Schémas XML, XPath, et autres...

Plan de ce cours

- **Rappels/approfondissement du cours précédent**
- Codage des caractères
- Espace de noms
- Schémas XML
- Les Outils de Développement
- XPath

En Résumé

```
<?xml version="1.0" standalone="no"?>
<?xml-stylesheet type="text/css"
                 href="style.css"?>
<!DOCTYPE root SYSTEM "definition.dtd" [
  <!ELEMENT root (title, entry*)>
  <!ENTITY  court "tititatotu">
  <!ENTITY  file SYSTEM "file.xml">
]>
```

déclaration

Info Traitement

DTD

entités

Prologue

```
<!-- XML Content -->
<root>
   <title id="a15">modèle</title>
   <entry>122 &lt; 124</entry>
   <entry>il m'a dit &court;</entry>

   ...

<root>
```

Element du document

Information de traitement

<?*target processing_instruction_data***?>**

- permet de transmettre des informations à une application particulière qui peut lire le document (plutôt que d'utiliser les commentaires).
  - par exemple pour indiquer au butineur comment afficher le document:

```
<?xml-stylesheet type="text/css"
                 href="events4.css"?>
```

  - Ou pour inclure du code ad-hoc …

```
<?php mysql_connect("www.db.org …",
          "name","pwd"); $result = … ?>
```

- Une information de traitement dont la cible commence par le mot clef xml- est réservée à une utilisation par une extension de XML

Attributs ou éléments?

- On doit souvent se poser la question de représenter une information par un élément ou par un attribut. On peut utiliser un élément si
  - la donnée est structurée
  - la donnée contient plusieurs lignes
  - la donnée change fréquemment
- On peut utiliser un attribut si
  - la donnée est simple, petite et change rarement
  - il y a un faible nombre de valeurs possible

Vocabulaires XML

- successeur de HTML (XHTML)
- formules mathématiques (MathML)
- présentation multimédia avec animations (SMIL)
- description du contenus des ressources (RDF)
- modélisation des structures chimiques (CML)
- commerce électronique (ebXML, cXML, ...)
- service vocaux (voiceXML)
- appareils avec écrans réduits, e.g., WAP phones (WML)
- images 2D vectorielles (SVG)
- ... (voir xml.coverpages.org/xmlApplications.html)

Plan de ce cours

- Rappels/approfondissement du cours précédent
- **Codage des caractères**
- Espace de noms
- Schémas XML
- Les Outils de Développement
- XPath

Codage des caractères

UCS : un jeu de caractères universel

- La norme ISO 10646 en accord avec l'Unicode définit un jeu de caractères universel : l'UCS (« Universal Character Set ») qui permet de représenter les caractères de toutes les langues actuelles mais aussi anciennes.
- Chaque caractère UCS est identifié par un code qui est un nombre représenté sur 4 octets (2^{32} - 1 positions).
- Les 65 536 premières positions de l'UCS (c.-à-d. les deux octets de poids faible) forment le BMP (« Basic Multilingual Plane ») et codent les jeux de caractères les plus courants (latin, grec, arabe, etc.). D'où deux codages :
  - UCS-4 : totalité de l'UCS,
  - UCS-2 : BMP.

Codages de transformation

- Plusieurs codages de transformation ont été définis :
  - UTF-8 : permet de coder les caractères de l'UCS en longueur variable en codant sur un octet les caractères ASCII qui sont les plus fréquents.
  - UTF-16 : permet d'inclure des caractères de l'UCS-4 dans une chaîne codée en UCS-2.

Déclaration du codage

- Toutes les applications XML doivent accepter les codages UTF-8 et UTF-16.
- D'autres codages peuvent être acceptés, tels que le codage ISO-8859-1 (« ISO-Latin »).
- Le codage des caractères d'une entité doit être déclaré dans la déclaration XML de cette entité, comme valeur de l'attribut « encoding ». S'il ne l'est pas, l'application considérera être en présence d'un codage UTF-8.

Plan du cours

- Rappels/approfondissement du cours précédent
- Codage des caractères
- **Espace de noms**
- Schémas XML
- Les Outils de Développement
- XPath

Espace de noms

- L'importation d'éléments ou d'attributs contenus dans des entités externes peut entraîner des conflits de noms. Ces conflits peuvent être évités en définissant des espaces de noms
- Un espace de noms est identifié de façon unique par une URI (Uniform Resource Identifier)
- Pour obtenir des noms uniques, il suffit de qualifier chaque nom par l'URI de l'espace de noms dont il provient. Le nom obtenu est appelé nom étendu.
- Pour simplifier l'écriture des noms étendus, on associe un préfixe (un nom XML) à chaque espace de noms
- <?xml version="1.0" encoding="ISO-8859-1" ?>
  <fact:facture xmlns:fact="http://www.domaine.com/facture">
  <fact:montant>10$</fact:montant>
  <fact:nom>Jean</fact:nom>
  </fact:facture>

Déclaration d'un espace de noms

- Pour déclarer un espace de noms (et son préfixe associé) dans le contexte d'un élément, il suffit d'insérer la déclaration suivante dans la balise ouvrante:

  `xmlns:`*`préfixe`*`="`*`URI de l'espace de noms`*`"`

- On peut déclarer un espace de noms par défaut par l'attribut :

  `xmlns="`*`URI de l'espace de noms`*`"`

  ou annuler toute déclaration par défaut

  `xmlns=""`

- La déclaration d'un espace de noms est visible dans l'élément la contenant et dans tous ses descendants à moins qu'un nouvel espace de même préfixe ou bien un nouvel espace par défaut ne soit déclaré.

Noms qualifiés

- Tout nom d'élément ou tout nom d'attribut qui n'est pas une déclaration d'espace de noms, est un nom qualifié ayant l'une des deux formes suivantes

préfixe:*nom-local*

nom-local

- Un nom qualifié préfixé appartient à l'espace de noms associé à ce préfixe dans l'élément englobant le plus imbriqué.
- Un nom qualifié non préfixé :
  - appartient à l'espace de noms par défaut déclaré dans le plus imbriqué des éléments contenant ce nom, s'il en existe un.
  - n'appartient pas à un espace de noms s'il n'existe pas de déclaration d'espace de noms par défaut dans les éléments le contenant.

Exemple d'espace de noms

- Supposons que l'URI `def/biblio.xml` contienne une DTD pour la description de références bibliographiques, on pourrait rajouter des références dans un commentaire de la manière suivante

```
<a xmlns:b="http://www.w3.org/biblio.xml">
... </a>
<comment xmlns:biblio="def/biblio.xml">
   <biblio:titre>Why P = NP</biblio:titre>
   <auteur>Jean<auteur>
</comment>
```

- Rappels/approfondissement du cours précédent
- Codage des caractères
- Espace de noms
- **Schémas XML**
- Les Outils de Développement
- XPath

XML Schéma

- Un schéma d'un document définit:
  - les éléments possibles dans le document
  - les attributs associés à ces éléments
  - la structure du document
  - les types de données
- Le schéma est spécifié en XML
  - pas de nouveau langage
  - balisage de déclaration
  - espaces de nom spécifiques xsd: ou xs:
- Présente de nombreux avantages
  - structures de données avec types de données
  - extensibilité par héritage et ouverture
  - analysable à partir d'un parseur XML standard

Objectifs des schémas

- Reprendre les acquis des DTD
  - Plus riche et complet que les DTD
- Permettre de typer les données
  - Eléments simples et complexes
  - Attributs simples
- Permettre de définir des contraintes
  - Existence, obligatoire, optionnel
  - Domaines, cardinalités, références
  - Patterns, ...
- S'intégrer à la galaxie XML
  - Espace de noms
  - Infoset (structure d'arbre logique)

Le modèle des schémas

- Déclaration des éléments et attributs
  - Nom
  - Typage similaire à l'orienté objet
- Spécification de types simples
  - Grande variété de types
- Génération de types complexes
  - Séquence (Sequence)
  - Choix (Choice)
  - Tas (All)

XML Schema : exemple

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:element name="commande" type="CommandeType"/>
<xsd:element name="commentaire" type="xsd:string"/>
<xsd:complexType name="CommandeType">
    <xsd:sequence>
        <xsd:element name="livrer" type="Adresse"/>
        <xsd:element name="facturer" type="Adresse"/>
        <xsd:element ref="commentaire" minOccurs="0"/>
        <xsd:element name="produits" type="ProduitType"/>
    </xsd:sequence>
    <xsd:attribute name="date_com" type="xsd:date"/>
</xsd:complexType>
```

