**INRIA Lorraine**
**Equipe PROTHEO**

# TOM & XQuery

Etudiant: Phan Nghiem Long
Tuteur: Pierre-Etienne Moreau

# Objective

- TOM devient un choix pour les developpeurs qui utilisent XQuery.
  - Implémenter les use cases de XQuery
  - Trouver les manques de TOM
  - Implémenter un translateur:
    XQuery ->  JAVA & TOM

# XQuery

- Un langage proposé par W3C pour traiter les documents XML
- Basé sur XPath
- Un langage fonctionnel
  - Pouvoir accéder aux éléments dans les documents, contruire les nouveaux éléments, mais ne pas pouvoir changer les documents
  - Chaque expression est une requête
  - Les requêtes "nested" sont permises
- Modèle de donnée basée sur le modèle abstrait de donnée XML
  - Elément: valeur ou noeud XML
  - Séquences des éléments

# Modèle de donnée

- Litéraux:
  - "string", int, real, ...
- Séquence:
  - Est une liste des éléments (avec l'ordre)
  - Pouvoir contenir les valeurs hétérogènes
    - Ex: (1, "string", 1.2)
  - Séquence vide: ()
  - Pas de séquence encastrée
    - Ex: (1, (2, 2), 1)  ==> (1, 2, 2, 1)
  - Un singleton est une séquence avec 1 élément
- Noeud XML
  - Elément (XML), text, attribut, document , PI, ...
  - L'order de document

# Expressions

- Opérateurs arithmetiques: +, -, *, div, mod
- Opérateurs comparatifs: =, <, >, <=, >=, !=
- Opérateurs booléens: and, or, not
- Opérateur "concat": ","
- Opérateurs pour les séquences: union, intersect, except
- If-then-else
- Construction des éléments XML
- Expression XPath
- Expression FLWOR (flower)

# Expression FLWOR

- For Let Where Order Return - FLWOR
- Similaire à des requêtes Select-From-Where de SQL

```
for $b in document("lib.xml")//book
where $b/author/name="ABC"
return
    $b/title
```

- Syntaxe:
  - **for** $var in expr [**where** expr] [**order by** expr] **return** expr
  - **let** $var in expr [**where** expr] [**order by** expr] **return** expr
- Pouvoir contenir plusieurs clauses for/let

```
Let $x:=1 let $y=2
return
    $x+$y
```

# Sémantique de FLWOR

- `for` **`$x`** `in` **`expr`** `[where` **`predicat`**`]` `return` **`expression`**
  - **`Predicat`** et **`expression`** peuvent dépendre de la valeur de **`$x`**
  - Si **`expr`** a la valeur (v1, v2, ..., vn), alors
    - **`$x`** a la valeur v1, si **`predicat`** est *true*, evaluer **`expression`**
    - **`$x`** a la valeur v2, si **`predicat`** est *true*, evaluer **`expression`** **...**
  - Les valeurs de l'**`expression`** sont placées dans une séquence
- `Let` **`$x:=expr`** `return` **`expression`**
  - **`$x`** va avoir la valeur de l'**`expression`**

# Exemple XQuery (donnée)

- Fichier "bib.xml"

```
<!ELEMENT bib  (book* )>
<!ELEMENT book  (title,  (author+ | editor+ ), publisher,
  price )>
<!ATTLIST book  year CDATA  #REQUIRED >
<!ELEMENT author  (last, first )>
<!ELEMENT editor  (last, first, affiliation )>
<!ELEMENT title  (#PCDATA )>
<!ELEMENT last  (#PCDATA )>
<!ELEMENT first  (#PCDATA )>
<!ELEMENT affiliation  (#PCDATA )>
<!ELEMENT publisher  (#PCDATA )>
<!ELEMENT price  (#PCDATA )>
```

**exemple de donnée:**

```
<book year="1994">
    <title>TCP/IP Illustrated</title>
    <author><last>Stevens</last><first>W.</first></author>
    <publisher>Addison-Wesley</publisher>
    <price> 65.95</price>
</book>
```

# Exemple XQuery (requête)

**Requête:**

```
<bib>
    for $b in doc("http://bstore1.exemple.com/bib.xml")/
      bib/book
    Where
        $b/publisher="Addition-Wesley"
        and $b/@year>1992
    Return
        <book year="{$b/@year}">
            {$b/title}
        </book>
</bib>
```

**Résultat:**

```
<bib>
    <book year="1994">
        <title>TCP/IP Illustrated</title>
    </book>
</bib>
```

# TOM

- Outil qui donne la capabilité de faire les calculs symboliques dans les langages impératifs: C et Java
- Développer une application avec TOM:
  - Définir les types de donnée
  - Définir les opérateurs sur les types de donné définits
  - Après avoir contruire l'arbre des termes avec backquot "`", utiliser **`%match`** pour filtrer les termes.

