



Langages du Web

MI – Semestre 7
Département informatique
UFR des Sciences et technologies
Université de Thiès



Plan du cours

1. Présentation du cours
2. Introduction
3. XML et ses applications
4. RDF, OWL & RIF



Plan du cours

1. Présentation du cours
2. Introduction
3. XML et ses applications
4. RDF, OWL & RIF



Présentation du cours (1)

Unité d'Enseignement

Titre : INFORMATIQUE

Sigle : INF 113

Élément constitutif

Titre : Langages du Web

Sigle : INF 1132

Autres éléments constitutifs de l'UE

Titre : Bases de données réparties

Sigle : INF 1131

Volume horaire



Présentation du cours (2)

Horaires

- CM : 20H
- TD/TP : 20H
- TPE : 40H
- Coefficient de l'UE : 3
- Crédits de l'UE : 9

Evaluation

- Contrôle des connaissances : 40%
- Examen écrit : 60%



Présentation du cours (3)

Responsable du cours

Pr Mouhamadou THIAM
Maître de conférences en Informatique
Intelligence Artificielle : Sémantique Web
Email : mthiam@univ-thies.sn

Travaux dirigés et pratiques

M. ??????

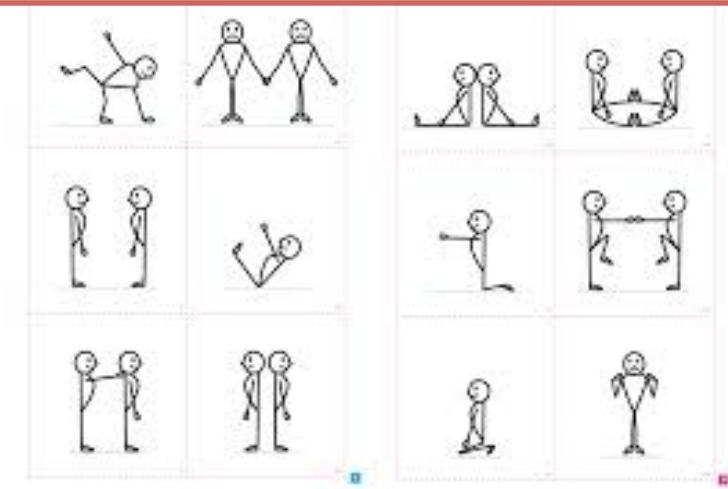
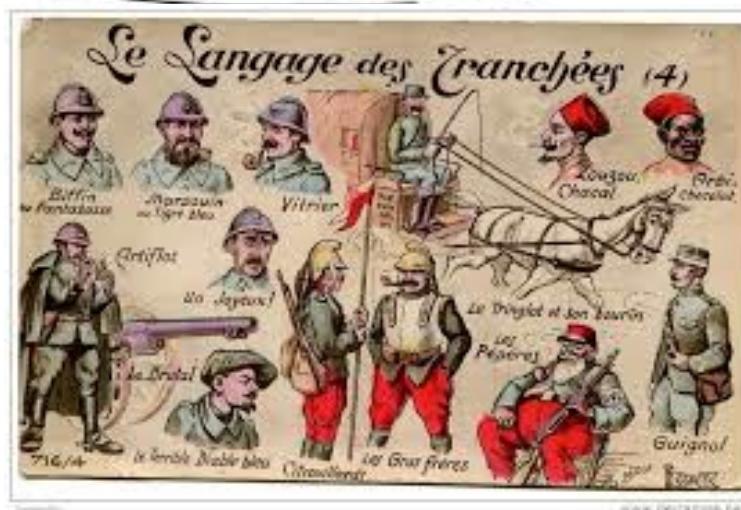


Plan du cours

1. Présentation du cours
2. Introduction
3. XML et ses applications
4. RDF, OWL & RIF



Langage ...





Langage ...

- Langage humain
- Langage animal
- Langage formel
- Langage naturel
- Langage de programmation
- Langage informatique
- Langage de script
- Langage SMS
- Langage des fleurs
- Langage interprété informatique



Web...





Web [wikipedia]...

- **World Wide Web (WWW)**, littéralement la « **toile (d'araignée) mondiale** », communément appelé le **Web**, et parfois la **Toile**, est un système **hypertexte** public fonctionnant sur **Internet**.
- Le Web permet de consulter, avec un navigateur, des pages accessibles sur des sites. L'image de la toile d'araignée vient des hyperliens qui lient les pages web entre elles



Web...

- Applications d'Internet
- Distinct d'autres applications comme
 - courrier électronique,
 - messagerie instantanée,
 - partage de fichiers en pair à pair.
- Inventé par ***Tim Berners-Lee*** et ***Robert Cailliau*** au début des années 90



La révolution du Web...

- Depuis vingt cinq ans, Internet révolutionne l'informatique « grand public »
- **HTML** est le langage du Web...
 - Même si on trouve aussi du .doc, .ps, .pdf... et des images (jpg, gif), du son, de la vidéo...



La révolution du Web...

- Des milliards de pages existent actuellement
 - public/privé,
 - statique/dynamique,
 - visible/caché
- Support naturel pour l'information distribuée
 - À destination d'êtres humains et
 - de plus en plus, pour des applications



La révolution du Web...

- Exemples:
 - B2C (commerce électronique)
 - B2B (achats groupés)
 - Bibliothèques et fonds documentaires en ligne
- HTML n'est pas adapté pour ces applications
 - Ces applications (programmes) ont besoin de typage pour représenter **la structure** des données



Prérequis ...

- Les bases de données (relationnelle, ...)
 - Conception de schémas : relationnel, UML, ...
 - Les langages de requêtes : SQL, OQL, ...
- Le langage HTML : tables, listes, formulaires, ...
- Le langage Java
 - Héritage
 - Utilisation d'API



Prérequis ...

- Les bases de données (relationnelle, ...)
 - Conception de schémas : relationnel, UML, ...
 - Les langages de requêtes : SQL, OQL, ...
- Le langage HTML : tables, listes, formulaires, ...
- Le langage Java
 - Héritage
 - Utilisation d'API



Problématique ...