```
<xsd:complexType name="ProduitType">
    <xsd:sequence>
        <xsd:element name="produit" minOccurs="0" maxOccurs="unbounded">
        <xsd:complexType>
                <xsd:sequence>
                        <xsd:element name="nom_prod" type="xsd:string"/>
                        <xsd:element name="quantite">
                                <xsd:simpleType> <xsd:restriction base="xsd:positiveInteger">
                                <xsd:maxExclusive value="100"/> </xsd:restriction>
                                </xsd:simpleType>
                        </xsd:element>
                        <xsd:element name="prix" type="xsd:decimal"/>
                        <xsd:element ref="commentaire" minOccurs="0"/>
                        <xsd:element name="date_livraison" type="xsd:date" minOccurs="0"/>
                </xsd:sequence>
                <xsd:attribute name="num_prod" type="xsd:positiveInteger" use="required"/>
        </xsd:complexType>
        </xsd:element>
    </xsd:sequence>
</xsd:complexType>
</xsd:schema>
```

Les types simples (1)

- string
  - Confirm this is electric
- normalizedString
  - Confirm this is electric
- token
  - Confirm this is electric
- byte
  - -1, 126
- unsignedByte
  - 0, 126
- base64Binary
  - GpM7
- hexBinary
  - 0FB7
- integer
  - -126789, -1, 0, 1, 126789
- positiveInteger
  - 1, 126789
- negativeInteger
  - -126789, -1
- nonNegativeInteger
  - 0, 1, 126789
- nonPositiveInteger
  - -126789, -1, 0
- int
  - -1, 126789675
- unsignedInt
  - 0, 1267896754

Les types simples (2)

- long
  - -1, 12678967543233
- unsignedLong
  - 0, 12678967543233
- short
  - -1, 12678
- unsignedShort
  - 0, 12678
- decimal
  - -1.23, 0, 123.4, 1000.00
- float
  - -INF, -1E4, -0, 0, 12.78E-2, 12, INF, NaN
- double
  - -INF, -1E4, -0, 0, 12.78E-2, 12, INF, NaN
- boolean
  - true, false 1, 0
- time
  - 13:20:00.000, 13:20:00.000-05:00
- dateTime
  - 1999-05-31T13:20:00.000-05:00
- duration
  - P1Y2M3DT10H30M12.3S
- date
  - 1999-05-31
- gMonth
  - --05--
- gYear
  - 1999

Les types simples (3)

- gYearMonth
  - 1999-02
- gDay
  - ---31
- gMonthDay
  - --05-31
- Name
  - shipTo
- QName
  - po:USAddress
- NCName
  - USAddress
- anyURI
  - http://www.example.com/,
  - http://www.example.com/doc.html#ID5
- language
  - en-GB, en-US, fr
- ID
  - "A212"
- IDREF
  - "A212"
- IDREFS
  - "A212" "B213"
- ENTITY
- ENTITIES
- NOTATION
- NMTOKEN, NMTOKENS
  - US
  - Brésil Canada Mexique

Commandes de base xsd:

- element : association d'un type à une balise
  - attributs name, type, ref, minOccurs, maxOccurs, ...
- attribute : association d'un type à un attribut
  - attributs name, type
- complexType : une composition de types
  - définit une agrégation d'éléments typés
  - nécessaire pour décrire des attributs
- simpleType : les multiples types de base
  - entier, réel, string, time, date, ID, IDREF, …,
  - extensibles par des contraintes

Les types complexes

- Définition d'objets complexes
  - <xsd:sequence> : collection ordonnée d'éléments typés
  - <xsd:choice>: choix entre éléments typés
  - <xsd:all> : tous les éléments peuvent être présents, dans n'importe quel ordre
- Les attributs se placent en dernier
- Exemple

```
<xsd:element name= "adresse" type= "AdresseFR"/>
  <xsd:complexType name="AdresseFR">
     <xsd:sequence>
        <xsd:element name="nom" type="xsd:string"/>
        <xsd:element name="rue" type="xsd:string"/>
        <xsd:element name="ville" type="xsd:string"/>
        <xsd:element name="CP" type="xsd:decimal"/>
      </xsd:sequence>
     <xsd:attribute name="pays" type="xsd:NMTOKEN" fixed="FR"/>
  </xsd:complexType>
```

Héritage de types

- Définition de sous-types par héritage
  - Par extension : ajout d'informations
  - Par restriction : ajout de contraintes

- Exemple :

```
<complexType name="AdressePays">
  <complexContent>
    <extension base="Adresse">
       <sequence>
         <element name="pays"     type="string"/>
       </sequence>
    </extension>
  </complexContent>
</complexType>
```

Portée des Noms

```
<xsd:element name="person">
   <xsd:complexType> ...
    <xsd:element name="nom">
          <xsd:complexType>
              <xsd:sequence>
                  <xsd:element name="firstname" type="xsd:string"/>
                  <xsd:element name="lastname" type="xsd:string"/>
              </xsd:sequence>
          </xsd:complexType>
    </xsd:element>
     ...
   </xsd:complexType>
</xsd:element>

<xsd:element name="product">
  <xsd:complexType> ...
     <xsd:element name="nom"   type="xsd:string"/>
  </xsd:complexType>
</xsd:element>
```

- L'élément nom n'a pas la même signification dans person et dans product
  => Ceci est impossible dans une DTD !!!

Les patterns

- Contraintes sur type simple prédéfini
- Utilisation d'expression régulières
  - Similaires à celles de Perl
- Exemple (3 chiffres puis deux majuscules)

```
<xsd:simpleType name="NumItem">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="\d{3}-[A-Z]{2}"/>
  </xsd:restriction>
</xsd:simpleType>
```

Réutilisation de déclarations

- Possibilité de référencer un élément plus global
  - <element ref="Nom" /> (ci-dessus)
  - La référence porte sur l'attribut « name » (i.e. le nom de l'élément)
  - Importe l'élément et son type

- Possibilité d'inclure les types associés à un espace de noms
  - <import nameSpace = "http:// ....
    schemaLocation = "http:// ..." />

- Possibilité d'étendre un schéma
  - <redefine schemaLocation="http:// ..."/>
    .... Extensions ...
    </redefine>

Contraintes sur les attributs

Comme pour les DTDs, on peut rendre un attribut :

- obligatoire : use=”required”
- ou optionnel : use=”optional”

On peut aussi lui donner une valeur par défaut :

default = “toto”

XML Schema : exemple

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:element name="commande" type="CommandeType"/>
<xsd:element name="commentaire" type="xsd:string"/>
<xsd:complexType name="CommandeType">
    <xsd:sequence>
        <xsd:element name="livrer" type="Adresse"/>
        <xsd:element name="facturer" type="Adresse"/>
        <xsd:element ref="commentaire" minOccurs="0"/>
        <xsd:element name="produits" type="ProduitType"/>
    </xsd:sequence>
    <xsd:attribute name="date_com" type="xsd:date"/>
</xsd:complexType>
```

```
<xsd:complexType name="ProduitType">
    <xsd:sequence>
        <xsd:element name="produit" minOccurs="0" maxOccurs="unbounded">
        <xsd:complexType>
                <xsd:sequence>
                        <xsd:element name="nom_prod" type="xsd:string"/>
                        <xsd:element name="quantite">
                                <xsd:simpleType> <xsd:restriction base="xsd:positiveInteger">
                                <xsd:maxExclusive value="100"/> </xsd:restriction>
                                </xsd:simpleType>
                        </xsd:element>
                        <xsd:element name="prix" type="xsd:decimal"/>
                        <xsd:element ref="commentaire" minOccurs="0"/>
                        <xsd:element name="date_livraison" type="xsd:date" minOccurs="0"/>
                </xsd:sequence>
                <xsd:attribute name="num_prod" type="xsd:positiveInteger" use="required"/>
        </xsd:complexType>
        </xsd:element>
    </xsd:sequence>
</xsd:complexType>
</xsd:schema>
```

Référence à un Schéma XML

Pour faire référence à un Schéma XML dans un document XML, il y a deux cas :

- Si le Schéma n'est pas lié à un espace de nom :

```
<balise_racine
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="mon_schema.xsd">
```

- Si le Schéma est lié à un espace de nom :

```
<balise_racine
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:SchemaLocation=
     "http:mon-espace-de-nom.com     mon_schema.xsd">
```

Plan du cours

- Rappels/approfondissement du cours précédent
- Codage des caractères
- Espace de noms
- Schémas XML
- **Les Outils de Développement**
- XPath

Comment concevoir DTD/Schema ?