**TOM supporte XML**

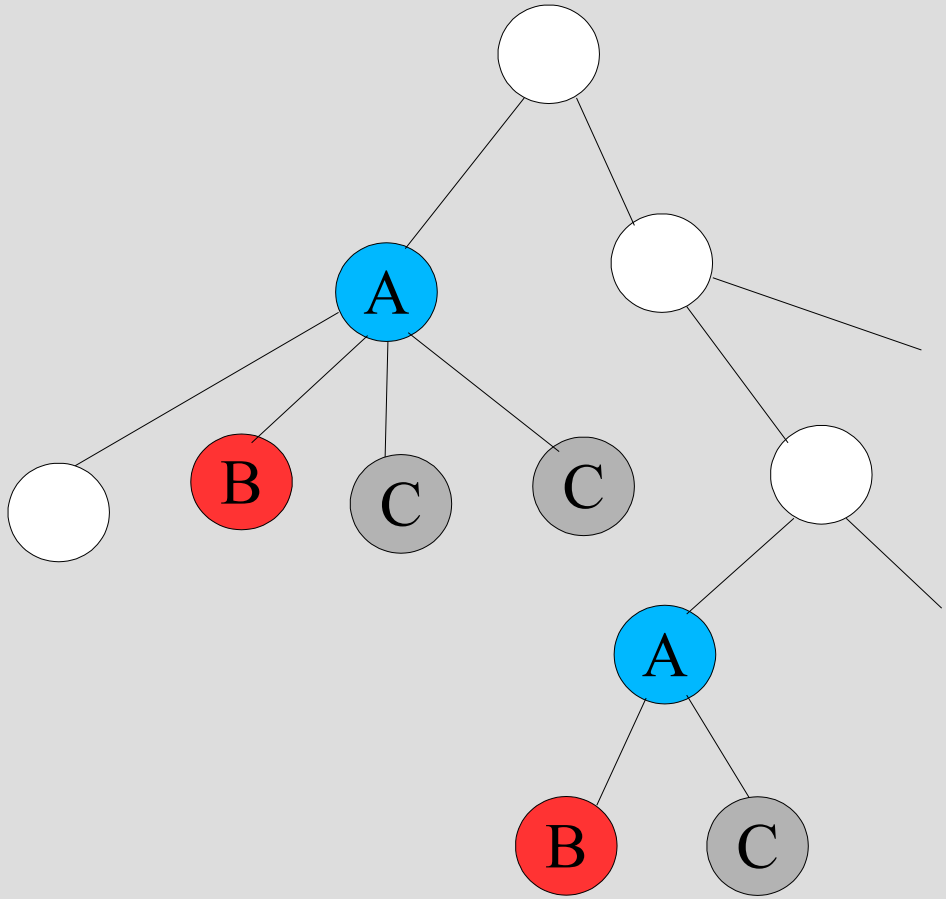
# Support XML de TOM

- Pouvoir utiliser la syntaxe de XML:

```
<A><B attribut1 = "string"/></A>
```

- Ex:

```
%match (Node a) {
   <A>
      <B/>
      <C/>
   </A>
   ->
   {
      // faire qqch
   }
   }
```



# Implémentation des use-cases

- Implémenter les use-cases XQuery en TOM et Java systématiquement.
- Comparer la performance avec celle de Galax (un engine XQuery de Bell lab)