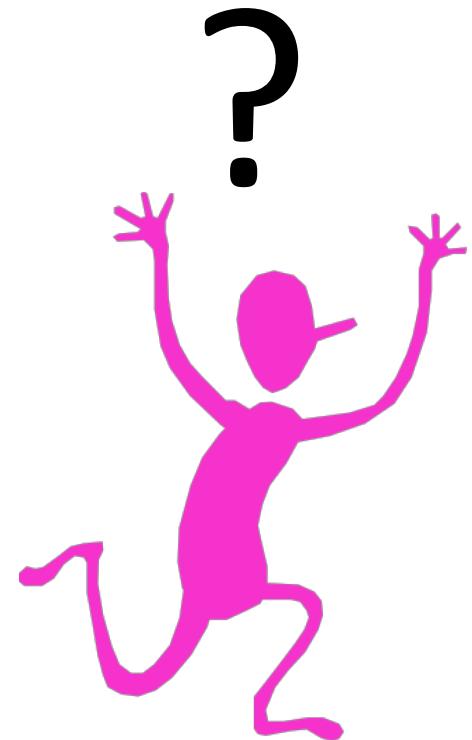
- Besoins d'accès contenu du web de manière précise
- Recherche mot-clef utilisée : pas satisfaisant;
- Méthodes ne garantissent pas
 - Pertinence de la réponse,
 - Réponse : document lui-même
 - Nombre exorbitant de réponses
 - Majeure partie répondent pas à la question posée
- Le web sémantique **human → machine readable**



Résumé historique du web

- **1989** : **Tim Berners-Lee** (CERN, Genève) commence le développement d'un **système hypertexte**.
- **1990** : Premières définitions pour **HTTP, HTML, URL**.
- **1992** : Premier annuaire de sites Web. **26 sites**.
- **1994** : **Netscape Navigator** 1.0 ; Fondation du **W3C**
- **1998** : Plus de **2 millions de sites** ; Création de **Google**.
- **2000** : **XHTML** 1.0.
- **2004** : **Firefox** 1.0.
- **2005** : Plus de **60 millions de sites** !!!

END





Plan du cours

1. Présentation du cours
2. Introduction
3. XML et ses applications
4. RDF, OWL & RIF

Lectures

- Obligatoires
 - Recommandations du W3C - http://www.w3.org/TR/technology-stds#w3c_all
 - XML, XML Schema
 - XPath, XQuery
 - XSLT
 - <http://www.w3.org/History/1989/proposal.html>
 - *The Semantic Web : A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities* by TIM BERNERS-LEE, JAMES HENDLER and ORA LASSILA
- Facultatives
 - Recommandations HTML du W3C



XML et ses applications : plan

1. Introduction à XML
2. Schémas de documents XML
 - a. DTD
 - b. XML Schema
3. Le modèle DOM
4. Interrogation : XPath
5. Transformation et présentation : XSLT
6. Programmation : XML et Java (API DOM, SAX, ...)

Introduction à l'eXtensible Markup Language –XML

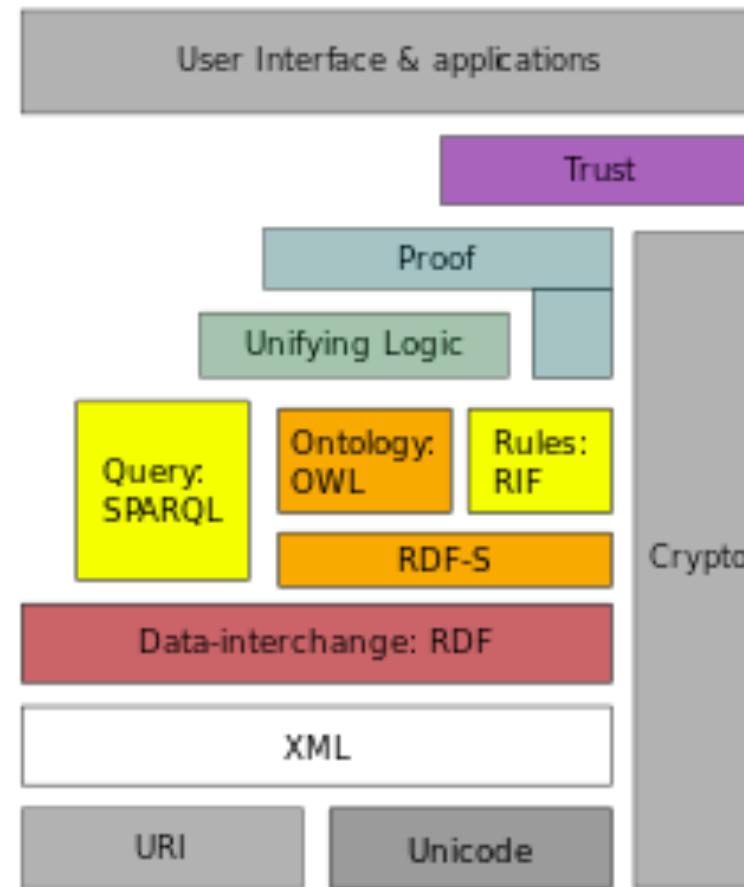


De SGML à XML en passant par HTML

- SGML[1969, ISO 8879 1986]
 - Inventé par *Charles Goldfarb*, juriste chez IBM.
 - Du balisage typographique au balisage logique.
- HTML[1989-1992, HTML5 depuis 2007]
 - Inventé par *Tim Berners-Lee* comme langage de description des pages du Web.
 - Concept d'ancre : hypertexte.
- XML[1996, Rec. W3C 1998, Dan Cannolly]
 - Une synthèse de SGML et de HTML.



Modèle en couches de Tim Berners-Lee





XML : Document semi-structuré

- Un Document Semi-Structuré peut se définir par la définition de ses principaux composants (Poulet et al, 1997): **contenu** et **structure**
 - **Contenu** : le contenu d'un document semi-structuré désigne le contenu textuel ou multimédia par exemple des paragraphes, des figures ou des images
 - **Structure**
 - **Logique** : elle définit l'organisation hiérarchique des données du document
 - **Physique** : elle définit la présentation du document sur le support



XML : le modèle de données du Web

- HTML versus XML
 - Attention : HTML et XML ne sont pas concurrents !!!
 - Présentation des données pour HTML
 - Représentation des données pour XML : les données et leur visuel est indépendant
 - Pour s'en convaincre, la présentation d'un document XML nécessite sa transformation en HTML via XSLT
 - XSLT= eXtensible Stylesheet Language Transform



XML : le modèle de données du Web

- Exemple
 - Le fichier [HTML](#) contenant présentation et données
 - Le fichier [XML](#) ne contenant que les données
 - Le fichier [XSL](#) transformant le fichier XML en HTML

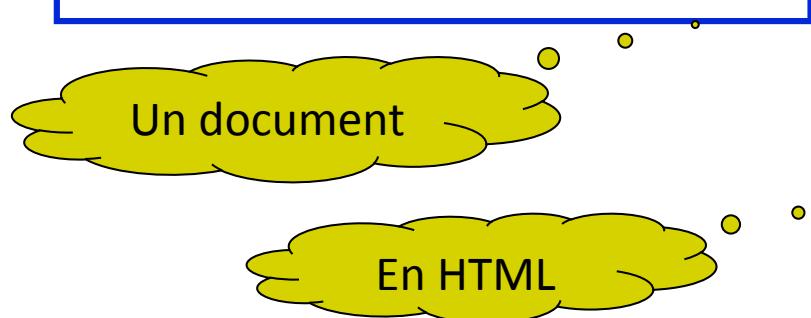


Exemple de document (text / HTML)

