- la structure d'un document XML est définissable et validable par un **schéma**
- un document XML est **entièrement transformable** dans un autre document XML.

Les schémas ont plusieurs avantages :

- un éditeur se sert du schéma pour faciliter l'édition d'un doc. XML et/ou vérifier sa conformité pour rapport à son schéma
- Un programme XSLT se base sur le schéma pour l'origine et la destination de la transformation et/ou à la vérification des règles XSLT
- En combinaison avec une analyse validante, le processeur XSLT vérifie avant l'éxécution de la transformation si le doc. à transformer est conforme

Plan

Publication de données

L'entête habituelle :

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="1.0">
```

```
<xsl:template match="/">
  <html>
    <head>
      <title>Le cinéma <xsl:value-of select="nom"/>
      </title>
    </head>
    <body>
    <p>
<!--Insérer une image GIF avec le titre comme nom-->
      <img src="{nom}.gif" align="left" height="220"/>
      <h1> <xsl:value-of select="nom"/> </h1>
           <xsl:value-of select="adresse"/>, <br/>
      <i>  <xsl:value-of select="departement"/></i>,
           <xsl:value-of select="ville"/>
    </p>
```

```
<xsl:apply-templates select="salle"/>
  </body>
  </html>
    </xsl:template>

    <xsl:template match="salle">
<!-- Extraction d'attribut : no de salle et places -->
      <h2>Salle No <xsl:value-of select="@num"/>,
      <xsl:value-of select="@places"/> places </h2>
      <!-- Boucles sur toutes les séances -->
      <h3>Séances</h3><ol>
      <xsl:for-each select="seances/seance">
        <li><xsl:value-of select="."/></li>
        </xsl:for-each></ol>
      </xsl:template>
 </xsl:stylesheet>
```

Transformer des chaîne de caractères

La manipulation des chaînes de catactères : la forme de transformation la plus primitive

Pour des manipulations complexes les programmeurs ont 2 choix :

1. faire appel à des fonctions écrites dans un autre langage (Java ;-)) lorsque la portabilité n'est pas critique
2. écrire des fonctions avancées en XSLT

Transformer des chaîne de caractères

La manipulation des chaînes de catactères : la forme de transformation la plus primitive

Pour des manipulations complexes les programmeurs ont 2 choix :

1. faire appel à des fonctions écrites dans un autre langage (Java ;-)) lorsque la portabilité n'est pas critique
2. écrire des fonctions avancées en XSLT

- XSLT ne dispose pas de l'artillerie lourde offerte par Perl
- XSLT offre principalement 9 fonctions de base

Supprimer/remplacer certains caractères d'une chaîne

string *translate* (string *chaine*, string *arg*1, string *arg*2)
Exemple : supprimer tous les caractères numérique qui apparaissent dans une chaîne donnée.

```
<xsl:param name=''chaine''>
<xsl:template name=''supprime''>
    <xsl:value-of select=
     ''translate($chaine, '0123456789','')''/>
</xsl:template>
```

la valeur du paramètre $chaine peut être obtenue via la méthode GET du protocole HTTP) :
http ://www.bidouille.net/transforme.xsl ?chaine=12 avril 2012 12 :30

les fonctions de bases

- string *concat* (string *chaine*1, string *chaine*2, string *chaines*$*$)
  `<!--concatenation d'au moins 2 chaînes-->`

les fonctions de bases

- string *concat* (string *chaine*1, string *chaine*2, string *chaines*$*$)
  `<!--concatenation d'au moins 2 chaînes-->`
- Boolean *contains* (string *chaine*, string *sous* − *chaine*)`<!--test si` *chaine* `contient la` *sous* − *chaine*`-->`

les fonctions de bases

- string *concat* (string *chaine*1, string *chaine*2, string *chaines*$*$) `<!--concatenation d'au moins 2 chaînes-->`
- Boolean *contains* (string *chaine*, string *sous* – *chaine*)`<!--test si chaine contient la sous – chaine-->`
- string *normalize* – *space*(string *chaine*?) `<!--supprimer tous les espaces début, fin et remplacer à 1 les consécutifs-->`