- A la main
  - syntaxe complexe, devient illisible
- Interface graphique IDE
  - partir d'un fichier d'exemples
  - générer un premier schéma via l'outil
  - modifier le schéma graphiquement
- A partir de UML
  - décrire données avec UML
  - générer un modèle logique hiérarchique

Diagramme XML Spy

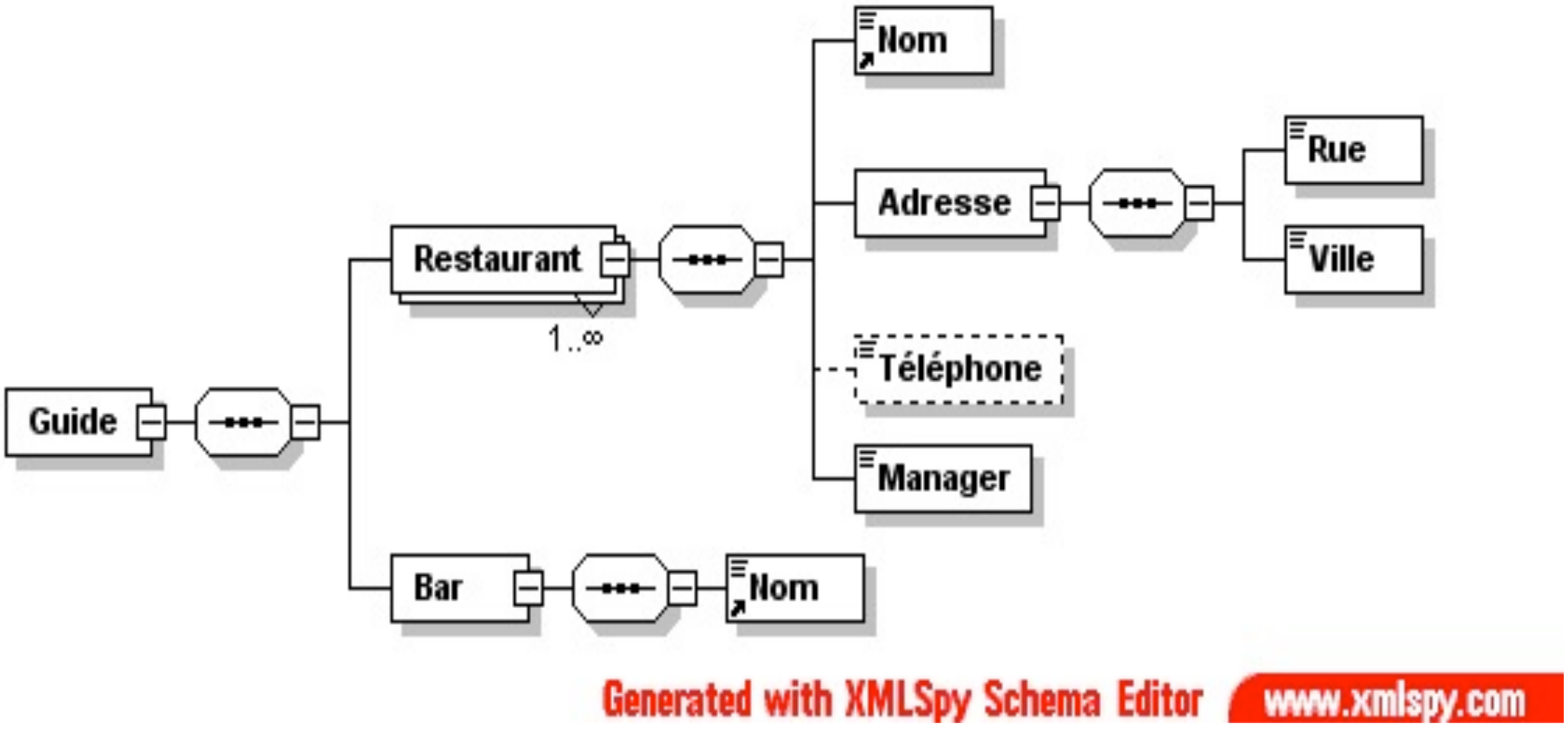
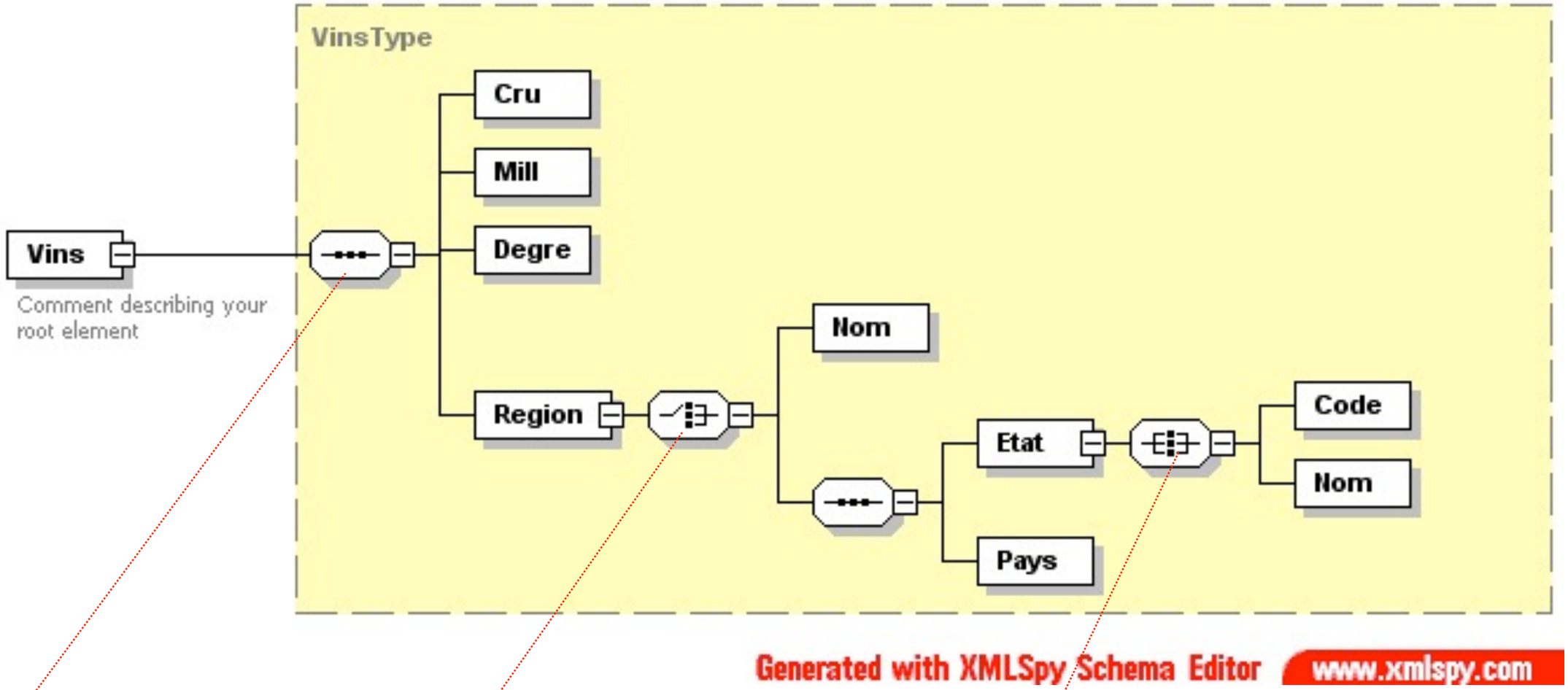


Diagramme de type (XML Spy)