Construire une application XML

Domaines d'application d'XML
Vue d'ensemble

XML : battage ou réalité ?



```
<!--extrait d'un document HTML-->
<HTML>
  <HEAD>
    <TITLE> Construire une application XML </TITLE>
  </HEAD>
  <BODY>
    <center>
      <P>Domaines d'application d'XML <br>
          Vue d'ensemble</P>
    </center>
    <P>XML : battage ou réalité ? </P></center>
  </BODY>
</HTML>
```



Exemple de document (HTML / XML)

```
<!--extrait d'un document HTML-->
<HTML>
<HEAD>
    <TITLE> Construire une application XML </TITLE>
</HEAD>
<BODY>
<center>
    <P>Domaines d'application d'XML <br>
        Vue d'ensemble</P>

    <P>XML : battage ou réalité ? </P></center>
</BODY>
</HTML>
```

En HTML

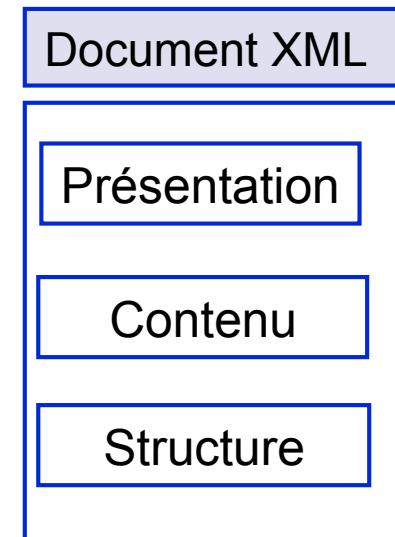
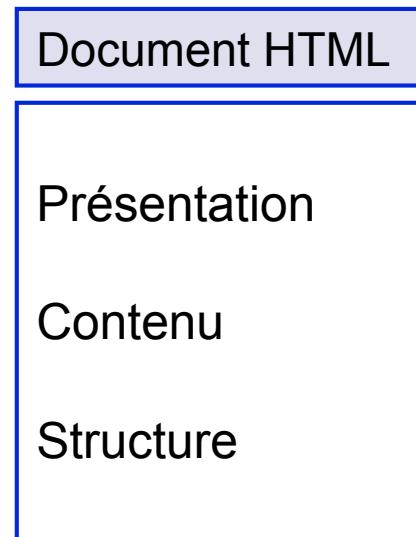
En XML

```
<!--extrait d'un document XML-->
<?xml version="1.0" ?>
<livre>
    <couverture>
        <titre> Construire une application XML </titre>
        <auteur> Jean-Christophe Bernadac </auteur>
        <auteur> François Knab </auteur>
    </couverture>
    <chapitre n="1">
        <p> XML : battage ou réalité ? </p>
    </chapitre>
</livre>
```

Presentation
Contenu
Structure



HTML versus XML





Exemple de document

Itinéraires skieurs dans la Vallée de la Clarée

par Jean-Gabriel Ravary

Le Polygraphe, éditeur
1991

Vallon des Muandes

Vallon situé à l'est du refuge des Drayères.

Le vallon le plus utilisé pour la traversée sur la Vallée Etroite. Ce vallon est également accessible du refuge Laval.

Col de Névache (2 794 m) ** n° 1

S'élever au-dessus du refuge des Drayères en direction est. Suivre la rive droite du torrent de Brune puis s'engager sur le flanc droit du ravin des Muandes que l'on quitte vers 2500 m pour rejoindre le col situé au nord.

Descente possible sur Valmeinier. Départ assez raide.

Pointe de Névache (2 892 m) * n° 2**

Du col de Névache (*itinéraire n° 1*), suivre la ligne de crête qui mène à la pointe de Névache. Attention : corniches possibles. Crampons utiles au printemps.



Le guide “itinéraires skieurs” en XML

```
<?xml version="1.0"?>
<guide>
    <titre>Itinéraires skieurs dans la vallée de la Clarée</titre>
    <auteur>Jean-Gabriel Ravary</auteur>
    <éditeur>Le Polygraphe</éditeur>
    <année>1991</année>
    ...
    <vallon id="V15">
        <nom>Vallon des Muandes</nom>
        <intro>
            <para>Vallon situé à l'est du refuge des Drayères.</para>
            <para>Le vallon le plus utilisé pour la traversée sur la Vallée
                Étroite. Ce vallon est également accessible du refuge Laval.</para>
        </intro>
        ... Itinéraires ...
    </vallon>
</guide>
```



Le guide “itinéraires skieurs” en XML

```
<itinéraire id="I15.1">
    <nom>Col de Névache</nom><alt>2794</alt><cotation>**</cotation>
    <num>1</num>
    <para>
        S'élever au-dessus du refuge des Drayères en direction est. Suivre la rive droite du
        torrent de Brune puis s'engager sur le flanc droit du ravin des Muandes que l'on quitte
        vers 2500 m pour rejoindre le col situé au nord. Descente possible sur Valmeinier.
        <note type="prudence">Départ assez raide.</note>
    </para>
</itinéraire>
```



Le guide “itinéraires skieurs” en XML

```
<itinéraire id="I15.2">
    <nom>Pointe de Névache</nom><alt>2892</alt><cotation>***</cotation>
    <num>2</num>
    <para>Du col de Névache <renvoi cible="I15.1"/>, suivre la ligne de crête qui mène à la
        pointe de Névache.
        <note type="prudence">Attention : corniches possibles.</note>
        <note type="materiel">Crampons utiles au printemps.</note>
    </para>
</itinéraire>
```



En résumé

- Un **document** XML est découpé en **éléments** structurés hiérarchiquement.
- Un document a un élément racine appelé **élément du document**.
- Un élément est composé :
 - d'un **nom** qui spécifie son type,
 - **d'attributs**,
 - d'un **contenu** formé d'**éléments** ou de **textes**.



En résumé

- Un texte est une chaîne de caractères.
- Un attribut a un nom et une valeur qui est une chaîne de caractères.
- Syntaxiquement, les éléments d'un document XML sont marqués dans le document lui-même par des paires de balises ouvrantes et fermantes.



Structure d'un document XML

- Un document XML est composé :
 - d'un **prologue facultatif** (voir ci-après « Organisation d'un document XML »),
 - de **l'élément du document** qui est lui-même composé d'**éléments et de textes**



Structure d'un document XML

- Dans le prologue et dans le contenu d'un élément, on peut insérer :
 - des **commentaires**,
 - des **instructions de traitement** destinés aux applications traitant le document.
- Un document peut être découpé en **entités** enregistrées dans un ou plusieurs fichiers.



Nom et tokens de nom

- Un **caractère de nom** est soit une lettre, soit un chiffre, soit un point, soit un tiret, soit un espace souligné, soit un deux-points.
 - {[A..Z], [0..9], ., -, _, :}
- Un **nom** est une suite de un ou plusieurs caractères dont : le premier est soit une lettre, soit un espace souligné, soit un deux-points { _, : } , chacun des suivants est un caractère de nom.