les fonctions de bases

- string *concat* (string *chaine*1, string *chaine*2, string *chaines*$*$) <!--concatenation d'au moins 2 chaînes-->
- Boolean *contains* (string *chaine*, string *sous* – *chaine*)<!--test si *chaine* contient la *sous* – *chaine*-->
- string *normalize* – *space*(string *chaine*?) <!--supprimer tous les espaces début, fin et remplacer à 1 les consécutifs-->
- Boolean *starts* – *with* (string *chaine*, string *sous* – *chaine*) <!--si chaîne débute avec sous-chaîne-->

les fonctions de bases

- string *concat* (string *chaine*1, string *chaine*2, string *chaines*$*$) `<!--concatenation d'au moins 2 chaînes-->`
- Boolean *contains* (string *chaine*, string *sous* – *chaine*)`<!--test si chaine contient la sous – chaine-->`
- string *normalize* – *space*(string *chaine*?) `<!--supprimer tous les espaces début, fin et remplacer à 1 les consécutifs-->`
- Boolean *starts* – *with* (string *chaine*, string *sous* – *chaine*) `<!--si chaîne débute avec sous-chaîne-->`
- number *string* – *length*(string *chaine*?) `<!--longueur-->`

les fonctions de bases

- string *concat* (string *chaine*1, string *chaine*2, string *chaines*$*$) <!--concatenation d'au moins 2 chaînes-->
- Boolean *contains* (string *chaine*, string *sous* – *chaine*)<!--test si *chaine* contient la *sous* – *chaine*-->
- string *normalize* – *space*(string *chaine*?) <!--supprimer tous les espaces début, fin et remplacer à 1 les consécutifs-->
- Boolean *starts* – *with* (string *chaine*, string *sous* – *chaine*) <!--si chaîne débute avec sous-chaîne-->
- number *string* – *length*(string *chaine*?) <!--longueur-->
- string *substring* (string *chaine*, number *debut*, number *longueur*?) <!--extraire une sous-chaîne à partir de *debut* et de la taille de *longueur*-->

les fonctions de bases

- string *concat* (string *chaine*1, string *chaine*2, string *chaines*∗) <!--concatenation d'au moins 2 chaînes-->
- Boolean *contains* (string *chaine*, string *sous* − *chaine*)<!--test si *chaine* contient la *sous* − *chaine*-->
- string *normalize* − *space*(string *chaine*?) <!--supprimer tous les espaces début, fin et remplacer à 1 les consécutifs-->
- Boolean *starts* − *with* (string *chaine*, string *sous* − *chaine*) <!--si chaîne débute avec sous-chaîne-->
- number *string* − *length*(string *chaine*?) <!--longueur-->
- string *substring* (string *chaine*, number *debut*, number *longueur*?) <!--extraire une sous-chaîne à partir de *debut* et de la taille de *longueur*-->
- string *substring* − *before* (string *chaine*, string *sous* − *chaine*) <!--la chaîne avant la première occurence de *sous* − *chaine*-->

Nombres, op. math. et date / heure

Seules le opérations arithmétiques de base sont disponibles : compter, aditionner et formater des nombres

Ex : Calcul de la valeur absolue *math* : $abs(x)$

```
    <xsl:template name="math:abs">
 <xsl:param name="x"/>
    <xsl:choose>
        <xsl:when test="$x &lt; 0">
            <xsl:value-of select="$x*-1"/>
        </xsl:when>
        <xsl:otherwise>
            <xsl:value-of select="$x"/>
        </xsl:otherwise>
        </xsl:choose>
</xsl:template>
</xsl:stylesheet>
```

Nombres, op. math. et date / heure

- La page Web : http ://www.incrementaldevelopment.com/xsltrick/ contient quelques curiosités mathématiques écrites en XSLT.
- XSLT 1.0 ne sait pas quelle heure il est. . .

Nombres, op. math. et date / heure

- La page Web : http ://www.incrementaldevelopment.com/xsltrick/ contient quelques curiosités mathématiques écrites en XSLT.
- XSLT 1.0 ne sait pas quelle heure il est. . .
  l'obtention de la date et de l'heure courants ne peut être implémentée.

Nombres, op. math. et date / heure

- La page Web : http ://www.incrementaldevelopment.com/xsltrick/ contient quelques curiosités mathématiques écrites en XSLT.