# Comparaison de performance

| Use case | large | very large | huge |
|---|---|---|---|
| 1_1 | 0.9 (0.3) | 2 (2.0) | 4.6 (11.3) |
| 1_2 | 2.1 (0.5) | 6.1 (2.5) | 14.7 (12.6) |
| 1_3 | 2.0 (0.5) | 5.6 (2.4) | 13.5 (12.1) |
| 1_4 | 6.2 (41.7) | 29.9 (NA: > 15 m) | 2 m 46.7 (NA: >30 m) |
| 1_5 | 6.3 | 42.3 | Out of mem |
| 1_6 | 1.8 (0.4) | 4.9 (2.8) | 11.2 (14.6) |
| 1_7 | 1.0 (0.6) | 2.2 (1.3) | 5.3 (3.4) |
| 1_8 | 1.0 (0.3) | 2.2 (2.6) | 5.1 (13.0) |
| 1_9 | 1 | 1.9 | 4.9 |
| 1_10 | 7.9 | 2 m 25.4 | 14 m 55.0 |
| 1_11 | 2.5 (0.8) | 7.3 (4.0) | 18.0 (12.9) |
| 1_12 | 22.4 (2 m 12.7) | | |

- Noire: Java & TOM
- Green: Galax

(pen4M 2.2, 512 MBRAM, MandrakeLinux 10.0)

- **Donnée large**: 500 books, 200 authors, 200 editors, 200 publishers, 1000 – 3000 reviews, 800 – 1500 prices, 70% books have authors (with 1-3 authors each book), 30% books have editors (with 1-3 editors each book)
- **Donnée very large**: 2000 books, 800 authors, 800 editors, 200 publishers, 2000 – 4000 reviews, 3200 – 6000 prices, 70% books have authors (with 1-3 authors each book), 30% books have editors (with 1-3 editors each book)
- **Donnée huge**: 5000 books, 2000 authors, 2000 editors, 200 publishers, 5000 – 10000 reviews, 8000 – 15000 prices, 70% books have authors (with 1-3 authors each book), 30% books have editors (with 1-3 editors each book)

Données sont générées avec TOXGENE (http://www.cs.toronto.edu/tox/toxgene/)

# Expression XPath

- XPath permit de naviguer dans les documents XML
- Commencer à partir d'un noeud quelconque:

```
document(“bib.xml”)//book/chapter/title
$varname/chapter/title
```

- Se compose de plusieurs pas, les pas sont séparés par "/" ou "//" (opérateurs de chemin "/" et "//")
  - / (slash): chercher le sous-noeud direct
  - // (slashslash): chercher le sous-noeud, sans regardant le niveau encastré
- Retourne une séquence des noeuds qui satisfaient la chemin donnée

# Un pas de XPath

```
customer/child ::    name      [$i=100]
          {Axis} {nodetest} {prédicat}
-un pas-/----------un pas-------------
```

- Axis: la direction de travers
- Nodetest: Une description du noeud current
- Prédicat: Un ou plusieurs filtre pour diminuer les noeuds dans le résultat.

# Axis

- Axis: la direction de travers
  - Child: (default): travers vers les noeuds enfants
  - Parent: (abbr: ".."): travers vers le noeud parent
  - Self: (abbr "."): sans bouger :D
  - Attribute (abbr "@"): travers vers les attributs du noeud current

- Ex:

```
$var//parent::name
$var//@name
```

# NodeTest

- Nodetest: Une description du noeud souhaité
  - NodeKindTest: tester le sorte du noeud (document, élément, attribut, text, comment ....)
  - NodeNameTest: tester le nom du noeud
- Pouvoir utiliser le XML namespace dans le NodeNameTest:
  - Ex: `/tomnamespace:match`
- Pouvoir utiliser les wild-cards:
  - Ex:

```
/*:match
/tomnamespace:match
/*
```