Par exemple : xml:lan, extrait_de titre, poeme-79

- Un **token de nom** est une suite de un ou plus caractères de nom.



Un élément XML

- Un élément est composé :
 - d'une **balise de début** qui contient le nom de l'élément et éventuellement ses attributs `<elt attr="valAttr">`,
 - d'un **contenu**,
 - d'une **balise de fin `</elt>`**.



Un élément XML

- Par exemple :

```
<note type="prudence">  
    Départ assez raide.  
</note>
```

- balise de début : `<note type="prudence">`
- nom : `note`
- attribut : `type="prudence"`
- contenu : `Départ assez raide.`
- balise de fin : `</note>`



Contenu d'un élément XML

- **Vide** : <elt></elt> ou <elt/>

Exemple :

```
<renvoi cible="l15.1"></renvoi>
```

ou

```
<renvoi cible="l15.1"/>
```

- **Composé** d'éléments :

```
<intro>
```

```
    <para>Vallon situé à l'est du refuge ...</para>
```

```
    <para>Le vallon le plus utilisé pour la traversée ...</para>
```

```
</intro>
```



Contenu d'un élément XML

- **Mixte** : mélange de textes et d'éléments

```
<nom>Col de Névache</nom>
<para>Du col de Névache <renvoi cible="l15.1"/>,
    suivre la ligne de crête qui mène à la pointe de Névache.
    <note type ="prudence">Attention : corniches possibles. </note>
    <note type="matériel">Crampons utiles au printemps.</note>
</para>
```



Elément mixte

- Le contenu d'un élément mixte est constitué d'une chaîne de caractères dans laquelle peuvent être insérés des éléments. Cette insertion découpe ce contenu en deux types de constituants :
 - les plus longues suites d'au moins un caractère dans lesquelles ne sont pas insérés des éléments : nous les appellerons **textes**,
 - les éléments.



Elément mixte

- Par exemple, le contenu :
Du col de Névache <renvoi cible="I15.1"/>, suivre la ligne de crête qui mène à la pointe de Névache.
- comprend dans l'ordre :
le texte : Du col de Névache
l'élément : <renvoi cible="I15.1"/>
le texte : , suivre la ligne de crête qui mène à la pointe de Névache.



Section CDATA

- Lorsqu'un texte contient des caractères qui jouent un rôle de **délimiteur** dans la syntaxe XML, il est nécessaire de pouvoir **inhiber** ce rôle. Ceci peut être fait en insérant le texte contenant ces délimiteurs dans une section **CDATA** sous la forme suivante :

<![CDATA[*texte contenant des délimiteurs*]]>



Section CDATA

- Le texte inséré peut contenir n'importe quels caractères excepté la chaîne `]]`. Une section CDATA ne peut donc pas en contenir une autre.
- Par exemple, la phrase :
`« L'expression <ALT>2794</ALT> est un élément XML. »`
- peut être représentée par l'élément suivant :
`<phrase>L'expression
 <![CDATA[<ALT>2794</ALT>]]> est un élément XML.
</phrase>`



Attributs

- Un **attribut** est un couple **nom-valeur** où :
 - le nom est un nom XML,
 - la valeur est une suite de caractères.
- Par exemple :
`type="prudence"`
- Si une valeur d'attribut est placée entre guillemets, elle peut contenir des apostrophes et si elle est placée entre apostrophes, elle peut contenir des guillemets.
- Par exemple :
`select="itinéraire[cotation='****']"`
`select='itinéraire[cotation="****"]'`



Liens internes

- Tout élément peut avoir un attribut ayant pour valeur un token de nom qui l'identifie dans le document :
 - les tokens de nom **I15.1** et **I15.2** identifient les itinéraires n° 1 et n° 2 du Vallon des Muandes dans le document « Itinéraires skieurs ».
- L'identificateur d'un élément permet d'y faire référence depuis d'autres éléments :
 - <itinéraire id="**I15.1**"></itinéraire>
 - l'élément <renvoi cible="**I15.1**"> renvoie à l'itinéraire n° 1 de ce même vallon).



Commentaires

- Un commentaire est une phrase ayant la forme suivante :
<!--texte du commentaire-->
- Un commentaire peut contenir n'importe quel caractère **excepté --**.
Un commentaire ne peut donc pas inclure un autre commentaire.
- Un commentaire peut être inclus dans le contenu d'un élément mais pas à l'intérieur d'une balise.

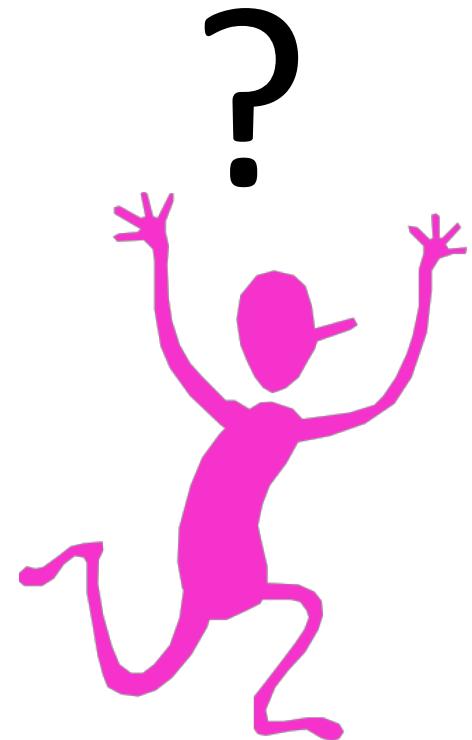
- Exemple :
<!-- Les itinéraires sont classes par vallon -->



Résumé

- **langage de balises** (simple à implanter, mais extensible) : chacun définit ses propres balises
- **séparation de la présentation et du contenu**, conçu pour
 - **décrire données** en se concentrant sur leur **structure**
 - **assurer l'interopérabilité**
- Utilisation d'un **DTD** ou un **XML Schéma** : modèle de données
- XML et HTML complémentaires
 - **XML** ne remplace pas **HTML**
 - **XSL / XSLT** transforme **XML** en **HTML**