Séquence

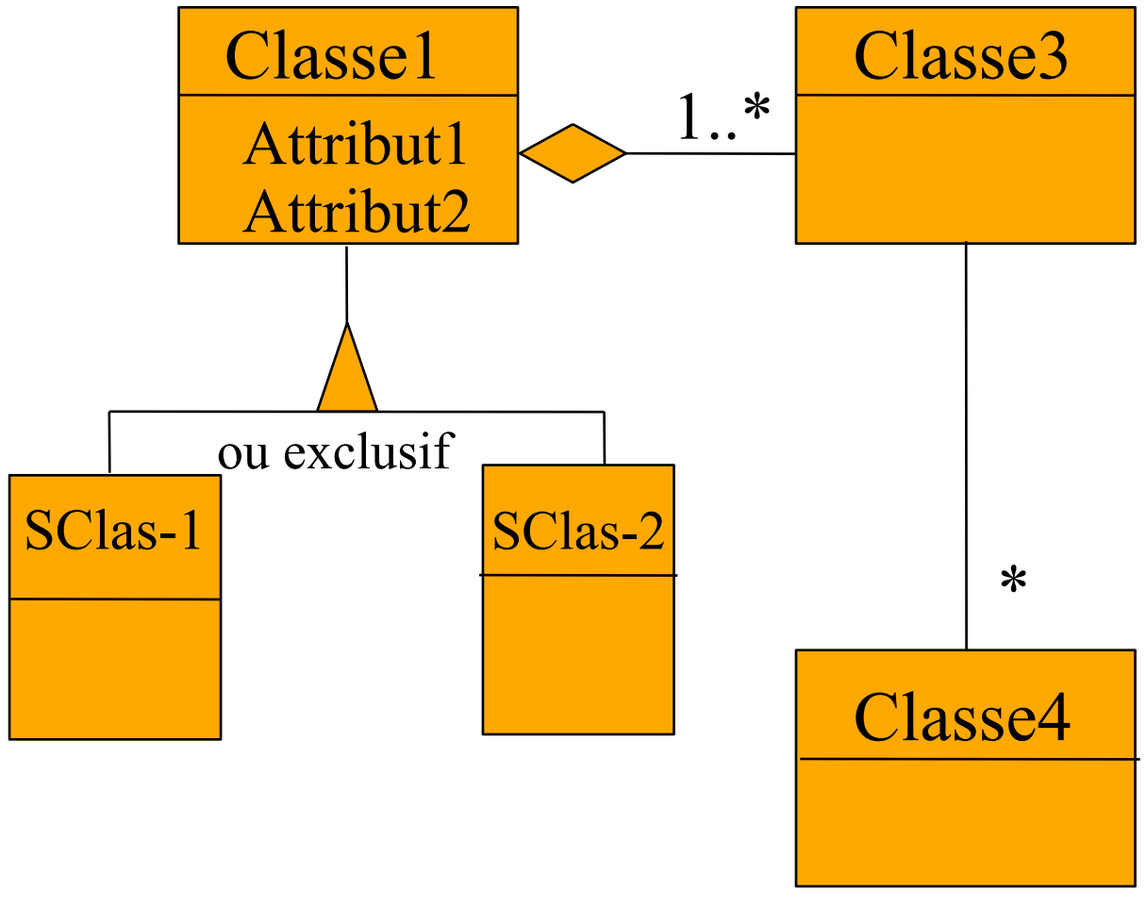
Choix

Tas

Rappels UML

- Modélisation de données et traitements
- Concepts pour les données
  - paquetage (package)
  - classe
  - attribut
  - association
  - agrégation
  - généralisation
- Contraintes
  - associations 0..* ou 1..* (*,+)
  - attributs avec nul possible (?)
  - généralisation exclusive

- Diagramme de classes



Conception UML de schémas

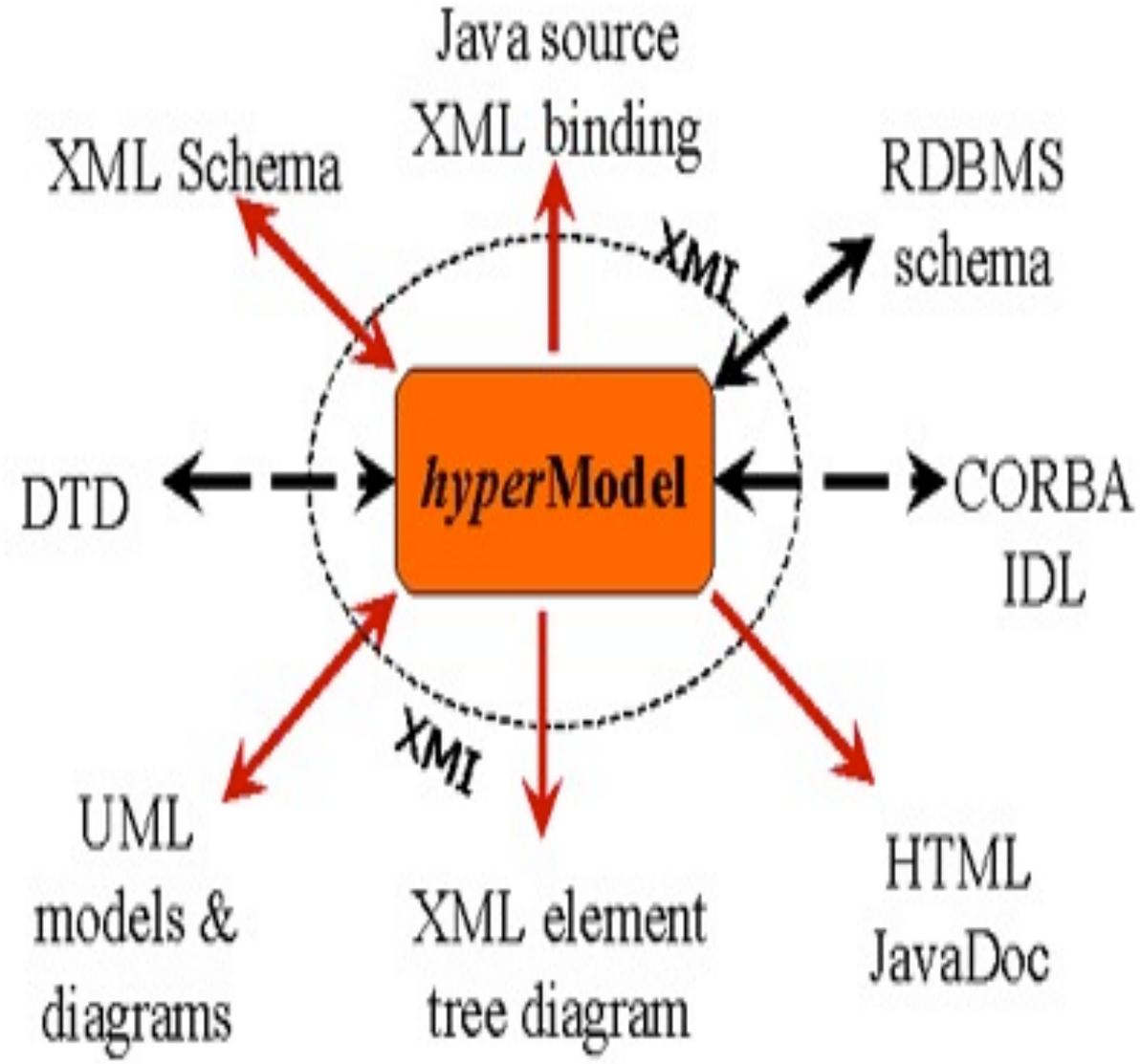
- Possible à partir d'une modélisation UML
  - Définir des paquetages de classes et associations
  - Orienter les associations ➔ arbre
  - Préciser les types de données
  - Possibilité d'hériter de stéréotypes XML
- Voir "Modeling XML Applications with UML"
  - David Carlson, A. Wesley
- Intégrer aux produits
  - Rose, Objecteering, Designer, HyperModel ...