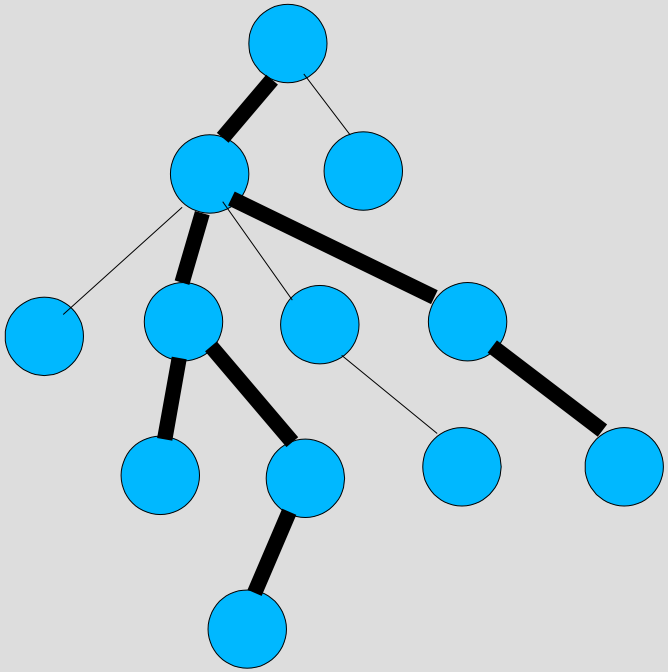
# Les prédicats

- Prédicat: Un ou plusieurs filtre pour diminuer les noeuds dans le résultat.
- Chaque prédicat est une expression
  - Prédicat booléen: true -> sélectionner ce noeud
    - Ex: `$var/section [$id = 100]`
  - Prédicat numérique: sélectionner seulement le noeud à la position spécifiée.
    - Ex: `$var/section [3]`
    - Prédicat numérique sans parenthèses
    - Prédicat numérique avec parenthèses

```
((($var//section)[3]/sub-section)[2]
```

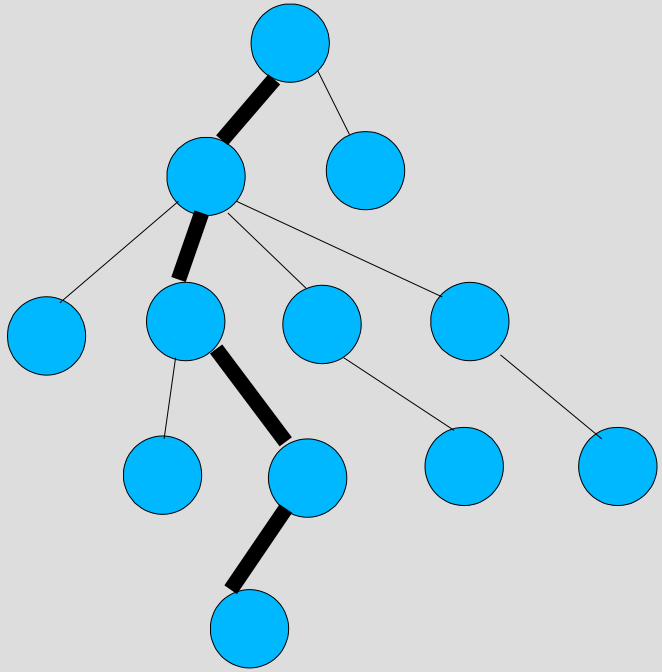
# Qu'est-ce TOM manque?

TOM



peut comparer un arbre total

XPath/XQuery



Spécialisé de naviguer dans le document

# Qu'est-ce TOM manque?

- Le souhait: écrire la chemin de XPath dans SEULEMENT UN partern XML dans %match
- TOM manque
  - Support de XML namespace
  - La possibilité de traverser vers le parent d'un noeud
  - Support de l'opérateur slashslash ("//")
  - Prédicat numérique: sans et avec parenthèses
  - Prédicat booléen

# Support de XML namespace

```
<loria:equipe><loria:total/><loria:equipe>
```

- DOM: déjà fournir des fonctions pour déterminer le namespace

  ==> pas de difficulté.

- Wildcard avec namespace
  - Normalement la signe "*" est le wildcard.
  - TOM actuel: "_"

    `<_> <B/> </_>`: n'importe quel noeud qui contient <B> comme le fils
  - Solution: ulilise "_" comme la notation de wildcard:

    ```
    <_:B> <_:B/>
    <namespace:_> <namespace:_/>
    ```

# Support de l'opérateur //

- //: chercher le noeud sans regardant le nombre de niveau encastré.
  - L'idée principale de TOM: un terme est une application de la racine à ses enfants directs
    f ( x, y) : application de f à x et y.

    => Difficile: il nous rester à dépenser du temps

- Utilisation alteractive: GenericTraversal de TOM

# Travers vers le parent

- TOM actuel

  F ( x, y) : application de F à x et y.