END



Introduction aux schémas de données XML



Schéma

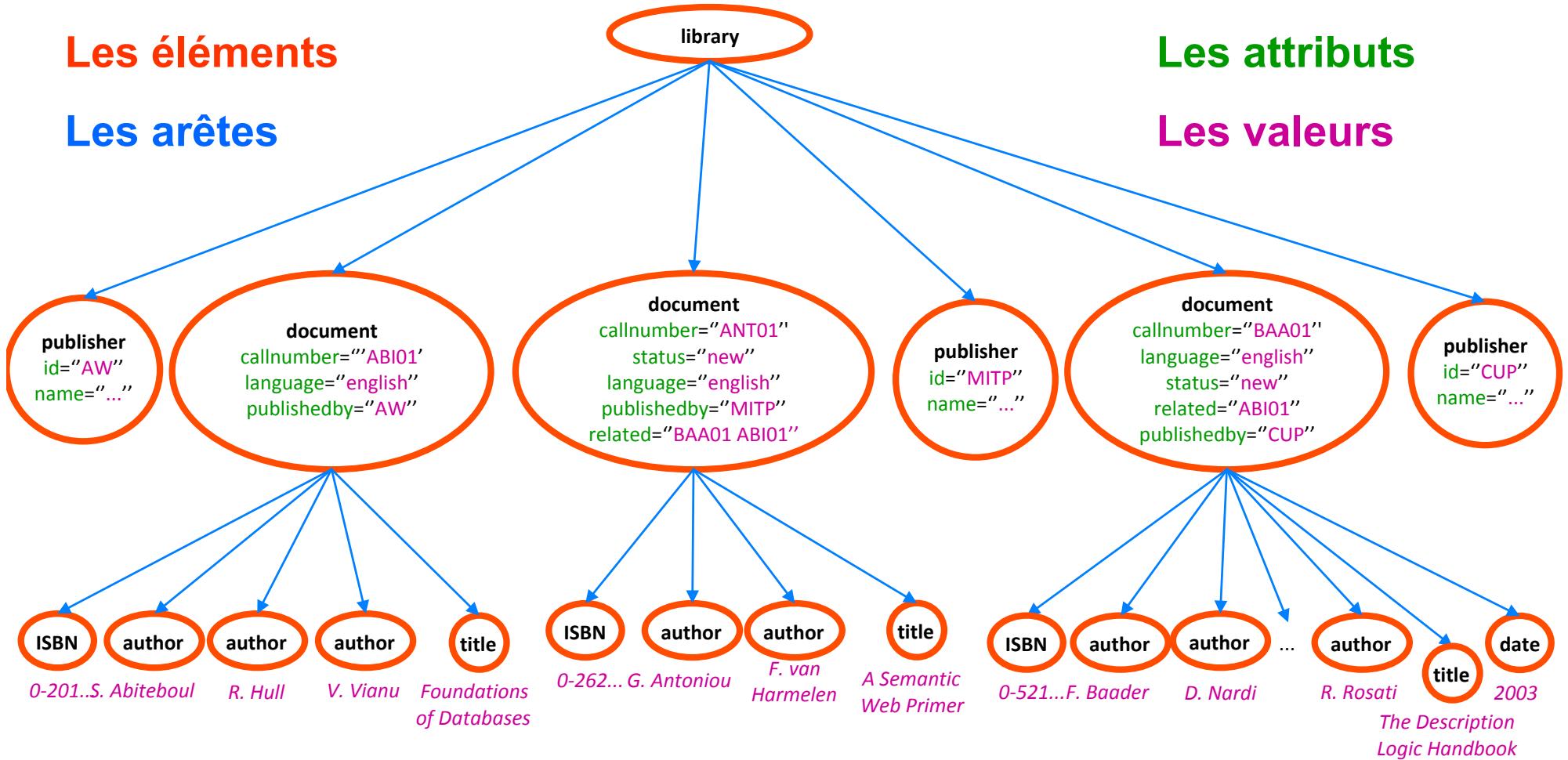
- Le schéma peut être commun à plusieurs documents
 - Facilite l'échange / le partage de données
- Le schéma permet une représentation plus fine des données
 - XML sans schéma : arbre
 - XML + schéma : graphe
- Dans le modèle de données XML, deux langages de définition de schémas
 - DTD
 - Pour des schémas simples
 - XML Schema
 - Pour des schémas complexes