- XSLT 1.0 ne sait pas quelle heure il est. . .
- Ou alors en passant par : http ://www.exslt.org/date/index.html
- sachant que EXSLT.org est une communauté qui propose des extensions de XSLT 1.0 dans différents domaines et notamment : dans les fonctions math. y compris des opérateurs ensemblistes, les dates et les heures, les expressions régulières . . .

Nombres, op. math. et date / heure

- La page Web : http ://www.incrementaldevelopment.com/xsltrick/ contient quelques curiosités mathématiques écrites en XSLT.
- XSLT 1.0 ne sait pas quelle heure il est. . .
- Ou alors en passant par : http ://www.exslt.org/date/index.html
- sachant que EXSLT.org est une communauté qui propose des extensions de XSLT 1.0 dans différents domaines et notamment : dans les fonctions math. y compris des opérateurs ensemblistes, les dates et les heures, les expressions régulières . . .
- nombreuses de ces extensions sont reprises dans **XSLT 2.0** à confirmer

Sélection et Parcours

Un traitement en XSLT implique 2 opérations :

1. déterminer les élements à consulter (sélection)
2. connaître l'ordre dans lequel le faire (parcours)

Sélection et Parcours

Un traitement en XSLT implique 2 opérations :

1. déterminer les élements à consulter (sélection)
2. connaître l'ordre dans lequel le faire (parcours)

- La sélection est du ressort de XPath, une spéc. distincte mais fortement liées à XSLT

Sélection et Parcours

Un traitement en XSLT implique 2 opérations :

1. déterminer les élements à consulter (sélection)
2. connaître l'ordre dans lequel le faire (parcours)

- Le parcours dépend des structures de contrôles intégrées dans XSLT et la façon dont les règles sont organnisées à partir de ces structures

Le langage XPath

Dans XPath le référencement de nœuds s'effectue en décrivant des parcours dans l'arbre par exemple dans les instructions XSLT suivantes :

1. `<xsl :apply-templates select=''cinema/salle''/>`
   sélection de tous les nœuds de type "salle" fils d'un nœud de type "cinema" lui-même fils d'un nœud contexte.

Le langage XPath

Dans XPath le référencement de nœuds s'effectue en décrivant des parcours dans l'arbre par exemple dans les instructions XSLT suivantes :

1. `<xsl :apply-templates select=''cinema/salle''/>`
   sélection de tous les nœuds de type "salle" fils d'un nœud de type "cinema" lui-même fils d'un nœud contexte.
2. `<xsl :apply-templates match=''cinema/salle''/>`
   désignation des nœuds auxquels s'applique une règle

Le langage XPath

Dans XPath le référencement de nœuds s'effectue en décrivant des parcours dans l'arbre par exemple dans les instructions XSLT suivantes :

1. `<xsl :apply-templates select=''cinema/salle''/>`
   sélection de tous les nœuds de type "salle" fils d'un nœud de type "cinema" lui-même fils d'un nœud contexte.
2. `<xsl :apply-templates match=''cinema/salle''/>`
   désignation des nœuds auxquels s'applique une règle
3. `<xsl :value-of select=''salle/@num''>`
   extraction de valeurs

Le langage XPath

En synthèse :

Une *expression XPath* désigne un ou ++ nœuds en exprimant des chemins dans un arbre XML. Leur évaluation selon *le contexte* donne :

- soit une valeur numérique ou alphanumérique
- soit un sous-ensemble des nœuds de l'arbre.