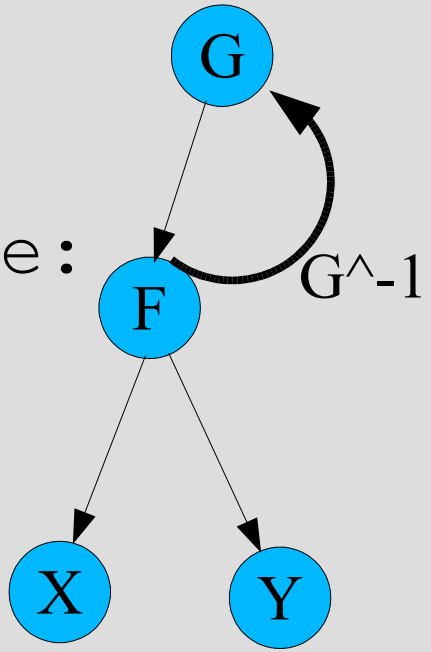
- Solution:

```
G (..., F (x, y), ...)
=> F peut être considéré comme:
F (G^-1, x, y)
```

  - Définition d'un terme dans TOM

```
%typeterm term {
   implement            { ATermAppl }
   get_fun_sym(t)       { t.getAFun() }
   cmp_fun_sym(t1,t2)   { t1 == t2 }
   get_subterm(t, n)    { t.getArgument(n) }

   set_parent_sym(t, parent) {t.setParent(parent);}
   get_parent_sym(t)         {t.getParent(); }
}
```



# Prédicat numérique

- La syntaxe de XML de TOM n'est pas convenable pour les prédicat numérique avec parenthèse.
  - Eliminer les partie innoutil pour XQuery
    `<A><B>~~</B></A>~~`
  - Ajouter les prédicats sans parenthèses:
    `<A><B>[1]`
  - Ajouter les prédicats avec parenthèses:
    `<A>(<B>[1])[2]`

**Devoir garder l'ancienne notation XML de TOM**

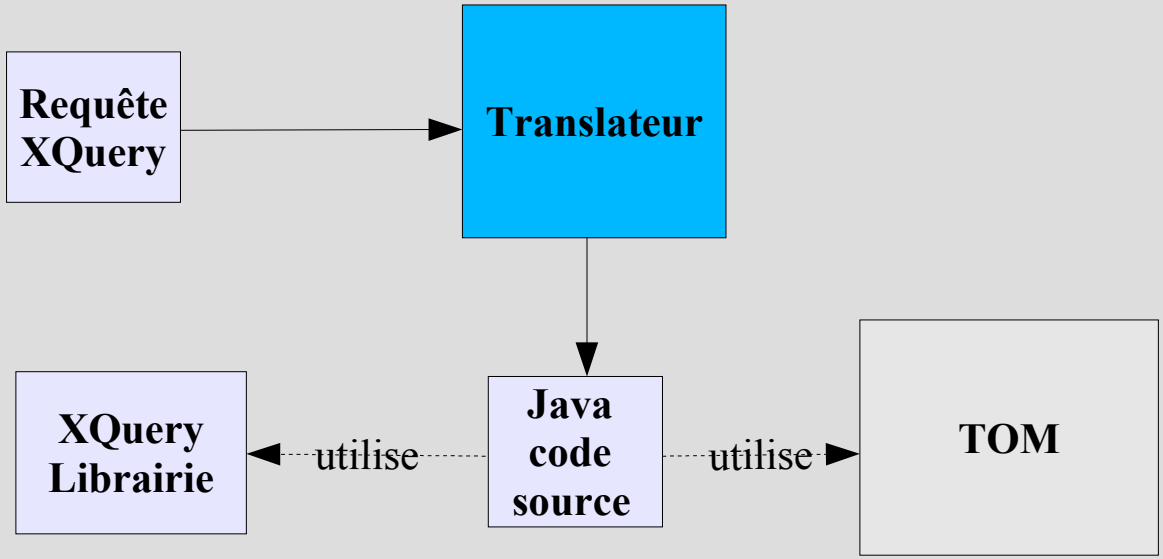
# Prédicat booléen

- Les prédicats peuvent être n'importe quelles expressions
- TOM fournie des statements "embeded" dans les autres langages de programmation.