Représentation graphique de données XML



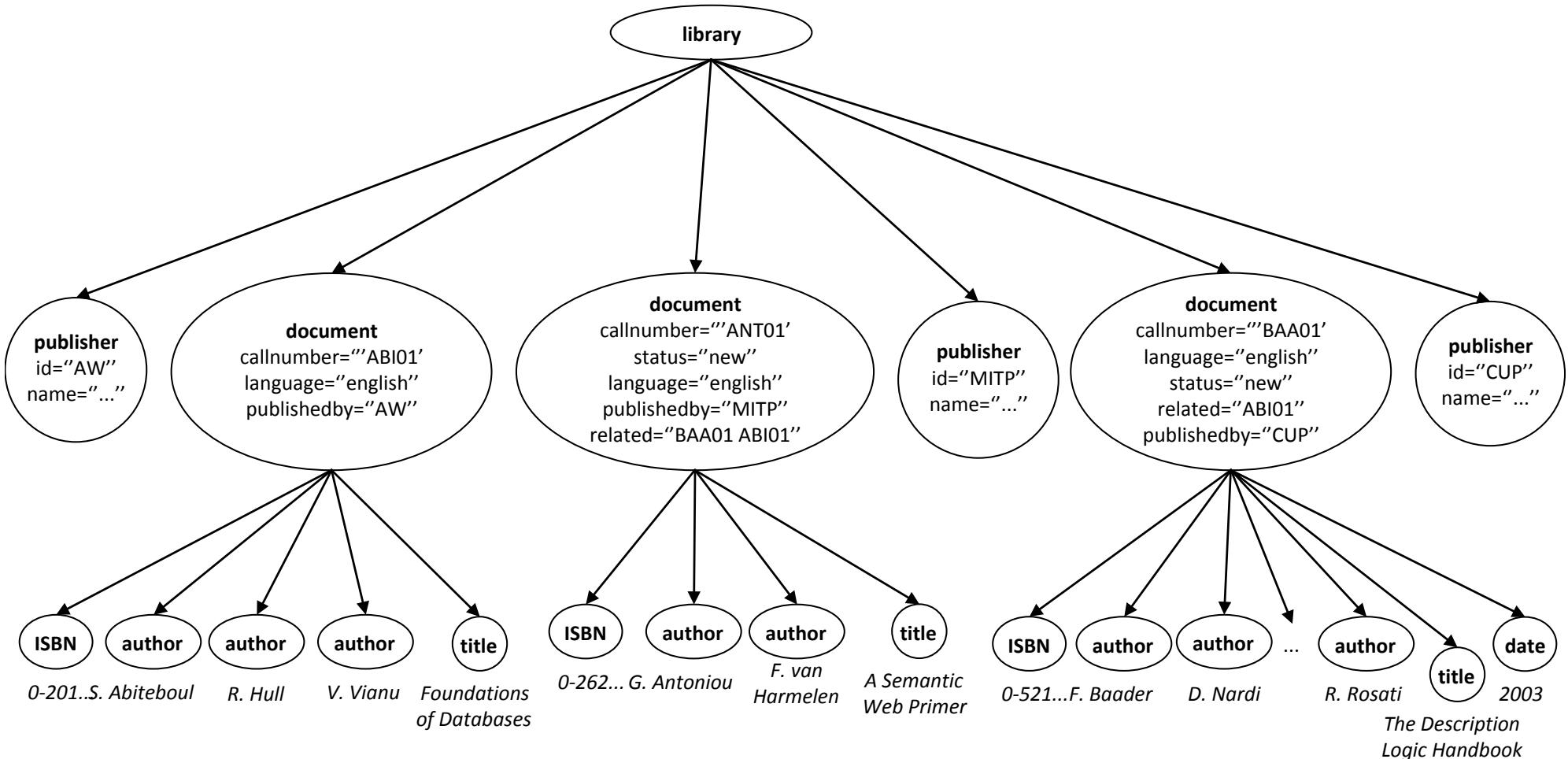
Les éléments

Les arêtes



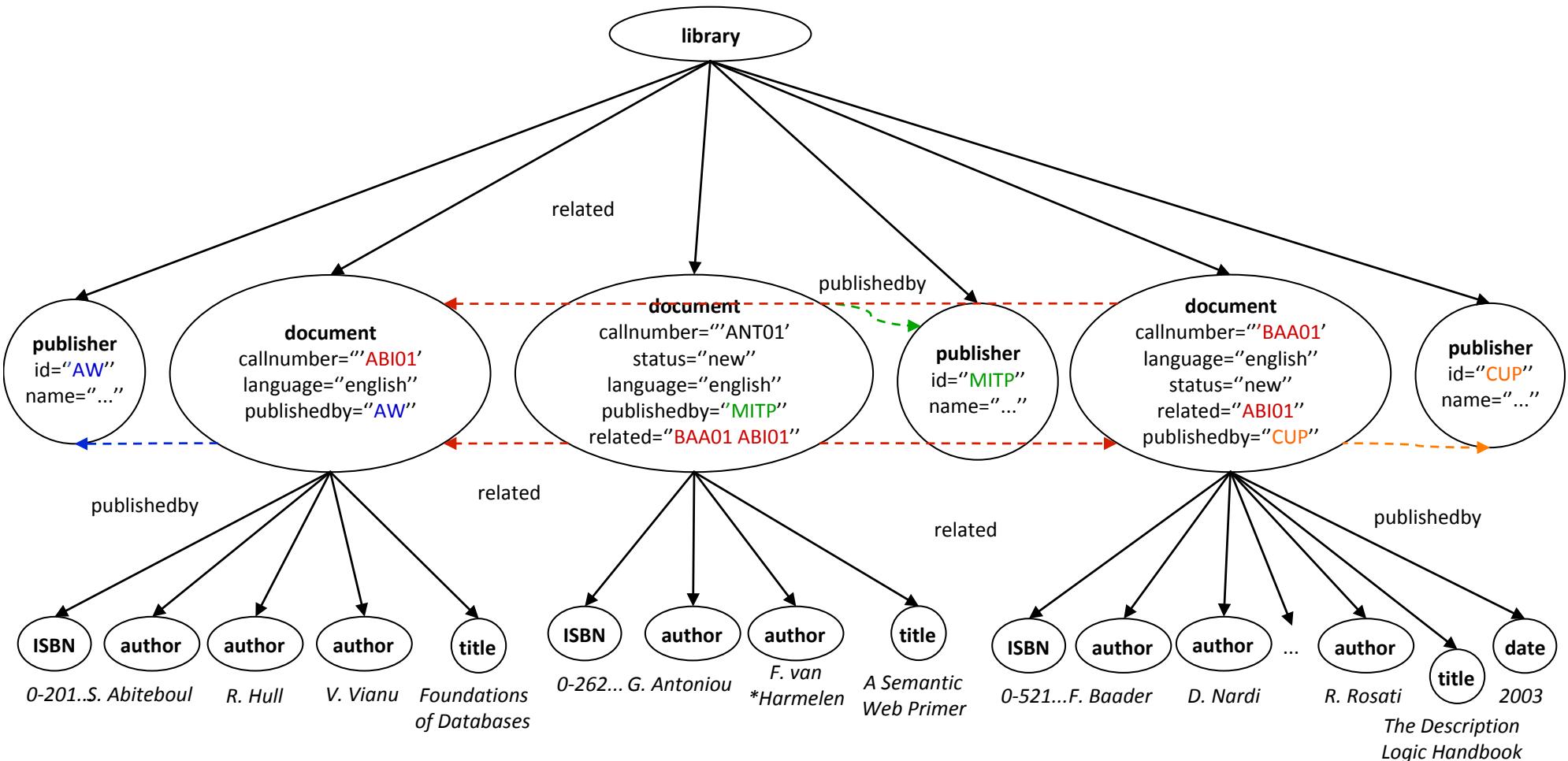


XML sans schéma : arbre





XML avec schéma : graphe





Schémas : validité

- Un schéma permet de vérifier la **validité/cohérence** des données.
- Le schéma peut être commun à plusieurs documents
 - Facilite l'échange / le partage de données
- Dans le cas général, on **DOIT** associer un schéma à tout document

DTD

Document Type Definition



DTD

- Les éléments qui décrivent un document peuvent être définis dans une **DTD** (Déclaration de Type de Document), mais ce n'est pas obligatoire.
- Un document XML est dit **valide** s'il est précédé de sa DTD et si sa description est conforme à cette DTD.
- Un document XML est dit **bien formé** s'il n'est pas précédé d'une DTD mais si sa description est syntaxiquement correcte.



DTD

- La déclaration d'un type de document (DTD) est composée d'une suite de déclarations :
 - déclaration d'éléments
 - déclaration des attributs d'un élément
 - déclaration d'entités



DTD d'un guide d'itinéraires à skis

- Un guide est composé d'un **titre**, d'une liste d'un ou plusieurs **auteurs**, d'un **éditeur**, d'une **année** et d'une liste d'un ou plusieurs **vallons**.
- Un titre, un auteur, un éditeur et une année sont des **textes**.
- Un vallon est composé d'un **nom**, d'une **introduction** et de la liste des **itinéraires** que l'on peut y réaliser (un ou plusieurs itinéraires).
- Un nom est un **texte**.



DTD d'un guide d'itinéraires à skis

- Une introduction est composée d'une liste d'un ou plusieurs **paragraphes**.
- Un paragraphe est un texte dans lequel sont insérés des **renvois** vers d'autres itinéraires et des **notes**.
- Une note est un texte donnant des **consignes** de prudence ou recommandant l'utilisation d'un **matériel** spécifique (crampons, piolet, etc.).

DTD d'un guide d'itinéraires à skis



- <!ELEMENT guide (titre, auteur+, editeur, année, vallon+)>
 - <!ELEMENT titre (#PCDATA)>
 - <!ELEMENT auteur (#PCDATA)>
 - <!ELEMENT editeur (#PCDATA)>
 - <!ELEMENT année (#PCDATA)>
 - <!ELEMENT vallon (nom, intro, itinéraire+)>
 - <!ATTLIST vallon id ID #REQUIRED>
 - <!ELEMENT nom (#PCDATA)>
 - <!ELEMENT intro (para+)>
 - <!ELEMENT para (#PCDATA | renvoi | note)*>
 - <!ELEMENT renvoi EMPTY>
 - <!ATTLIST renvoi cible IDREF #REQUIRED>
 - <!ELEMENT note (#PCDATA)>
 - <!ATTLIST note type (prudence | materiel) "prudence">
 - <!ELEMENT itinéraire (nom, alt, cotation, num, para+)>
 - <!ATTLIST itinéraire id ID #REQUIRED>
 - <!ELEMENT alt (#PCDATA)>
 - <!ELEMENT cotation (#PCDATA)>
 - <!ELEMENT num (#PCDATA)>



Déclaration d'un élément

- Un élément est défini par la déclaration :
`<!ELEMENT nom modèle de contenu>`
- Un élément mixte pouvant contenir des éléments T_1, \dots, T_n a pour modèle de contenu :
`(#PCDATA | T_1 | ... | T_n)^*`



Déclaration d'un élément

- Un élément composé d'une suite d'éléments T_1, \dots, T_n a pour modèle de contenu l'expression régulière construite sur le vocabulaire $\{T_1, \dots, T_n\}$ à l'aide des opérateurs :
 - , (infixe) : concaténation
 - $* +$ (suffixes) : 0 ou plusieurs répétitions et 1 ou plusieurs répétitions
 - ? (suffixe) : optionalité
- Un élément vide a pour modèle de contenu EMPTY.
- Un élément de contenu quelconque a pour modèle de contenu ANY.

Déclaration des attributs d'un élément



- A chaque type d'élément est attaché un ensemble d'attributs.
- Une définition d'attributs a la forme suivante :
*<!ATTLIST nom-élément nom-attribut1 type1 déclaration-de-défaut1
...
nom-attribut2 type2 déclaration-de-défaut2>*
- où :
 - le type est celui des valeurs de l'attribut,
 - la **déclaration de défaut** spécifie si la valeur de l'attribut doit être ou non présente dans le document et fournit éventuellement une valeur par défaut.
- des éléments de type différent peuvent avoir des attributs de même nom.



Déclaration des attributs d'un élément

- La déclaration de défaut peut être :
 - **#REQUIRED** : l'attribut doit être présent dans la balise de l'élément,
 - **#IMPLIED** : l'attribut est facultatif, *valeur* : valeur à affecter à l'attribut s'il est absent de la balise de l'élément (valeur par défaut),
 - **#FIXED *valeur*** : valeur que doit avoir l'attribut s'il est présent dans la balise de l'élément ou qui lui sera affectée s'il est absent de cette balise.
- Les déclarations de défaut sont prises en compte par un analyseur XML afin de compléter le document analysé.



Déclaration des attributs d'un élément

- Le type de valeur peut être :
 - **CDATA** : texte,
 - **ID** : nom identifiant l'élément dans le document (que nous appellerons **identificateur**),
 - **IDREF** ou **IDREFS** : identificateur ou suite d'identificateurs (séparés par une suite de séparateurs : espace, tabulation),
 - **NMTOKEN** ou **NMTOKENS** : nom ou suite de tokens de nom,
 - **(nom₁ | ... | nom_n)** : un des tokens de nom énumérés.
 - **ENTITY**, **ENTITIES** et **NOTATION**

Réalisation physique d'un document : les entités



- Un document XML est physiquement découpé en **entités**.
- Une entité est un fragment nommé de document.
- On distingue :
 - les **entités générales** qui sont des fragments nommés de l'élément du document,
 - les **entités paramètres** qui sont des fragments nommés de DTD,
 - les **entités prédefinies** qui sont des caractères réservés de XML,
 - les **entités caractères** qui sont des caractères du jeu de caractères universel UNICODE, nommés par leur code numérique.

Réalisation physique d'un document : les entités



```
<?xml version="1.0" ?>
<!DOCTYPE guide SYSTEM "guide.dtd"
[ ...
<!ENTITY vallon-muandes
  SYSTEM "muandes.xml">
...
]>
<guide>
<titre>Itinéraires skieurs dans la Vallée de la
Clarée</titre>
<auteur>Jean-Gabriel Ravary</auteur>
<éditeur>Le Polygraphe</éditeur>
<année>1991</année>
... &vallon-muandes;
...
</guide>
```

```
<?xml version="1.0" ?>
<!ELEMENT guide (titre,
auteur+, éditeur, année, vallon+)>
<!ELEMENT titre (#PCDATA)>
```

```
<?xml version="1.0" ?>
<vallon>
<nom>Vallon des Muandes</nom>
<intro>
<para>Vallon situé à l'est du
refuge des Drayères.</para>
<para>Le vallon le plus utilisé pour la
traversée sur la Vallée Etroite. ...</para>
</intro> ... </vallon>
```



Remplacement des entités

- Une référence à une entité est remplacée par sa valeur lorsque l'élément ou la DTD qui la contient est traité par un parseur XML.
- Ce remplacement pourra entraîner des remplacements en cascade si cette entité contient elle-même des références à des entités et ainsi de suite.



Entité générales

- **Déclaration** d'une entité générale :
 - **interne** (enregistrée dans sa déclaration)
`<!ENTITY nom "entité">`
 - **externe** (enregistrée dans un fichier externe à celui de sa déclaration)
`<!ENTITY nom SYSTEM "nom du fichier contenant l'entité">`
 - Par exemple :
`<!ENTITY ref "refuge">`
`<!ENTITY vallon-muandes SYSTEM "mon_site/muandes.xml">`



Entité générales : Référence

- **Référence** à une entité générale : *&nom de l'entité;*
 - Par exemple :
`<para>S'élever au-dessus du &ref; des Drayères ...</para>`
 - est équivalent à :
`<para>S'élever au-dessus du refuge des Drayères ...</para>`



Entité paramètres

- **Déclaration** d'une entité paramètre :
 - interne (enregistrée dans sa déclaration)
`<!ENTITY % nom "entité">`
 - externe (enregistrée dans un fichier externe à celui de sa déclaration)
`<!ENTITY % nom SYSTEM nom du fichier contenant l'entité>`
 - Par exemple :
`<!ENTITY % identificateur ID #REQUIRED>`



Entité paramètres : référence

- **Référence** à une entité paramètre :
%nom;
 - Par exemple :

```
<!ATTLIST renvoi cible %identificateur;>
```

 - au lieu de :

```
<!ATTLIST renvoi cible ID #REQUIRED>
```



Entité prédéfinies

- Les caractères < > & " qui sont des délimiteurs XML peuvent être remplacés dans un texte par une référence à une entité prédéfinie.
- Ces entités sont les suivantes :
< référence au caractère <
> référence au caractère >
& référence au caractère &
' référence au caractère '
" référence au caractère "
- Par exemple, la phrase :
« L'expression <ALT>2794</ALT> est un élément XML. »
- peut être représentée par l'élément suivant :
`<phrase>L'expression <ALT>2794</ALT> est un élément XML.</phrase>`



Entités caractères

- Un caractère non disponible sur la station de travail peut être représenté par son code **Unicode** en décimal ou en hexadécimal, sous la forme d'une référence à une entité :
 - `&#code décimal;`
 - `ode hexadécimal;`
- Par exemple :
 - `&` référence au caractère &
 - `¦` référence à la lettre grecque Φ



Organisation d'un document XML valide

- Un **document XML valide** est composé d'une **entité document** (sans nom) et d'un ensemble d'**entités externes**.
- L'entité document est composé d'un **prologue** et de l'élément du document.
- Le prologue est composé d'une **déclaration XML** et d'une DTD.
- **La déclaration XML indique** : la version de XML, le jeu de caractères et l'éclatement ou non du document en plusieurs entités externes.
- La DTD est constituée d'une **partie interne** placée dans l'entité document et d'une **partie externe**, enregistrée dans un fichier à part dont le nom est déclaré dans l'entité document.