De UML à XML : les choix

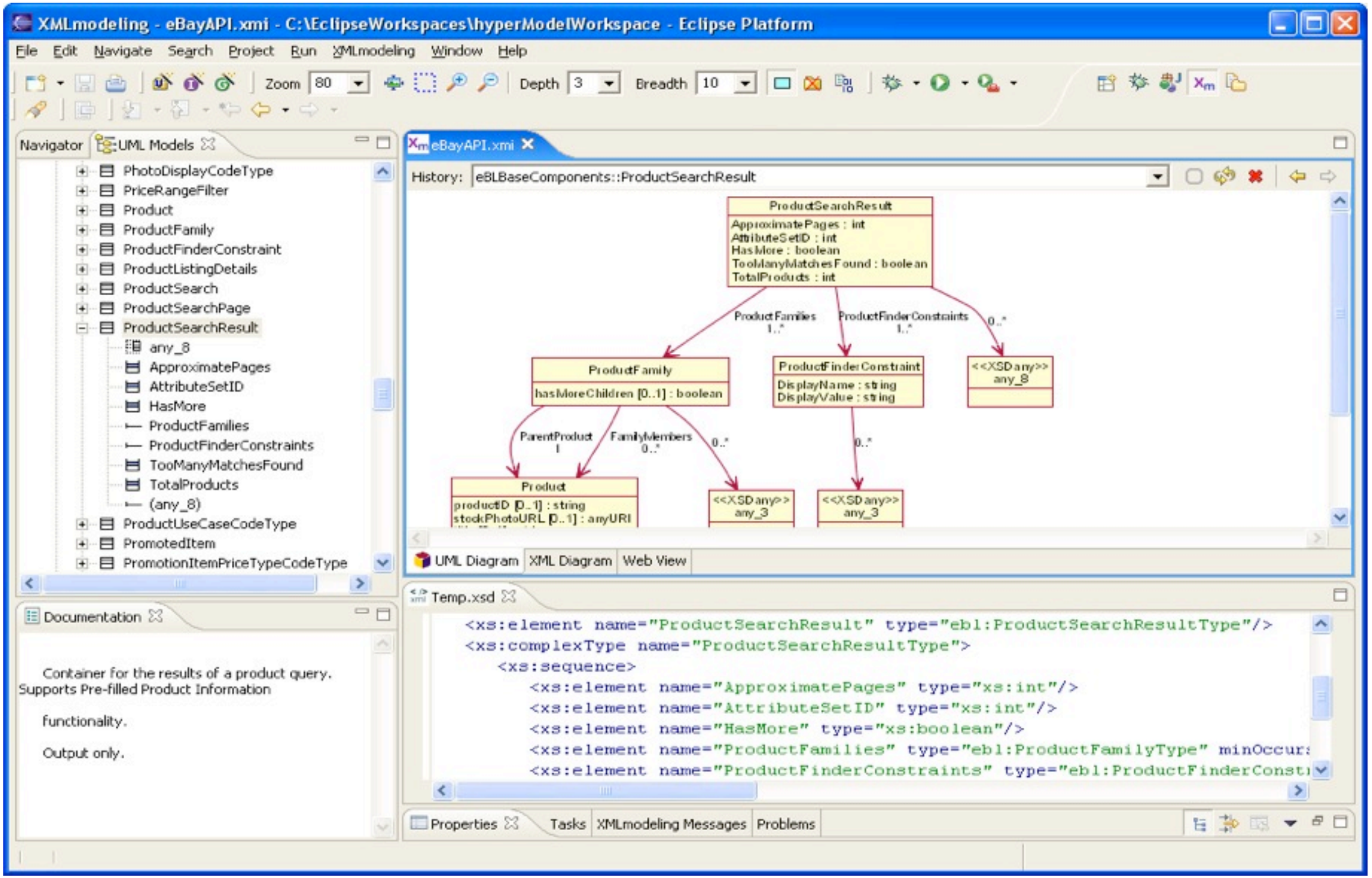
- Choix des messages (documents)
  - agrégation d'éléments
  - classes fortement liées
  - N instances
- Choix des identifiants et liens
  - références pour associations
- Attributs versus éléments
  - les attributs jouent un rôle descriptif ("méta")
- Choix des cardinalités
  - obligatoire ou optionnel (?)
  - 0..* (*) ou 1..* (+)

Produits: HyperModel

- De XML à UML
- et vice-versa
- Représentation XMI du modèle UML
- XMI = jargon XML de l'OMG pour modèle objet
- Traduction en toute sorte de modèle logique
- Ajoutable à Eclipse
- http://xmlmodeling.com



HyperModel et Eclipse



Les outils de développement

- IDE = Integrated Developement Environment
- Des éditeurs
  - De texte XML, parfois avec structure séparée
  - De schéma XML, avec interface graphique
  - De règles XSL, avec moteur de transformation
  - De requêtes XQuery, avec moteur sur document
  - De description WSDL, avec wrapper Web Service
- Des interfaces XML
  - Aux fichiers
  - Aux bases de données
  - Aux applications

Quelques outils de travail

Editeur	**Outil**	**Support**
Tibco	Turbo XML	DTD, XSL, XQuery Schéma
Altova	XMLSpy	DTD, Schéma XSL, XQuery
SyncRO Ltd.	Oxygen	DTD, Schéma XSL, XQuery
Data Junction	XML Junction	Schéma
Insight Soft.	XMLMate	DTD, Schéma, XSL, XPath
XML Mind	XMLMind Editor	DTD, Schéma, XSL, XPath

Plan du cours

- Rappels/approfondissement du cours précédent
- Codage des caractères
- Espace de noms
- Schémas XML
- Les Outils de Développement
- **XPath**

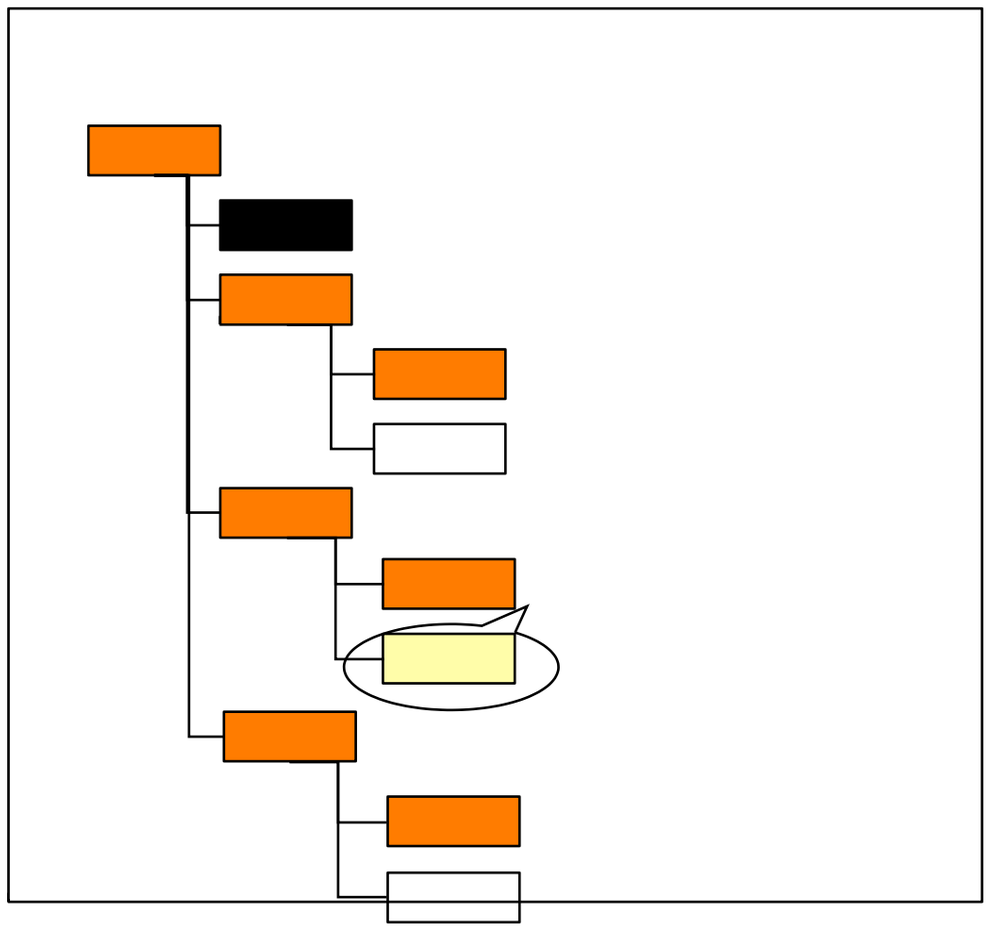
Le langage XPath:

une technologie XML pour le parcours de l'arbre d'un document

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

XPath : l'adressage XML

- XPath
  - Expressions de chemins dans un arbre XML
  - Permet de sélectionner des nœuds par navigation