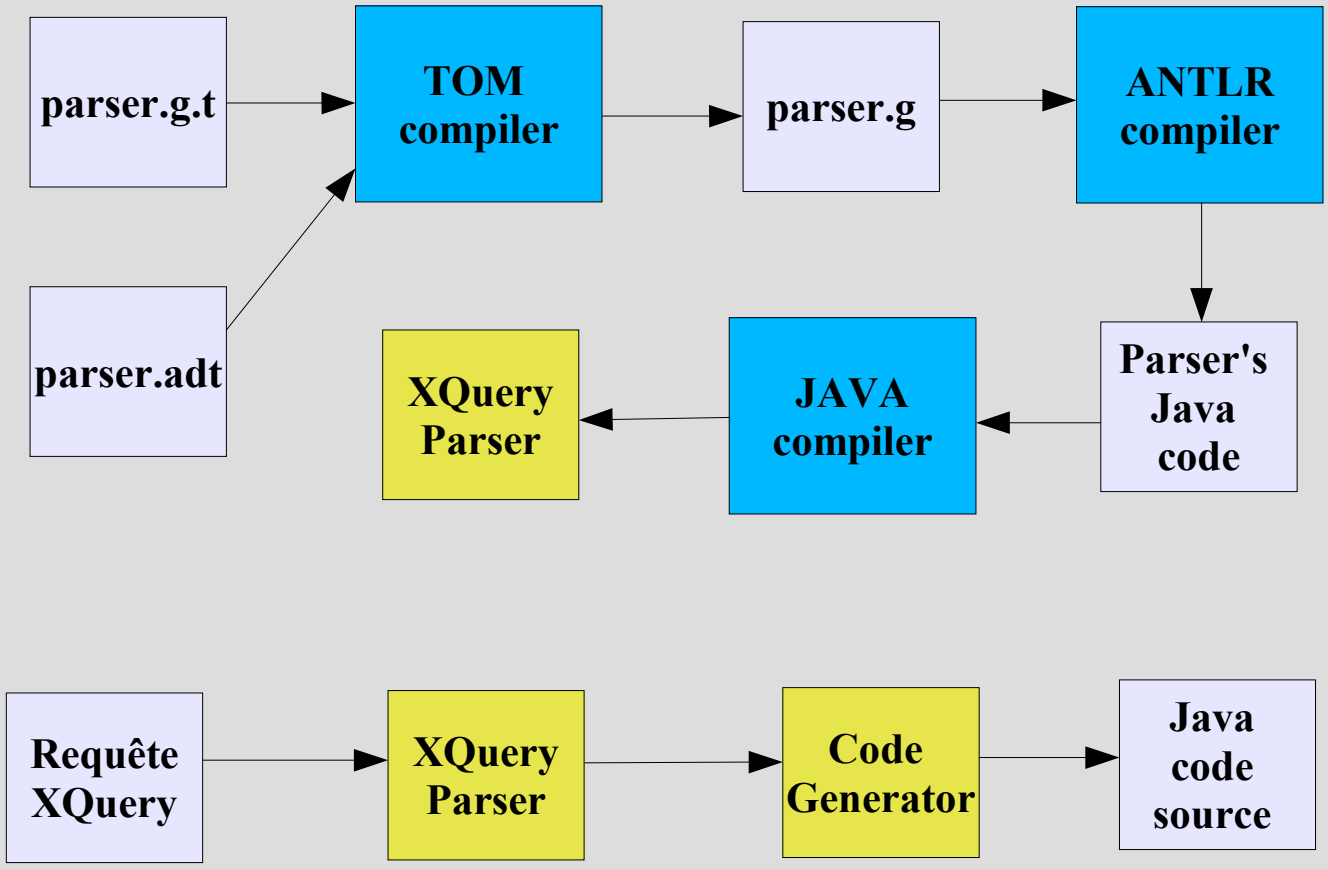
=== TROP difficile ===

# Le translateur XQuery

- But: transformer une requête vers un program Java avec TOM



# Le translateur (2)



# Conclusion

- TOM
  - unpeu loin de XQuery
  - assez rapide
- Pour le translateur
  - Pas efficace: c'est mieux si on développe une application qui éxécute directement les requêtes XQuery

# Perspective

- Pour le translateur:
  - Développer le fichier parser.t avec TOM, pour construire l'arbre AST
  - Développer le générateur.
  - Compléter la librarie run-time. (Ajouter les fonctions prédéfinies pour XQuery)
- Pour TOM:
  - Réaliser les solutions proposées
  - Proposer un modèle mathématique pour l'opérateur "//".

# Merci pour votre attention

# Références

- TOM: http://tom.loria.fr
- ATerm:
- Toxgene: http://www.cs.toronto.edu/tox/toxgene/
- ANTLR: http://www.antlr.org/
- XQuery: http://www.w3.org/TR/2003/WD-xquery-20031112/
- Galax: