XQuery

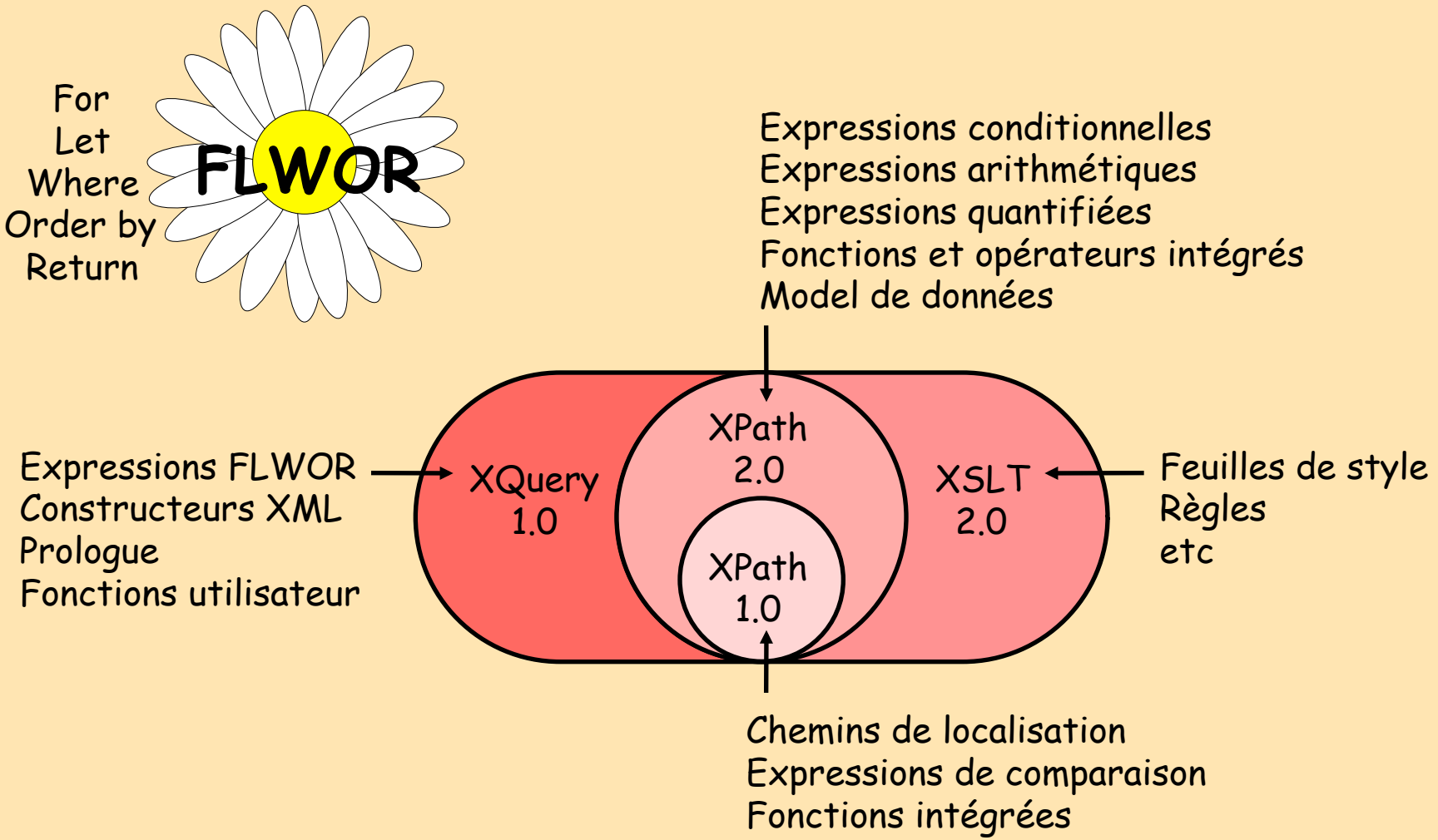
Philippe.Poulard@inria.fr

Sommaire

XQuery 1.0: An XML Query Language
W3C Recommendation 23 January 2007

http://www.w3.org/TR/xquery/

XQuery 1.0 and XPath 2.0 Functions and Operators



XPath/XQuery data model

Nodes

- Elements, attributs, autres composants XML

```
<size>225</size>
<project dept="production"/>
```

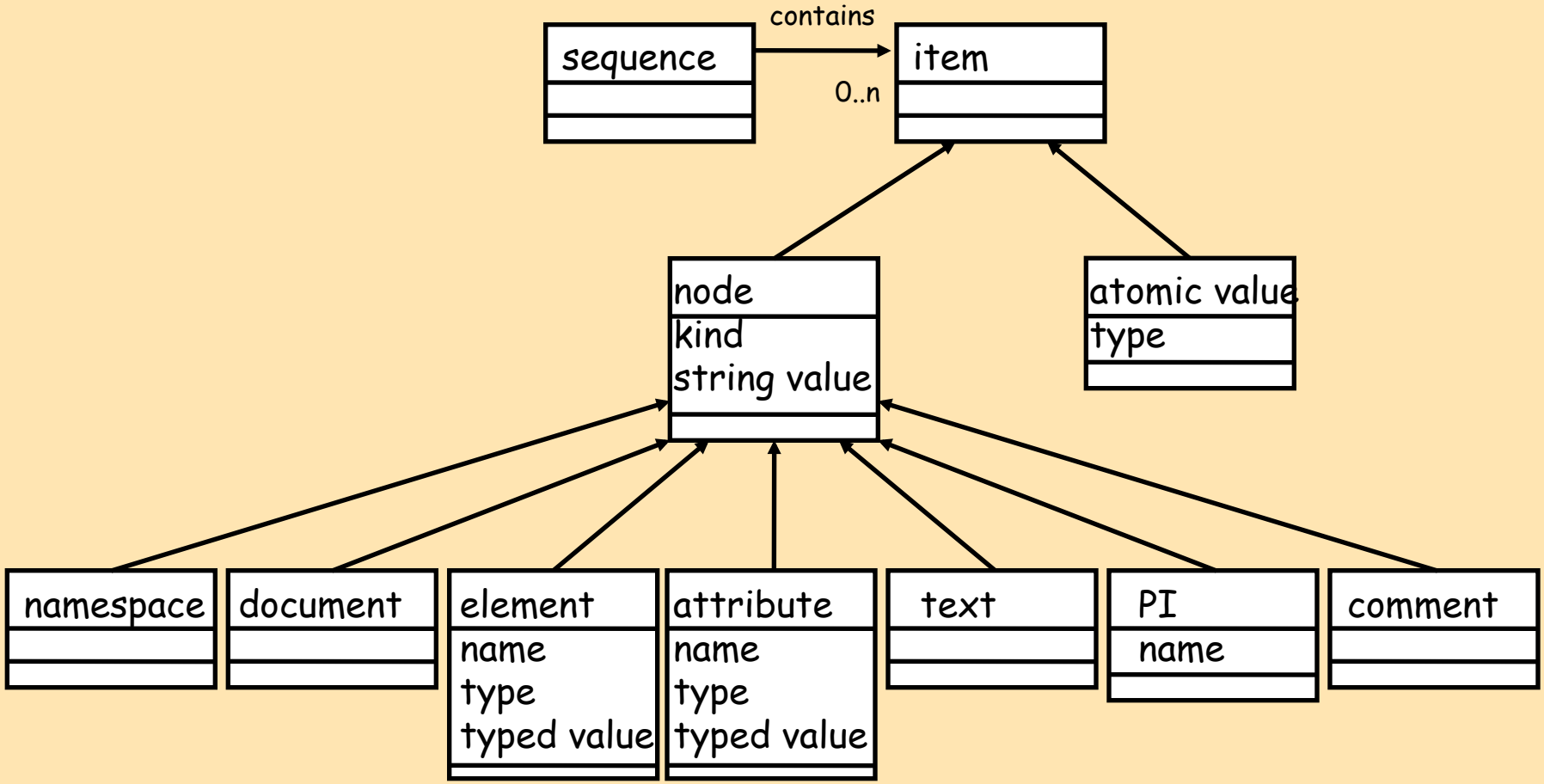
Atomic values

- Valeurs individuelles, ni élément ni attribut

```
225
"production"
```

Items

- Nodes
- Atomic values



Les nœuds :

- ont un "type" (node kind)
  - element, attribut, text, document, commentaire, instruction de traitement
- peuvent avoir un nom
  - `joueurs`, `org`
- ont une valeur textuelle (string value)
  - "`225`", "`FIFA`"
- peuvent avoir une valeur typée (typed value)
  - `xs:integer` 225, `xs:string` "FIFA"
- ont une identité unique

```
<size>225</size>
<project dept="production"/>
```

Valeurs individuelles
- sans relation avec un nœud particulier

Chaque valeur atomique a un type
- basé sur les types atomiques de W3C XML Schema
  - `xs:string`, `xs:integer`
- peut aussi être le type générique
  - `xs:untypedAtomic`
    - s'il n'y a pas de validation par un schéma

Création des valeurs atomiques

- Utilisation d'un littéral
  - `1234`, `"Hello world !"`
- Résultat de fonction
  - `count(//équipe)`
- Extraction explicite d'une valeur de nœud
  - `data(//joueurs[1])`
- Extraction implicite d'une valeur de nœud (atomisation)
  - `substring(//joueurs[1], 1, 2)`
- Utilisation d'un constructeur
  - `xs:date("1969-06-10")`

- Liste ordonnée de 0, 1 ou n items
- Une séquence d'un item est identique à l'item lui-même
- Une séquence d'aucun item est "la séquence vide"
  - différent de 0 et de ""
- Une séquence ne contient jamais une autre séquence

Création de séquences

- Résultat d'une expression qui retourne des nœuds
  - `équipes//joueurs`
    - retourne une séquence de nœuds `joueurs`
  - `équipes//pinguoins`
    - retourne une séquence vide
- Construction explicite
  - `(1, 2, 3)`
    - retourne une séquence de 3 entiers
  - `(1 to 3)`
    - retourne une séquence de 3 entiers
  - `()`
    - retourne la séquence vide

XPath 1.0

Nodes sets :
- ordonnés dans l'ordre du document
  - ou dans l'ordre inverse (selon l'axe)
- sans doublons

13 axes

Noms sans préfixes
- Sans espace de nommage

Fonctions contextuelles

Faible typage (adaptatif)

XPath 2.0

- Séquences : elles sont ordonnées
  - elles peuvent contenir des doublons
  - peuvent contenir aussi bien des valeurs atomiques que des nœuds
  - peuvent être dédoublonnées à l'aide de fonctions

12 axes
- l'axe `namespace::` disparaît

Possibilité de définir un espace de nom par défaut
- qui s'applique aux éléments sans préfixes

Les fonctions n'ont pas de contexte
- le contexte doit être transmis à la fonction

Typage fort (statique)

Comparaison

- par valeur : `eq, ne, lt, gt, ge`
  - chaque opérande doit être une aleur atomique ou un nœud ayant une seule valeur atomique
  - `product[number lt 500]`
- générale : `=, !=, <, <=, >, >=`
  - s'applique à des séquences ou des items
  - `product[number < 500]`

Tous les `product` qui ont un **unique** fils `number` dont la valeur est moins que 500
- **erreur** s'il y a plusieurs fils `number`

Tous les `product` qui ont **au moins un** fils `number` dont la valeur est moins que 500
- d'autres fils `number` peuvent avoir une valeur excédant 500

Prédicats

- peuvent contenir des expressions complexes
  - `chapter[if ($filter) then summary else true()]`
- s'appliquent aussi à des séquences
  - `(1 to 100)[. mod 5 = 0]`
- autres prédicats
  - `chapter[* except title]`

Chemins

- étapes utilisant des expressions à la place de tests de nœud
  - `animaux/(chats|chiens)`
  - `chapter/(if (summary) then summary else title)`
- la dernière étape peut retourner une valeur atomique
  - `book/chapter/title/substring(.,1,30)`

Comme dans XPath 1.0 :

- `$var`

Les variables apparaissent :

- dans les expressions FLWOR

```
for $project in doc("projects.xml")//project
return $project/name
```

- dans le prologue

```
declare variable $dept = "production";
```

- à l'extérieur de la requête par le processeurs

```
declare variable $dept as xs:string external;
```

- dans les signatures de fonction

```
declare function local:getProjectTeam(
    $doc as document(),
    $name as xs:string) as element()
{
    return $doc/project[name=$name]/team
};
```

Expressions alternatives

- Syntaxe `if-then-else`
- `()` autour de la condition
- Clause `else` obligatoire
  - peut être simplement `else ()`

```
if (exists($chapter/summary))
then summary
else title
```

Expressions itératives

- Version simplifiée des expressions FLWOR de XQuery
  - une seule clause `for`, pas de `let` ni de `where`

```
for $chapter in doc("book.xml")//chapter[@editable="yes"]
return substring($chapter/title,1,30)
```

Expressions quantifiées

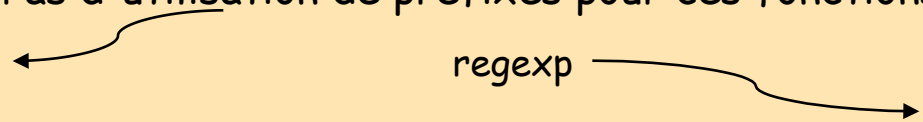
- Pour déterminer si quelques items ou tous les items d'une séquence satisfont un critère
  - `some … satisfies`
  - `every … satisfies`

```
some $editable in doc("book.xml")//chapter/@editable
satisfies ($editable = "yes")
```

```
every $editable in doc("book.xml")//chapter/@editable
satisfies ($editable = "yes")
```

Fonctions intégrées

- Plus de 100 fonctions XPath/XQuery
- Pas d'utilisation de préfixes pour ces fonctions



- Fonctions sur les chaînes :
  `substring, contains, matches, concat, normalize-space, tokenize`
- Fonctions sur les dates :
  `current-date, month-from-date, adjust-time-to-timezone`
- Fonctions sur les nombres :
  `round, avg, sum, ceiling`
- Fonctions sur les séquences :
  `index-of, insert-before, reverse, subsequence, distinct-values`
- Fonctions sur les nœuds :
  `data, empty, exists, id, idref`
- Fonctions sur les noms :
  `local-name, in-scope-prefixes, QName, resolve-QName`
- Rattrapage d'erreur et manipulation d'erreur :
  `error, trace, exactly-one`
- Fonctions sur les documents et les URIs :
  `collection, doc, root, base-uri`

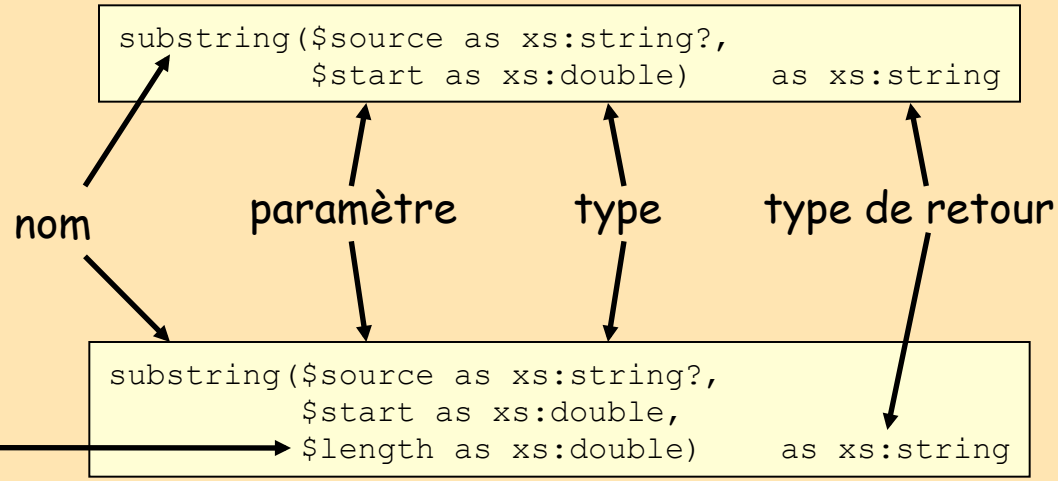
Fonctions utilisateur

- Peuvent être définies dans une requête XQuery ou une feuille de style XSLT
- Les noms de ces fonctions doit être préfixé

Signatures des fonctions

Chaque fonction a 1 ou plusieurs signatures

- qui définissent un nom, des paramètres et un type de retour



Séquences de types

- Les types de paramètres et les types de retour sont exprimées par des séquences de type
- Les plus courants sont de simples noms de types :
  `xs:string`, `xs:integer`, `xs:double`
- Autres types de séquence :
  `item()`, `node()`, `element()`, `attribute()`
- Indicateurs d'occurrence :
  `?`, `*`, `+`, ou rien

```
count($arg as item()*) as xs:integer
```

Doivent correspondre à l'une des signatures en nombre et type

```
substring($prodName, 1, 5)
substring($prodName, 1)
substring($prodName, 1, ())
substring($prodName, "1")
```

passer la séquence vide est différent d'omettre le paramètre

la chaîne "1" n'est pas l'entier 1

Listes d'arguments et séquences

La syntaxe des listes d'arguments est identique à celle des séquences

- à ne pas confondre

```
max($arg as xs:anyAtomicType*) as xs:anyAtomicType?
```

signature de `max`

```
max( (1, 2, 3) )
max( 1, 2, 3 )
```

Règles de conversion des types

Appliquées lorsque le type d'argument ne correspond pas au type dans la signature

Règles :

1. Les valeurs atomiques sont déduites
   - si une valeur atomique est attendue et qu'un nœud est reçu
   - la valeur atomique est extraite du nœud automatiquement
2. Les valeurs sans types sont transtypées (*cast*)
   - si un entier est attendu et qu'une valeur non typée est reçue
3. Les valeurs numériques sont promues
   - si un décimal est attendu et qu'un entier est reçu

Transtypage :

- Une chaîne ne peut être transtypée en entier
  - "1" ne donne pas 1
- Le contenu d'un nœud peut être extrait et transtypé

```
<elem>1</elem>
```

- peut-être de type entier
- peut-être non typé et transtypé en type entier

Exemple

```
xs:decimal?
```

⟵ type d'argument

Valeurs acceptées :

- une valeur atomique de `type xs:decimal`
- la séquence vide `()`
- une valeur atomique de type `xs:integer`
- la valeur atomique non typée `12.5`
- un nœud d'élément de type `xs:decimal`
- un nœud d'élement non typé dont le contenu est `12,5`

Modules de librairie

- Déclare un espace de nommage qui s'applique à toutes les fonctions et variables qu'il définit
- N'a pas de corps de requête
- Peut importer d'autres modules de librairies

```
module namespace proj = "http://www.example.com/projects";
declare function proj:count-projects($doc as document(),
                                     $dept as xs:string) as xs:integer {
    count($doc//project[@dept=$dept])
};
```

Import de modules

```
import module namespace proj = "http://www.example.com/projects"
    at "file:///path/to/module.xq";
<result>{ proj:count-projects("production") }</result>
```

```
for $dept in distinct-values(/project/@dept)
let $team := /project[@dept=$dept]/team
order by $dept
return <department name="{ $dept }">
           { $team }
       </department>
```

clauses `for` et `let` répétables

- Clause `for` :
  - à chaque itération, associe la variable `$dept` pour chaque item retourné par le path
- Clause `let` :
  - associe la variable `$team` au résultat du path

FLWOR expressions - Path expressions

Path expression

```
/project/team/person[@role="leader"]/name
```

Autre manière d'exprimer la même chose :

FLWOR expression

```
for $person in /project/team/person
where $person/@role="leader"
return $person/name
```

Objectifs de XQuery

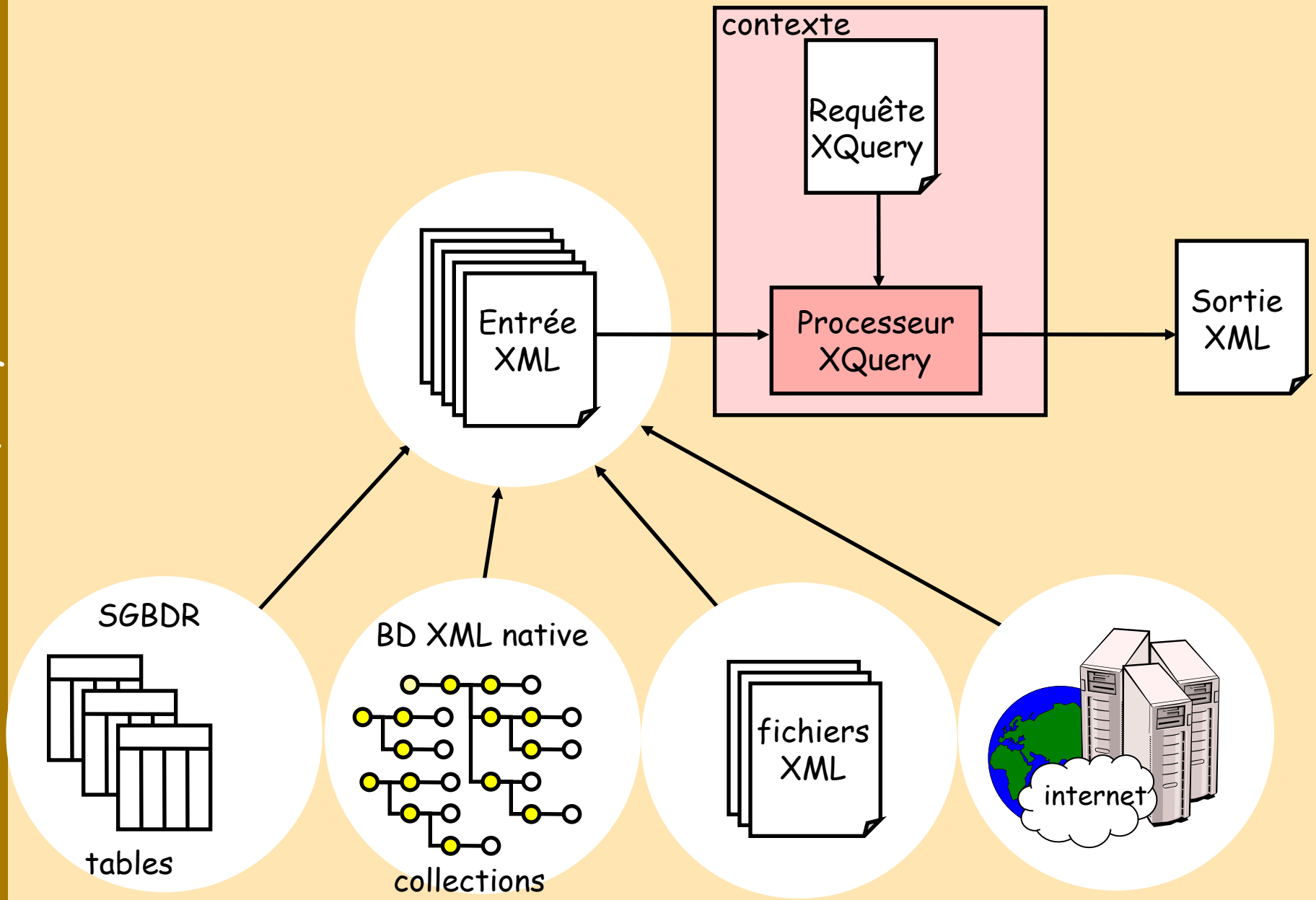
Un langage :

- utile à la fois pour interroger des données structurées ou non
- indépendant des protocoles
- déterministe
- déclaratif plutôt que procédural
- fortement typé
  (offre des possibilités d'optimisation et une meilleure détection des erreurs)
- capable d'accepter des collections de plusieurs documents
- compatible avec d'autres standards du W3C
  (Namespaces, W3C XML Schema, XPath 2.0)

Syntaxe XQuery

- Expressions imbriquées
- Compact, syntaxe non-XML
- Sensible à la casse
- Blancs :
  - autorisés (ignorés) entre les mots-clés du langage
  - considérés significatifs dans les chaînes entre quotes et les constructions d'élément
- Pas de charactère de fin de ligne particulier

Modèle de traitement de XQuery



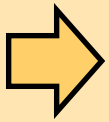
Tri

```
for $project in /project
where $project/@dept="production"
order by $project/name
return $project/name
```

```
<name>Increase profits</name>
<name>New products</name>
```

Entourer le résultat avec l'élément `ul`

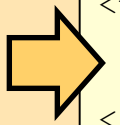
```
<ul>{
    for $project in /project
    where $project/@dept="production"
    order by $project/name
    return $project/name
}</ul>
```



```
<ul>
    <name>Increase profits</name>
    <name>New products</name>
</ul>
```

Entourer chaque `name` avec l'élément `li`

```
<ul type="square">{
    for $project in /project
where $project/@dept="production"
    order by $project/name
    return <li>{ $project/name }</li>
}</ul>
```

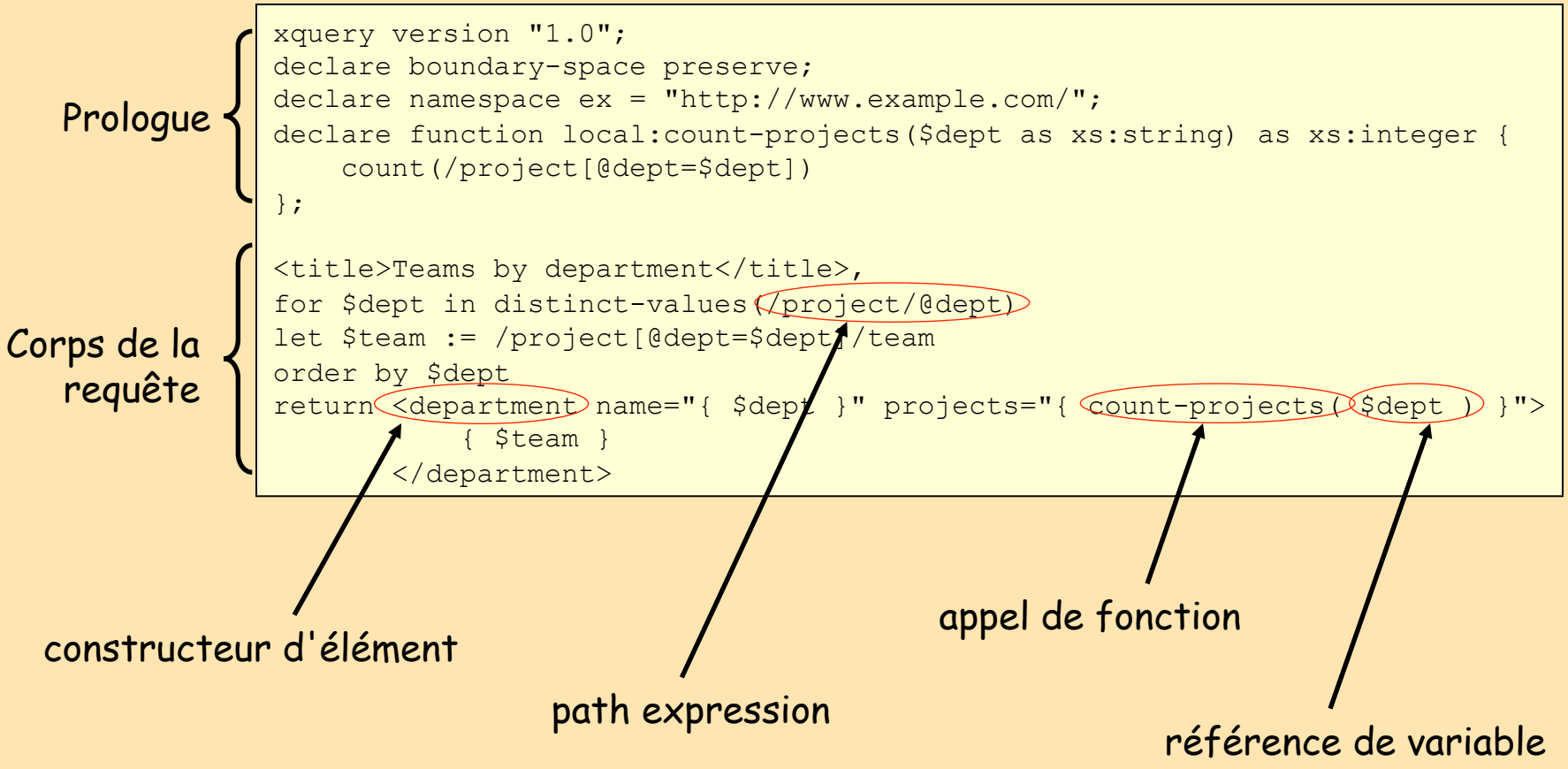


```
<ul type="square">
    <li><name>Increase profits</name></li>
    <li><name>New products</name></li>
</ul>
```

Eliminier l'élément `name`

```
<ul type="square">{
    for $project in /project
where $project/@dept="production"
    order by $project/name
    return <li>{ data( $project/name ) }</li>
}</ul>
```

```
<ul type="square">
    <li>Increase profits</li>
    <li>New products</li>
</ul>
```


Prologue
Corps de la requête
xquery version "1.0";
declare boundary-space preserve;
declare namespace ex = "http://www.example.com/";
declare function local:count-projects($dept as xs:string) as xs:integer {
count(/project[@dept=$dept])
};
<title>Teams by department</title>,
for $dept in distinct-values(/project/@dept)
let $team := /project[@dept=$dept]/team
order by $dept
return <department name="{ $dept }" projects="{ count-projects($dept) }">
{ $team }
</department>
constructeur d'élément
path expression
appel de fonction
référence de variable

Contient quelques directives :

- déclaration d'espaces de nommage

```
declare namespace ex = "http://www.example.com/";
```

- déclaration de fonctions

```
declare function local:count-projects() as xs:integer { … }
```

- déclaration de variables globales

```
declare variable $max := 12;
```

- déclaration de paramètres externes

```
declare variable $proj as xs:string external;
```

- import de modules externes

```
import module namespace ex = "http://www.example.com/"
                        at "../example-module.xq";
```

- import de schémas

```
import schema
    default element namespace "http://www/example.com/"
    at "http://www.example.com/path/to/example.xsd"
```

- collation par défaut

Ajout d'éléments et d'attributs

Inclusion à partir des documents d'entrée
En utilisant des constructeurs (syntaxe XML)
En utilisant des constructeurs calculés
- syntaxe spéciale { }
- permet de définir le nom d'un élément

```
for $project in /project
where $project/@dept="production"
order by $project/name
return $project/name
```

```
<name>Increase profits</name>
<name>New products</name>
```

Les noms des éléments sont inclus tels quels
Avec leurs attributs et leurs nœuds fils
Sans possibilité de changer ses attributs, nœuds fils, espaces de nom

```
<html>
  <body>
    <ul>{
      for $project in /project
      where $project/@dept="production"
      order by $project/name
      return $project/name
   }</ul>
  </body>
</html>
```

```
<html>
  <body>
    <ul type="square">
      <name>Increase profits</name>
      <name>New products</name>
    </ul>
  </body>
</html>
```

Syntaxe XML, peut contenir : littéraux, autres constructeurs d'éléments, expressions (entre crochets), un mélange des 3

- Apparaissent entre crochets { et }
- Peuvent être évaluées en :
  - Nœuds d'éléments
    - qui deviennent les fils de l'élément
  - Nœuds d'attributs
    - qui deviennent attributs de l'élément
  - Valeurs atomiques
    - qui deviennent du contenu textuel de l'élément
  - Une combinaison des 3

```
for $project in /project
where $project/@dept="production"
order by $project/name
return <li>{ $project/@dept }
                    { concat( "name", ": "  ) }
                              { $project/name }</li>
```

un nœud d'attribut devient un attribut

une valeur atomique devient un contenu textuel

un nœud d'élément devient un fils

```
<li dept="production">name: <name>Increase profits</name></li>
<li dept="production">name: <name>New products</name></li>
```

Constructeurs calculés

- Permet de calculer les noms et les valeurs
- Utile pour :
  - copier les éléments en entrée mais en opérant des changements mineurs
    - ex : ajouter un attribut `id` à tous les éléments
  - générer des attributs ou des éléments à partir du contenu
    - ex : créer un élément dont le nom est la valeur de l'attribut `dept`
  - rechercher les noms d'éléments et d'attributs dans un dictionnaire
    - ex : pour les traductions

```
element project {
    attribute dept { "production" },
    element { concat("na","me") } {
        "Increase profits"
    }
}
```

```
<project dept="production">
    <name>Increase profits</name>
</project>
```

```
for $dept in /project/@dept
order by $dept
return element { $dept } {
    /project[@dept=$dept]/name
}
```

```
<marketing>
    <name>More sales</name>
</marketing>
<production>
    <name>Increase profits</name>
    <name>New products</name>
</production>
<sales>
    <name>Cost reduction</name>
</sales>
```

```
doc("projects.xml")
```

```
doc("/path/to/projects.xml")
```

```
doc("file:///path/to/projects.xml")
```

```
doc("http://www.example.com/path/to/projects.xml")
```

```
collection("/path/to/coll/")
```

- Retourne une séquence de documents
- Peut être utilisé comme contexte pour un path

Contexte d'un path

- Celui spécifié (un document, une collection, un nœud, une variable…)
- "/" part de l'ensemble des éléments racines de la base de documents

```
for $dept at $i in /project/@dept
```

```
for $i in (1, 2), $j in (11, 12)
```

Comparaison dans l'ordre du document

```
$detp << $proj/@dept
```

```
$detp >> $proj/@dept
```

Combinaisons de séquences

```
($s1, $s2)
```
les doublons et l'ordre sont préservés

```
distinct-values(…)
```
pour dédoublonner

```
$s1 union $s2
$s1 | $s2
$s1 intersect $s2
$s1 except $s2
```

Espaces de noms

- Déclaration dans le constructeur d'élément
- Déclaration dans le prologue :
  - portée dans le corps de la requête
  - si c'est l'espace de nommage par défaut :
    - il s'applique aux éléments (pas comme XPath 1.0) et aux types
    - il ne s'applique pas aux variables ni aux fonctions

Espaces de noms prédéfinis

Préfixe	URI d'espace ne nom	Usage
xml	http://www.w3.org/XML/1998/namespace	xml:lang, xml:space, etc
xs	http://www.w3.org/2001/XMLSchema	Types de données xs:integer, xs:string, xs:date…
xsi	http://www.w3.org/2001/XMLSchema-instance	xsi:type, xsi:nil
fn	http://www.w3.org/2005/xpath-functions	Fonctions prédéfinies
local	http://www.w3.org/2005/xquery-local-functions	Fonctions utilisateur locales

```
declare function local:count-projects($dept as xs:string) as xs:integer {
    count(/project[@dept=$dept])
};
```

XQuery est fortement typé

Types W3C XML Schema :

- Chaînes
  - `xs:string`, `xs:token`, `xs:language`
- Numériques
  - `xs:integer`, `xs:decimal`, `xs:float`
- Dates, horaire et durée
  - `xs:date`, `xs:time`, `xs:duration`, `xs:dayTimeDuration`, `xs:yearMonthDuration`
- XML 1.0
  - `xs:NMTOKEN`, `xs:ID`, `xs:ENTITY`
- Autres
  - `xs:boolean`, `xs:anyURI`, `xs:QName`
- Données non typées (pour les données sans schéma)
  - `xs:untyped` (pour les éléments), `xs:untypedAtomic` (pour les attributs et les valeurs atomiques) : sont transtypées si nécessaire
- Type atomique générique (pour les signatures de fonction)
  - `xs:anyAtomicType`

Constructeurs de type

- Chaque type atomique est doté d'une fonction constructeur
- `xs:date("1969-06-10")` crée un `xs:date` à partir d'un `xs:string`
- `xs:float(12)` crée un `xs:float` à partir d'un `xs:integer`

SQL/XML dans SQL:2003

Vue XML d'une table SQL

USERS		
ID	FIRSTNAME	LASTNAME
1234 5678	John Philippe	Doe Poulard

```
<USERS>
  <row>
    <ID>1234</ID>
    <FIRSTNAME>John</FIRSTNAME>
    <LASTNAME>Doe</LASTNAME>
  </row>
  <row>
    <ID>5678</ID>
    <FIRSTNAME>Philippe</FIRSTNAME>
    <LASTNAME>Poulard</LASTNAME>
  </row>
</USERS>
```

Génération automatique de code SQL à partir de requête XQuery

```
for $u in collection('USERS')/USERS
return concat($u/FIRSTNAME,$u/LASTNAME)
```

```
SELECT ALL
{fn CONCAT(nrm5."FIRSTNAME",nrm5."LASTNAME")} AS RACOL1
FROM
"PEPPINO"."USERS" nrm5
```

```
for $u in collection('USERS')/USERS
return substring-before($u/FIRSTNAME, 'lo')
```

```
SELECT ALL
(CASE WHEN
{fn LOCATE('lo',nrm5."FIRSTNAME")} > 0
THEN
{fn LEFT( nrm5."FIRSTNAME", {fn LOCATE('lo',nrm5."FIRSTNAME")} -1)}
ELSE
'' END) AS RACOL1
FROM
"PEPPINO"."USERS" nrm5
```

http://www.jcp.org/en/jsr/detail?id=225

Java API pour XQuery
JSR 255 : développement en cours

```
XQDataSource dataSource = ...
XQConnection connection = dataSource.getConnection();
XQExpression xqExpression = connection.createExpression();
FileReader fileReader = new FileReader("myXQuery.xq");
XQSequence xqSequence = xqExpression.executeQuery(fileReader);
System.out.println(xqSequence.getSequenceAsString());
```

ou

```
PreparedExpression preparedExpression = connection.prepareExpression( fileReader);
XQSequence xqSequence = preparedExpression.executeQuery();
```

Passage de paramètres :

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
factory.setNamespaceAware(true);
FileReader fileReader = new FileReader("projects.xml");
Document document = parser.parse(fileReader);
xqExpression.bindNode(new QName("document"), document);
xqExpression.bindVariable(new QName("dept"), "production"));
```

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xcl:active-sheet
  xmlns:io="http://ns.inria.org/active-tags/io"
  xmlns:xcl="http://ns.inria.org/active-tags/xcl">
    <io:request name="result" style="stream" type="XQueryService"
        connect="xmldb:provider://host:8888/db/"
        source="myXQuery.xq"/>
    <xcl:transform source="{ $result }" output="file:///path/to/result.xml"/>
</xcl:active-sheet>
```

Passage de paramètres :

```
    <io:request name="result" style="stream" type="XQueryService"
            connect="xmldb:provider://host:8888/db/"
            source="myXQuery.xq">
        <xcl:param name="dept" value="production"/>
    </io:request>
```

XQuery and XPath Full

http://www.w3.org/TR/xpath-full-text-10/

XQuery Update Facility

http://www.w3.org/TR/xquery-update-10/

XQuery Scripting Extension

http://www.w3.org/TR/xquery-sx-10/