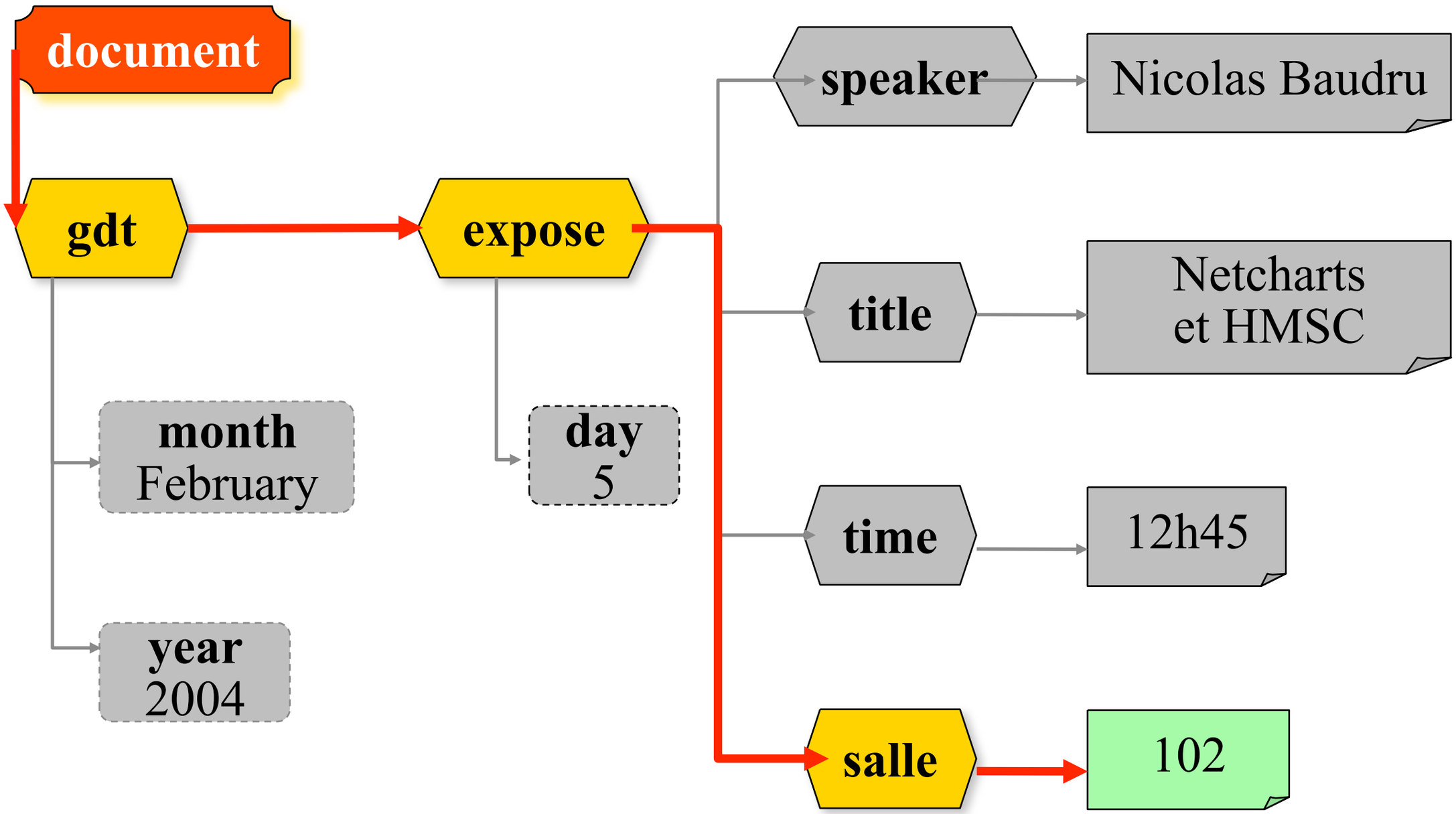
XML Path Language (XPath)

- XPath 1.0 est une recommandation du W3C (16/11/1999)
- Développé pour indiquer des ensembles d'éléments à transformer ou à interroger dans *XSLT* et *XQuery*
- C'est un langage déclaratif permettant de spécifier des « chemins » dans un arbre
  - XPath opère sur l'arbre d'un document.
  - Une expression permet de sélectionner, à partir d'un nœud donné, l'ensemble des nœuds accessibles en suivant tous les chemins conformes à un modèle appelé chemin de localisation
- La syntaxe est proche de celle utilisée pour les chemins d'accès dans les système de fichiers du type UNIX

XPath - Expression de chemins

- Une expression de chemins spécifie une traversée de l'arbre du document :
  - depuis un nœud de départ
  - vers un ensemble de nœuds cibles
  - les cibles constituent la valeur du cheminement
- Un chemin peut être :
  - absolu
    - commence à la racine
    - /étape1/…/étapeN
  - relatif
    - commence à un nœud courant
    - étape1/…/étapeN

Exemple: `//expose[@day<7]/salle/text()`



Expressions XPath - suite

- La valeur d'une séquence de pas de localisation est l'ensemble des nœuds « à l'extrémité des chemins » dénoté par cette séquence

- Une expression XPath permet de récupérer des ensemble de nœuds (node-set) mais également (par des fonctions de conversions) des:
  - chaînes de caractères
  - nombres flottant (conforme à la norme IEEE 754)
  - valeurs booléennes

Contexte d'une expression

- Le contexte d'une expression comprend
  - le nœud contexte
  - deux entiers : la position contexte et la taille contexte
  - un environnement (ensemble de liaisons variable/valeur)
  - une bibliothèque de fonctions prédéfinies
  - les déclarations d'espaces de noms visibles dans l'expression

Syntaxe et sémantique

- Cheminement élémentaire
  - axe::sélecteur [predicat]
- Directions
  - parent, ancestor, ancestor-or-self
  - child, descendant, descendant-or-self
  - preceding, preceding-sibling, following, following-sibling
  - self, attribute, namespace
- Sélecteur
  - nom de nœud sélectionné (élément ou @attribut)
- Prédicat
  - [Fonction(nœud) = valeur]

Expressions simplifiée

- Il existe trois « opérateurs simples » permettant de définir des pas de localisations
- étape vide: `//`

  récupère tout les descendants du noeud contexte
- fils élément: `element-name[predicates]`

  enfants « élément » du noeud contexte de nom `element-name` et vérifiant les conditions `predicates`
- fils attribut: `@attribute-name[predicates]`

  nœud attribut du noeud contexte de nom `attribute-name` et vérifiant les conditions `predicates` (optionnelles)

Exemples

- `//speaker`: tout les descendants du noeud document qui sont des nœuds éléments nommé `speaker`
- `/gdt/@month`: l'attribut `month` des éléments nommés `gdt` qui sont les fils du nœud document
- `/gdt/expose[comment]`: les éléments `expose` qui sont fils de `gdt` et qui ont un élément `comment` comme fils
- `/gdt[expose/comment]`: les éléments `gdt` qui contiennent un fils `expose` qui contiennent un élément `comment`
- `/gdt/expose[comment][time]`: les éléments `expose` qui contiennent un fils `comment` ET un fils `time`
- `/gdt/expose[position()=2]`: le deuxième élément `expose` fils de `gdt` à partir de la racine

Prédicats

- Les prédicats permettent de filtrer les nœuds obtenus à partir d'une expression
- Les prédicats sont formés à partir de
  - expressions booléennes: `and, or, not, =, ...`
  - expressions numériques: `+, -, ...`
  - expressions de node-set
  - fonctions de node-set: `position`(), `last`(), ...
- À partir du node-set S, obtenu à partir d'une expression XPath, on évalue le prédicat en prenant chacun des nœuds x de S comme contexte, la taille de S comme taille du contexte et la « position » de x dans S comme la position du nœud

Pas de localisation

- Un pas de localisation est caractérisé par :
  - un axe
  - un test de nœud
  - une suite éventuellement vide de prédicats

```
axe::node_test [pred_1] … [pred_n]
```

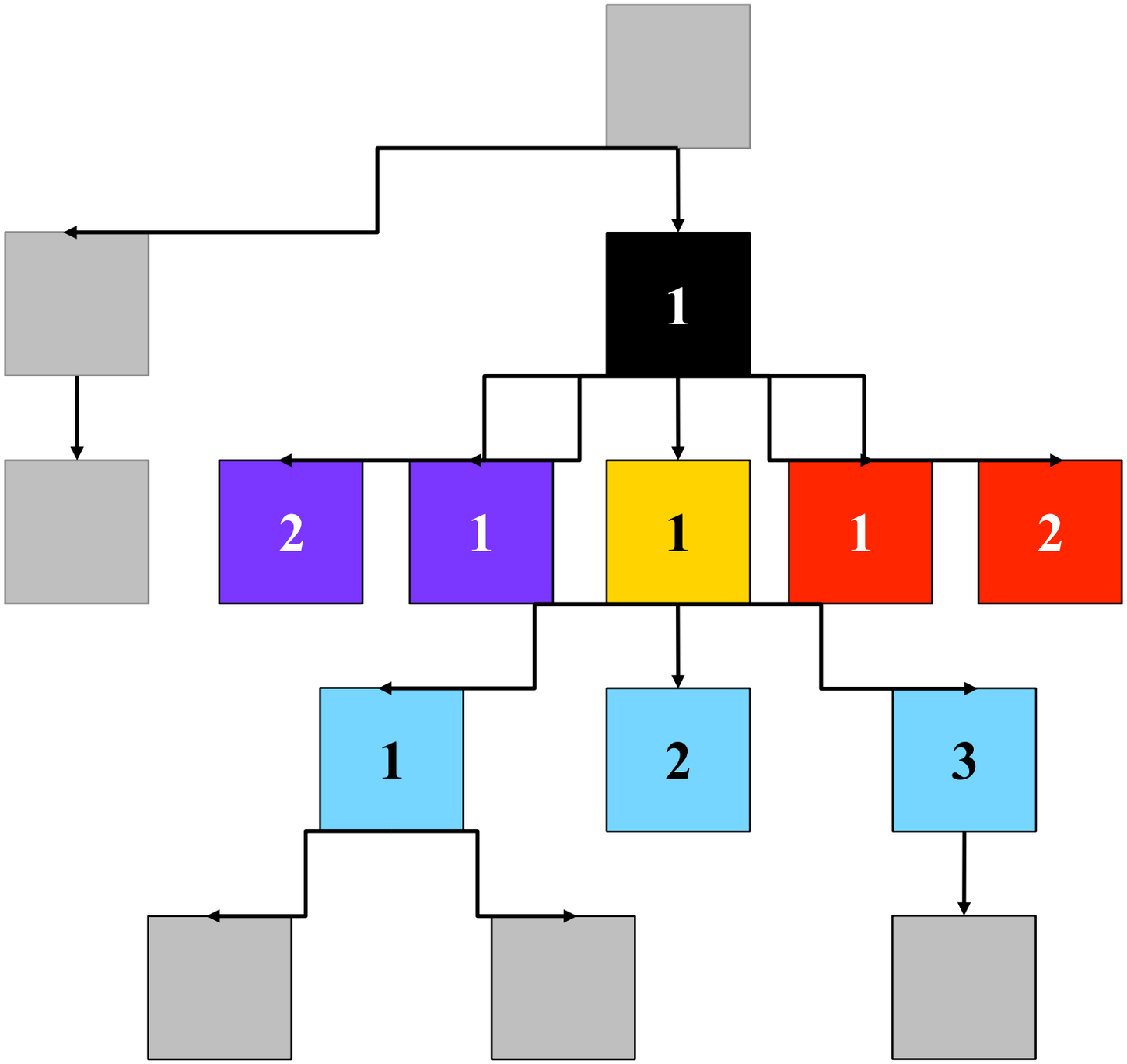
Axes (ou direction)

- Un axe spécifie quels sont les nœuds à considérer (quelle direction suivre), relativement au nœud contexte:
  - `self` : sélectionne le nœud contexte lui-même (comme avec `.`)
  - `child` : sélectionne les enfants du nœud contexte (comme avec `elmt_name`)
  - `descendant` : sélectionne les descendants du nœud contexte (comme avec `//`)
  - `parent` : sélectionne le père du nœud contexte (comme avec `..`). Le père du nœud document correspond au node-set vide.

Axes « cousins »

- `preceding-sibling` : sélectionne les frères précédents du nœud contexte
- `following-sibling` : sélectionne les frères suivants du nœud contexte (si le nœud contexte n'est pas un élément, l'axe est vide)
- `attribute` : sélectionne tout les nœuds attributs du nœud contexte (si le nœud contexte n'est pas de type élément, cet axe est vide)
- `namespace` : sélectionne tout les nœuds espaces de noms fils du nœud contexte (si le nœud contexte n'est pas un élément, cet axe est vide)

Axes



context (self)

parent

preceding-sibling

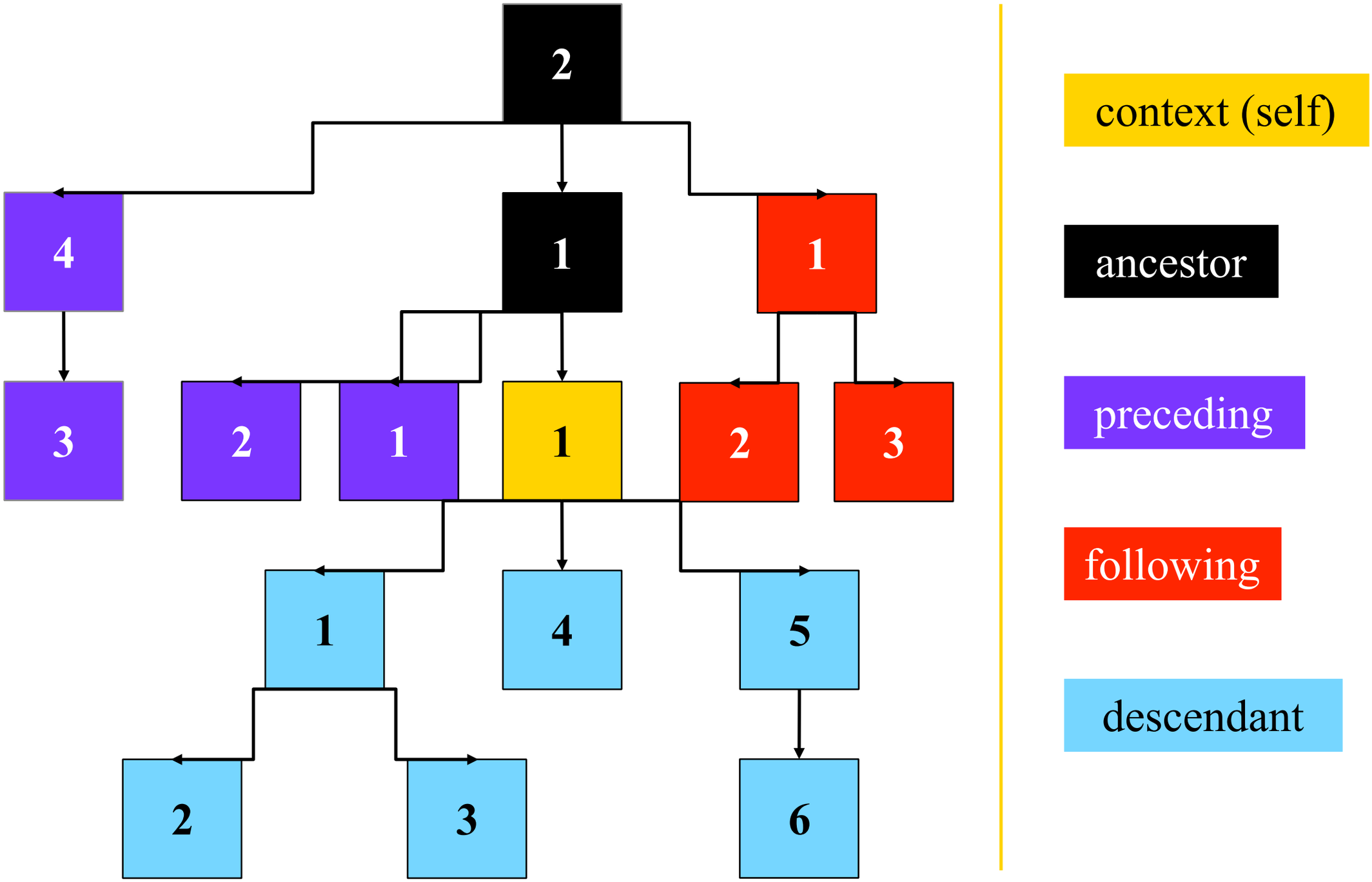
following-sibling

child

Axes - autres

- `ancestor` : sélectionne les ancêtres du nœud contexte
- `ancestor-or-self / descendant-or-self`
- `preceding` : sélectionne les nœuds qui sont avant le nœud contexte dans l'ordre du document mais pas ses descendants (et bien sur pas les nœuds attributs et espace de noms)
  - ce sont les élément dont le end-tag apparaît avant le start-tag du nœud contexte
- `following` : sélectionne les nœuds qui sont après le nœud contexte dans l'ordre du document …
  - ce sont les élément dont le start-tag apparaît après le end-tag du nœud contexte

Axes


2
4
1
1
3
2
1
1
2
3
1
4
5
2
3
6
context (self)
ancestor
preceding
following
descendant

Axes

- Un axe a un sens et un type
  - Un axe dont les nœuds sont self ou des nœuds qui suivent self dans l'ordre du document est un axe avant
  - Un axe dont les nœuds sont self ou des nœuds qui précèdent self est un axe arrière
  - Un axe a un type de nœud principal :
    - le type de nœud principal de l'axe `attribute` est « attribut »
    - le type de nœud principal de l'axe `namespace` est « espace de noms »
    - le type de nœud principal des autres axes est « element ».
- les axes `ancestor`, `ancestor-or-self`, `preceding` et `preceding-sibling` sont des axes arrières, les autres axes sont des axes avants.