Le langage XPath

Une expression XPath est constituée d'une suite d'étapes, matérialisées par des " /"
Chaque étape est elle-même divisible en trois composants :

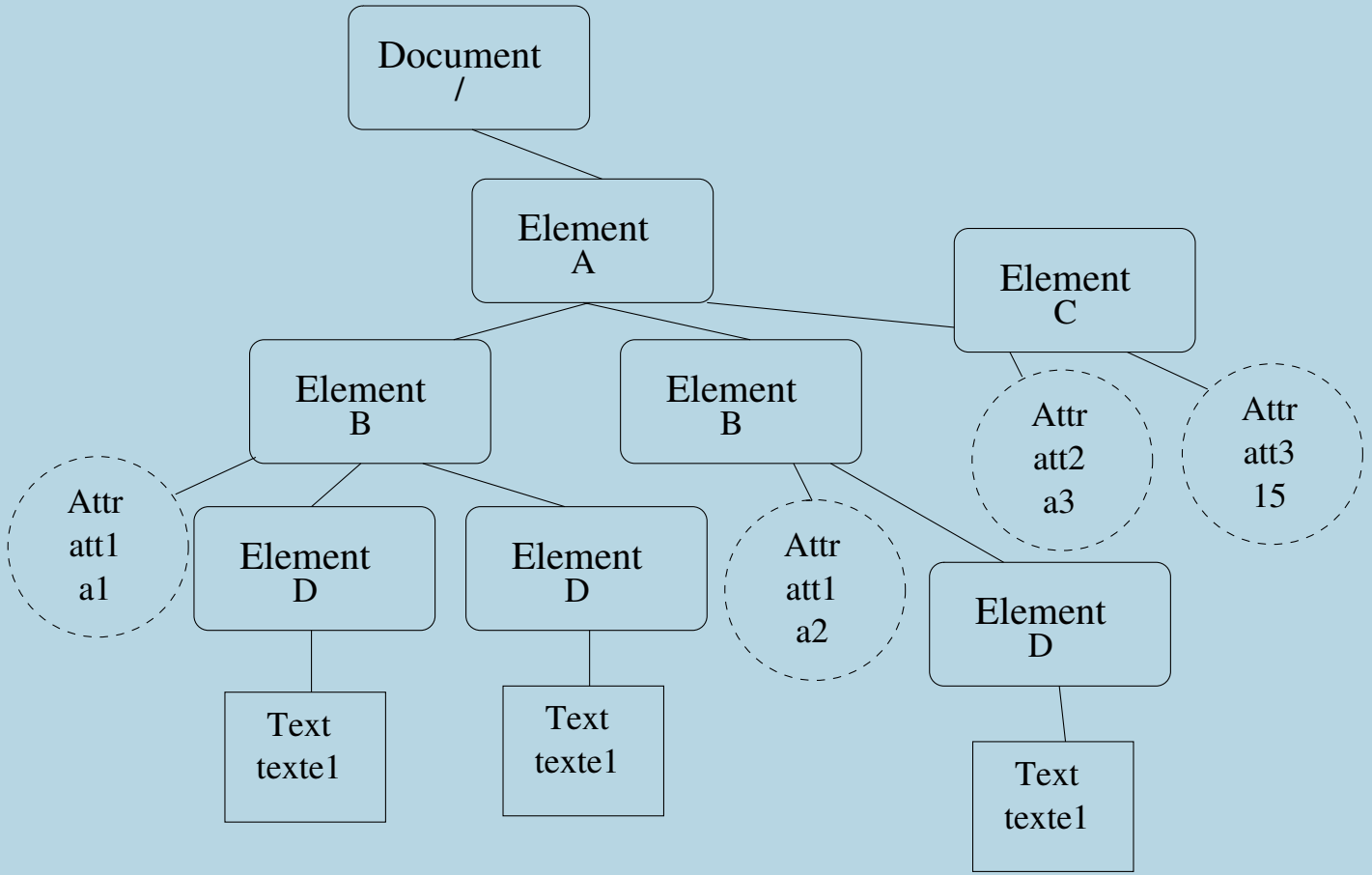
l'axe sens de parcours des nœuds à partir du nœud contexte ;

le filtre indiquant le type de nœud à retenir dans l'évaluation

le(s) prédicat(s) exprimant les propriétés que doivent satisfaire les nœuds retenus après filtrage pour être retenus dans le résultat.