Test de nœud (node_test)

- Les axes autres que `attribute` et `namespace` capturent des nœuds éléments, textes, commentaires, … dont le type principal est element. Un test de nœud permet de restreindre le node-set obtenus à partir d'une expression de type element
- Un test de nœud permet de restreindre selon le nom du noeud
  - `element_name`: sélectionne les nœuds de l'axe ayant le même type que le type principal de l'axe et dont le nom étendu est égal au nom étendu de n ;
  - `*` : sélectionne les nœuds de l'axe ayant le même type que le type principal de l'axe

Test de nœud - suite

- Le test peut aussi se faire sur le type du nœud:
- `node()` : sélectionne tous les nœuds (cas par défaut)
- `text()` : sélectionne les nœuds textes
- `comment()` : sélectionne les nœuds commentaires
- `processing-instruction(target)` : sélectionne les nœuds de l'axe représentant une instruction de traitement (proc. inst.) de nom `target`.

Exemples

- `child::element()[position()=2]`: second fils élément du noeud contexte
- `descendant::node()`: tout les nœuds descendants, c'est-à-dire les nœuds éléments, textes, commentaires et proc. inst.
- `following-sibling::*[position()=last()]`: le dernier des cousins de type élément
- `child::expose[position()=2]/child::commentaire[position()=1]`: le premier commentaire du second expose

Prédicat et filtrage

- Un prédicat est destiné à filtrer un ensemble de nœuds. Il est composé d'une expression booléenne appelée expression du prédicat.

- Le filtrage par un prédicat est réalisé relativement à un axe dit axe de filtrage.
  - Si le prédicat appartient à un pas de localisation, l'axe de filtrage est celui de ce pas,
  - Si le prédicat appartient à une expression de filtrage, l'axe de filtrage est l'axe child (axe par défaut)

- A chaque nœud de l'ensemble de nœuds est associée une position de proximité qui est égale à la position de ce nœud dans la séquence obtenue en triant les nœuds collectés dans le sens de l'axe de filtrage.

Valeur d'un pas de localisation

- La valeur d'un pas de localisation

  `axe::test` $p_1 \ldots p_n$

  à partir d'un nœud contexte n est l'ensemble de nœuds E_{n+2} construit de la façon suivante :
  - E_1 := ensemble des nœuds sélectionnés par l'axe `axe` à partir de n
  - E_2 := sous-ensemble des nœuds de E_1 qui vérifient le test de nœud `test`
  - Pour i de 1 à n : E_{i+2} := filtrage de E_{i+1} par le prédicat p_i

Valeur d'un chemin de localisation

- Un chemin de localisation relatif `p1/…/pn` a pour valeur l'ensemble de nœuds E_2 construit de la façon suivante :
  - E_1 := {nœud contexte}
  - Pour i de 1 à n :
    - E_2 := {}
    - Pour chaque nœud n de E_1 :
      - nœud contexte := n
      - E_2 := $E_2 \cup$ valeur(`pi`)
    - E_1 := E_2
- **Rappel**: la valeur d'un chemin de localisation absolu `/c` est valeur(`c`) dans un contexte où le nœud contexte est le nœud racine.

Fonctions de conversion

	type de *x*							
	chaîne		nombre		booléen		ensemble de nœuds	
	vide	non vide	nul	non nul	faux	vrai	vide	non vide
`string(`*x*`)`	*x*		représentation de *x* sous le format IEEE 754		`"false"`	`"true"`	valeur textuelle du nœud de *x* qui est le 1er dans l'ordre du document	
`number(`*x*`)`	nombre le plus proche de *x*, si *x* représente un nombre, "NaN" sinon		*x*		0	1	`number(string(`*x*`))`	
`boolean(`*x*`)`	`false`	`true`	`false`	`true`	*x*		`false`	`true`

Fonctions XPath

- **ceiling( )** : number ceiling(number x)
- **concat( )** : string concat(string s1, string s2, …)
- **contains( )**: boolean contains(string s1, string s2)
- **count( )**: number count(node-set set)
- **normalize-space( )**: string normalize-space(string s)
- **starts-with( )**: boolean starts-with(string s1, string s2)
- …
- **sum( )**: number sum(node-set nodes)

Fonctions de comparaison

- L'expression v1 `op` v2 (où `op` est un des opérateurs `=`, `!=`, `<`, `<=`, `>` ou `>=`) est vraie dans les cas suivants et fausse dans les autres :
  - v1 et v2 sont des ensembles de nœuds et il existe un nœud n1 de v1 et un nœud n2 de v2 tel que valeur-textuelle(n1) `op` valeur-textuelle(n2) est vraie
  - v1 est un ensemble de nœuds et
    - v2 est un booléen et boolean(v1) `op` v2 est vraie ;
    - v2 est une chaîne ou un nombre et il existe un nœud n de v1 tel que valeur-textuelle(n) `op` v2 est vraie ;

    (idem en échangeant v1 et v2)

Fonctions de comparaison - suite

- aucun des opérandes n'est un ensemble de nœuds et
  - `op` est `=` ou `!=` et
    - l'un des opérandes est un booléen et boolean(v1) `op` boolean(v2) est vraie ;
    - l'un des opérandes est un nombre et number(v1) `op` number(v2) est vraie ;
    - l'un des opérandes est une chaîne et string(v1) `op` string (v2) est vraie ;
  - `op` est `<`, `<=`, `>` ou `>=` et number(v1) `op` number(v2) est vraie.

Conversion automatique

- Si la valeur *v* d'un prédicat n'est pas un booléen, elle est convertie en booléen de la façon suivante :
  - si *v* est un nombre égal à la position contexte, *v* est convertie en `true` sinon *v* est convertie en `false` ;
  - si *v* n'est pas un nombre, elle est convertie en `boolean(`*v*`)`.
- Si la valeur *v* de l'argument d'une fonction prédéfinie n'a pas le type attendu (c.-à-d. celui de l'argument formel correspondant), *v* est automatiquement convertie, si cela est possible, en utilisant les fonctions de conversion `string`, `number` et `boolean`.

Fonctions sur les ensembles de nœuds

- Position et taille contexte: les expressions `position()` et `last()` ont pour valeurs respectives la position contexte et la taille contexte

- Union: l'expression e_1 `|` e_2 a pour valeur l'union des ensembles de nœuds *valeur*(e_1) et *valeur*(e_2)

- Filtrage: l'expression `e[p]` produit l'ensemble de nœuds obtenu par filtrage de l'ensemble *valeur*(*e*) par le prédicat *p*. L'axe de filtrage est l'axe `child`

- Accès par identifiant : l'expression `id(`*`idrefs`*`)` où *`idrefs`* est une suite de noms d'identificateurs séparés par des espaces

Abréviations

Syntaxe normale	Syntaxe abrégée
`child`	(c'est l'axe par défaut)
`attribute::`	`@`
`/descendant-or-self::node()/`	`//`
`self::node()`	`.`
`parent::node()`	`..`
`[position()=`*i*`]`	`[i]`

Si le chemin débute par `//` alors le contexte initial est l'élément racine

Exemples avec des abréviations

- `element()[2]`: second élément fils
- `//*`: tout les éléments descendant du nœud document
- `//text()`: tout les nœuds textes descendant du nœud document
- `expose[2]/comment[1]`: premier élément `comment` du second exposé
- `.//@day`: tout les nœuds attributs nommés `day` dans un descendant du nœud contexte
- `//expose[.//comment]/title`: le titre des exposés qui contiennent un élément `comment`