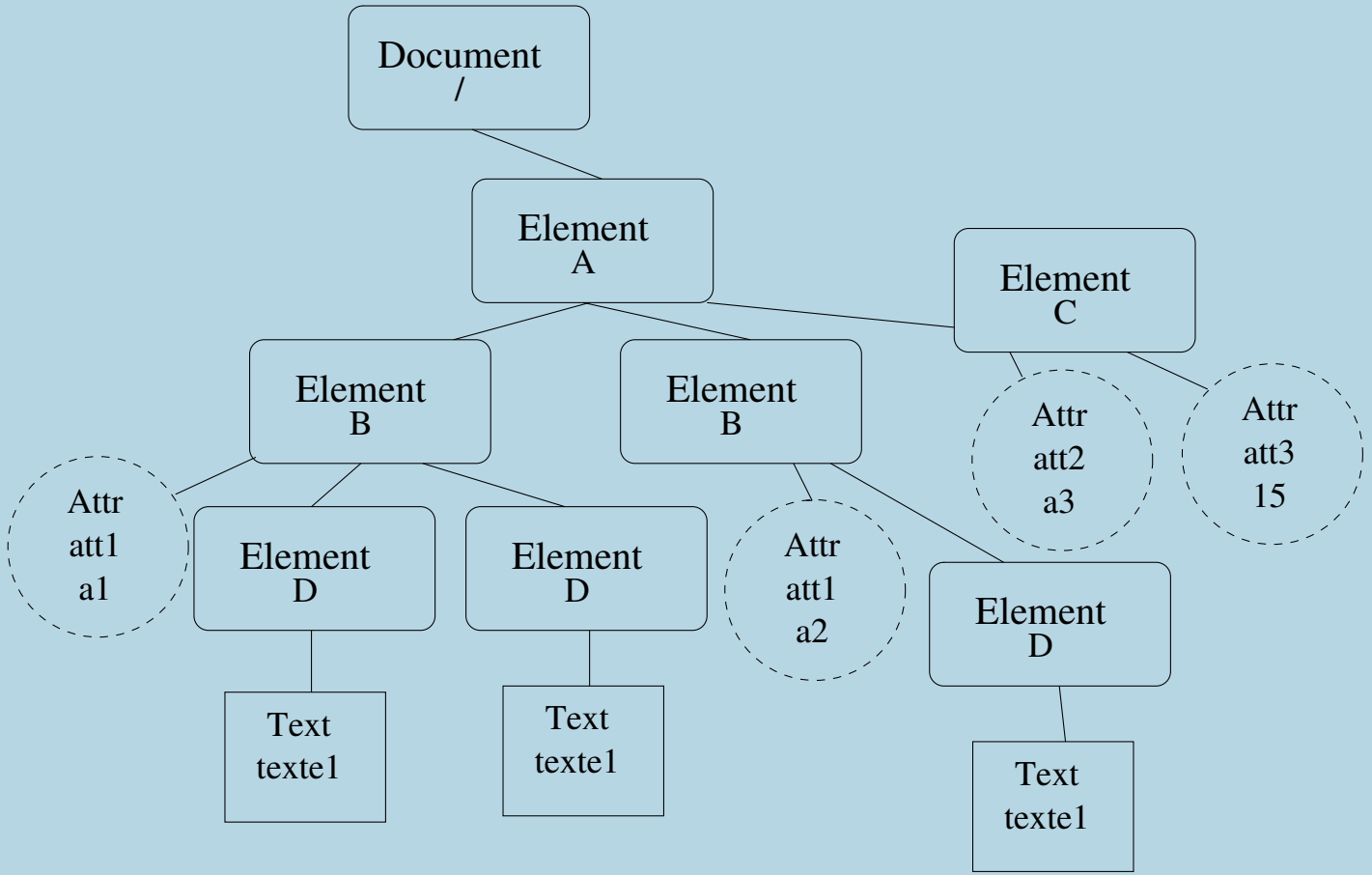
Les axes

/A/B/@att1



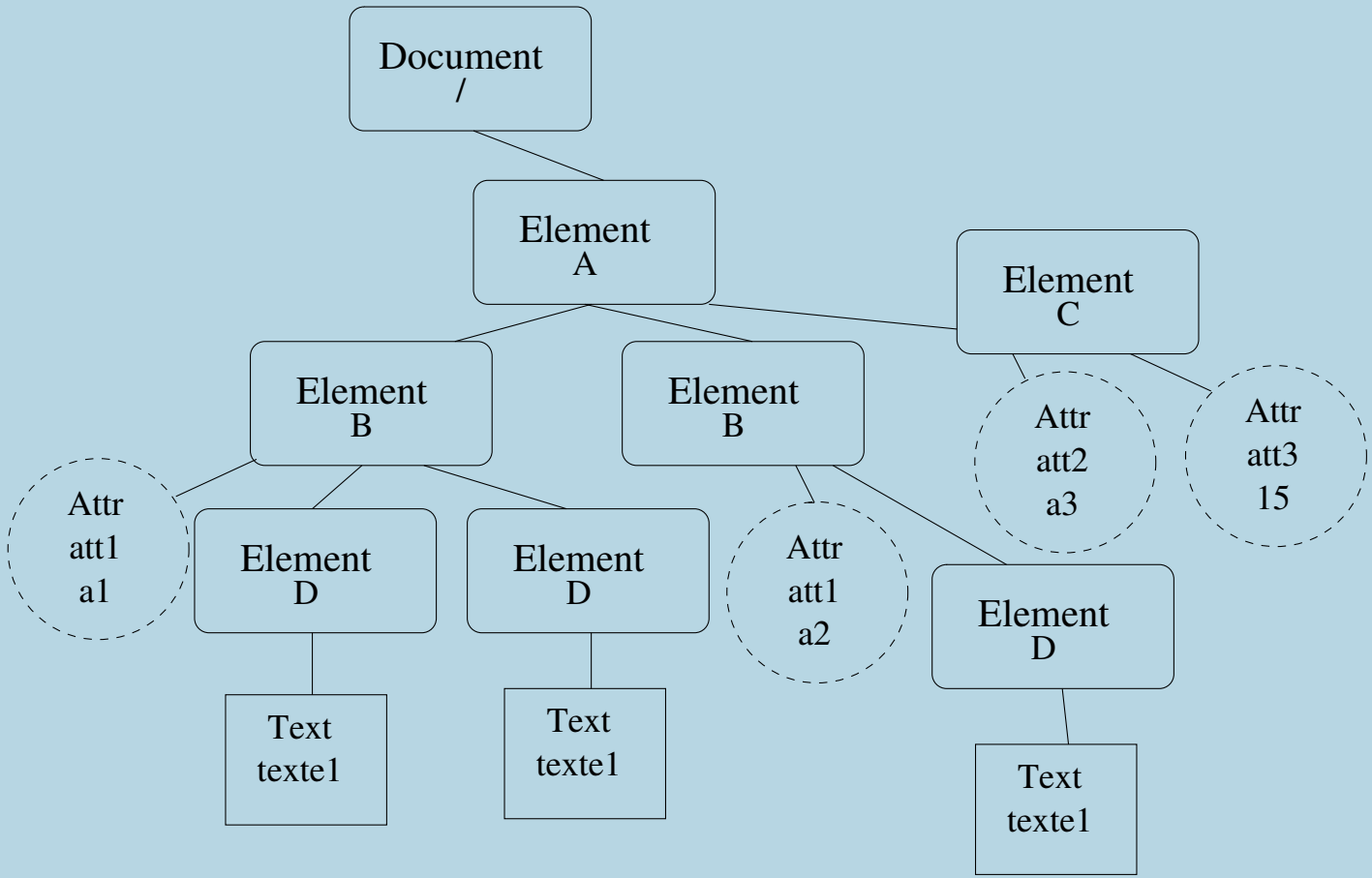
Les axes

/A/B/@att1 = /child : :A/child : :B/attribute : :att1



Les axes

child, attribute, parent, descendant, ancestor . . .



Les filtres

Deux types de filtrage pour selectionner des nœuds : par nom et par type

- *Filtre avec noms d'éléments* /A/B/D

Les filtres

Deux types de filtrage pour selectionner des nœuds : par nom et par type

- *Filtre avec nom d'attributs* /descendant : :node()/@att1

nom d'attrib. dernière étape car attrib. toujours nœud feuille

Les filtres

Deux types de filtrage pour selectionner des nœuds : par nom et par type

- *Filtre par type de nœud* /comment()

Désigne tous les commentaires fils de la racine

Les filtres

Deux types de filtrage pour selectionner des nœuds : par nom et par type

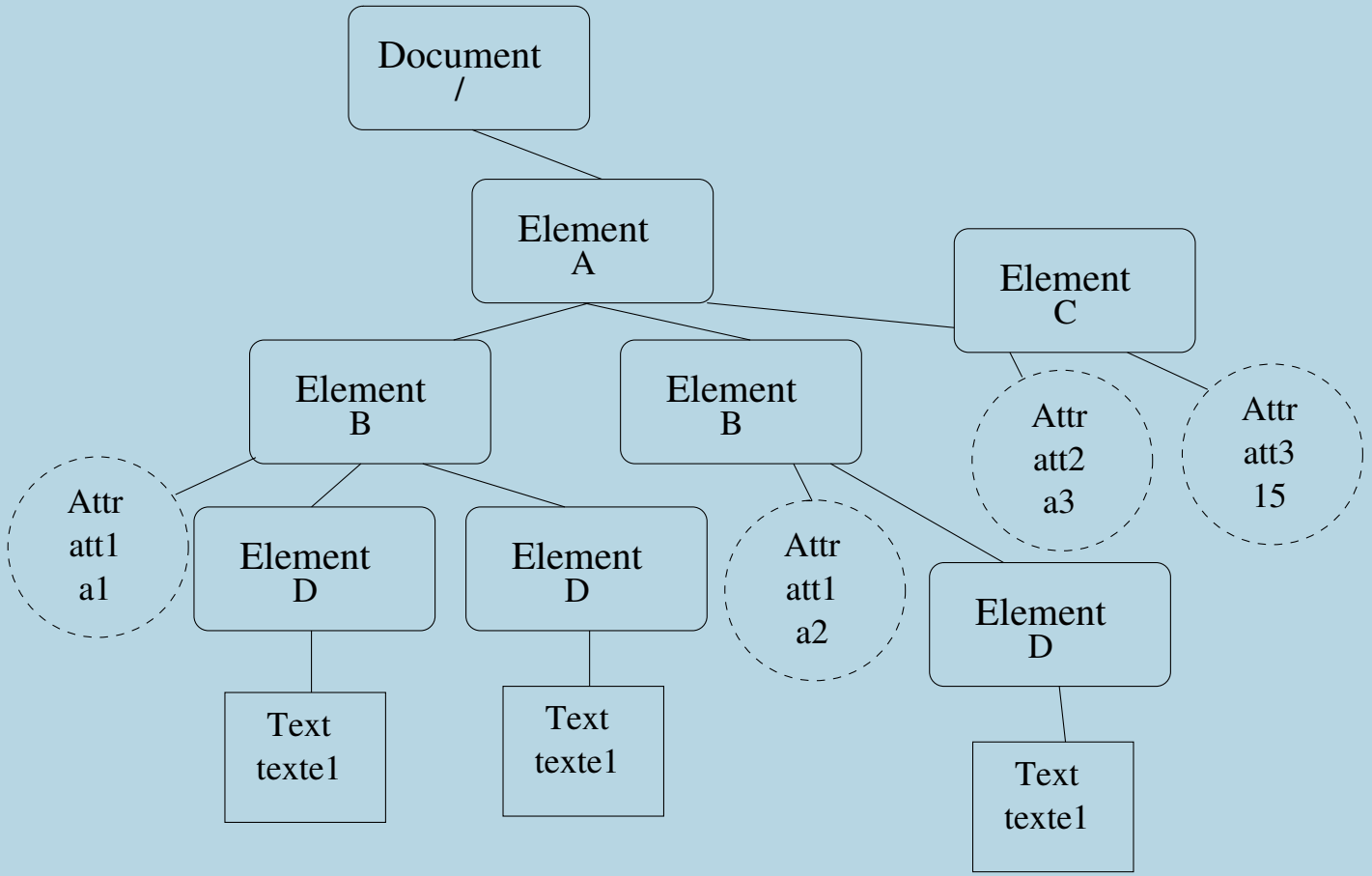
- *Filtre par type de nœud* /comment()

Filtre	Type de nœuds désignés
`text()`	Text
`comment()`	Commentaire "Comment"
`processing-instruction()`	instructions d'éxécution
`*`	Element (sauf attribut) ou Attr (avec axe `attribute`)
`node()`	n'importe quel type de nœud

Les filtres

Exemple d'une expression avec `text()`

/A/B//text() = /child : :A/child : :B/descendant-or-self : :node()/text()



Les Prédicats

La 3ième partie d'une étape est un `prédicat` :
autrement dit, une exp. booléenne constituées d'un ou ++ tests, composés avec les connecteurs logiques `and`, `or` et la négation fournie sous la forme d'une fonction `not()`
La forme générale d'une étape est donc
`axe : :filtre[prédicat1][prédicat2]...`

Les Prédicats

La 3ième partie d'une étape est un `prédicat` :
autrement dit, une exp. booléenne constituées d'un ou ++ tests, composés avec les connecteurs logiques `and`, `or` et la négation fournie sous la forme d'une fonction `not()`
La forme générale d'une étape est donc
`axe : :filtre[prédicat1][prédicat2]...`
à peu près n'importe quelle expression XPath peut être convertie en un booléen

Dans l'expression : /A/B[@att1]
s'il n'y a pas d'attribut de ce nom la valeur booléenne de ce prédicat est `false`

Les Prédicats

La 3ième partie d'une étape est un `prédicat` :
autrement dit, une exp. booléenne constituées d'un ou ++ tests, composés avec les connecteurs logiques `and`, `or` et la négation fournie sous la forme d'une fonction `not()`
La forme générale d'une étape est donc
`axe : :filtre[prédicat1][prédicat2]...`

Composition de prédicats /A/B[@att1='a1'][position()=last()]
tous les éléments B ayant comme attribut att1 valant a1 (premier prédicat) et on ne conserve que le dernier (second prédicat)

Les Prédicats

La 3ième partie d'une étape est un `prédicat` :
autrement dit, une exp. booléenne constituées d'un ou ++ tests, composés avec les connecteurs logiques `and`, `or` et la négation fournie sous la forme d'une fonction `not()`
La forme générale d'une étape est donc
`axe : :filtre[prédicat1][prédicat2]...`

Composition de prédicats /A/B[@att1='a1'][position()=last()]
tous les éléments B ayant comme attribut att1 valant a1 (premier prédicat) et on ne conserve que le dernier (second prédicat)

L'utilisation d'une succession de [ ] dénote une *composition* $\neq$ avec l'utilisation de connecteurs logiques : l'ens. des nœuds du 1^{er} prédicat est pris comme contexte du suivant.

Génération de code

La plupart des derniers progrès dans le développement logiciel sont liés à la notion de génération automatique de code à partir de spécifications de plus haut niveau.

Génération de code

La plupart des derniers progrès dans le développement logiciel sont liés à la notion de génération automatique de code à partir de spécifications de plus haut niveau.

- Le langage cible n'est pas du code machine executable

Génération de code

La plupart des derniers progrès dans le développement logiciel sont liés à la notion de génération automatique de code à partir de spécifications de plus haut niveau.

- Le langage cible n'est pas du code machine executable
- L'écriture d'un programme $\Rightarrow$ codage de $\neq$ types de connaissances

Génération de code

La plupart des derniers progrès dans le développement logiciel sont liés à la notion de génération automatique de code à partir de spécifications de plus haut niveau.

- Le langage cible n'est pas du code machine executable
- L'écriture d'un programme $\Rightarrow$ codage de $\neq$ types de connaissances

Mais qu'est-ce que XML et XSLT ont à voir avec cela ?

Génération de code

La plupart des derniers progrès dans le développement logiciel sont liés à la notion de génération automatique de code à partir de spécifications de plus haut niveau.

- Le langage cible n'est pas du code machine executable
- L'écriture d'un programme $\Rightarrow$ codage de $\neq$ types de connaissances

$\Rightarrow$ Spécifier les connaissances de l'appli. en XML, ce qui permet :

- de générer le code de l'appli. dans différents langages cibles,
- de générer la documentation,
- et éventuellement de produire des données de test.

Génération de code

La plupart des derniers progrès dans le développement logiciel sont liés à la notion de génération automatique de code à partir de spécifications de plus haut niveau.

- Le langage cible n'est pas du code machine executable
- L'écriture d'un programme $\Rightarrow$ codage de $\neq$ types de connaissances

$\Rightarrow$ Spécifier les connaissances de l'appli. en XML, ce qui permet :

- de générer le code de l'appli. dans différents langages cibles,
- de générer la documentation,
- et éventuellement de produire des données de test.

Mais bien entendu ce n'est pas *gratuit*