Organisation d'un document XML valide

- La partie interne de la DTD, l'élément du document et les entités externes peuvent appeler des entités externes. Ces appels doivent être non récursifs et non circulaires.
- L'organisation d'un document bien formé est similaire à celle d'un document à l'exception de la DTD qui est :
 - soit absente,
 - soit présente mais ne contient que des déclarations d'entités générales.



Document XML monofichier

- <?xml version="1.0" encoding="..." standalone="yes" ?>
- <!DOCTYPE *nom* [*déclarations*]>
 <*nom*>
 ...
 <*nom*>
- où :
 - L'attribut `standalone='yes'` indique que le document est contenu en entier dans le fichier.
 - Le nom de l'élément du document doit être identique à celui de la DTD



Document XML multifichiers

- <?xml version="1.0" encoding="..." standalone= "no" ?>
- <DOCTYPE *nom SYSTEM nom_fichier [partie interne de la DTD]*>
- élément du document
- où :
 - *nom_fichier* est le nom du fichier contenant la partie externe de la DTD ;
 - l'attribut **standalone="no"** indique qu'il est fait appel à des entités externes soit dans la partie interne de la DTD, soit dans l'élément du document ;
 - le nom de l'élément du document doit être celui de la DTD ;
 - une entité externe peut débuter (et c'est conseillé) par une déclaration XML sans attribut **standalone**.

END



Codage des caractères



Structure globale de document XML

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE nom SYSTEM "fichier.dtd ou URL" [
déclarations
]>
...
... corps du document ...
```

UCS : un jeu de caractères universel



- Le texte est stocké dans la mémoire de l'ordinateur sous forme de bits groupés en octets.
- 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 (de 1 à 4 octets) en codant sur un octet les caractères ASCII qui sont les plus fréquents.

UCS-32BE	UTF-8			
	Octet 1	Octet 2	Octet 3	Octet 4
00000000 00000000 0xxxxxxxx 00000000 00000yyy yyxxxxxxxx 00000000 zzzzyyyy yyxxxxxxxx 000uuuzz zzzzyyyy yyxxxxxxxx	0xxxxxxx 110yyyyy 1110zzzz 11110uuu	10xxxxxx 10yyyyyy 10zzzzzz 10yyyyyy	10xxxxxx 10yyyyyy	10xxxxxx

- UTF-16** : permet d'inclure des caractères de l'UCS-4 dans une chaîne codée en UCS-2.



Déclaration de 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 par défaut être en présence d'un codage UTF-8.

Entités externes non XML: Notation

Voir : <http://www.w3.org/TR/REC-xml-names/>

Définition



- Une *entité externe non XML* peut contenir n'importe quoi
 - image,
 - son,
 - données,
 - etc.
- Il faut au préalable définir un type :

```
<!NOTATION nom_du_type SYSTEM  
"url_associé_au_type" >
```

Exemples



- <!NOTATION JPEG SYSTEM "JPG">
- <!NOTATION JPG SYSTEM "JPG">
- <!NOTATION PNG SYSTEM "http://www.w3.org/TR/
REC-png">
- <!NOTATION SWF SYSTEM "http://
www.macromedia.com/software/flash">
- Définition de l'entité externe non XML
<!ENTITY maison SYSTEM "maison.jpg" NDATA jpeg>



Entité non XML: référence

Premier exemple

- La référence **&maison;** est impossible (ce n'est pas du XML).
- Son utilisation doit passer par un attribut typé « entité externe non XML » :

**DTD : | <!ELEMENT image EMPTY>
| <!ATTLIST image src ENTITY #REQUIRED>**

XML : | <image src="maison"/> <!-- pas de &..; -->

Entité non XML: référence



Deuxième exemple

- ***DTD***

```
<!NOTATION GIF89a PUBLIC "-//Compuserve//    NOTATION  
                                Graphics Interchange Format 89a//EN">  
<!ATTLIST image source ENTITY #REQUIRED>  
<!ENTITY vacances SYSTEM "images/plage.gif" NDATA GIF89a>  
<image source="vacances">
```

- La référence **&source;** est impossible (ce n'est pas du XML).
- XML : **<image source="vacances">**

Espaces de noms

Voir : <http://www.w3.org/TR/REC-xml-names/>



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 URL (Uniform Resource Locator).
- Pour obtenir des noms uniques, il suffit de qualifier chaque nom par l'URL 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 (alias)** à chaque espace de noms.



Déclaration d'un espace de noms

- La déclaration d'un espace de noms et de son préfixe associé consiste à insérer dans la balise ouvrante d'un élément contenant des noms (d'éléments ou d'attributs) issus de cet espace, l'attribut :
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 l'annihiler par la déclaration :
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 `monSite/dtdLivre.dtd` contienne une DTD pour la description de livres et que les éléments `auteur`, `éditeur` et `année` du guide «Itinéraires skieurs» soient conformes à ces définitions, la description de ce guide pourrait être la suivante :

```
<guide xmlns:livre="monSite/dtdLivre.dtd">
  <livre:titre>Itinéraires skieurs dans la Vallée de la Clarée
  </livre:titre>
  <livre:auteur>Jean-Gabriel Ravary</livre:auteur>
  <livre:éditeur>Le Polygraphe</livre:éditeur>
  <livre:année>1991</livre:année>

  ...
  <vallon>
    <nom>Vallon des Muandes</nom>
  ...
</guide>
```

END



Le langage XML Schema

<http://www.w3.org/TR/xmlschema-0/>

<http://www.w3.org/TR/xmlschema-1/>

<http://www.w3.org/TR/xmlschema-2/>

Introduction (1)



- Un schéma écrit en **XML Schema** définit une classe de documents
- Les **XML Schémas** améliore les **DTD** :
 - Un XML schéma est un fichier XML :
 - pas un nouveau langage, pas de parseur dédié.
 - Si on sait lire un fichier XML, on peut lire un schéma
 - Propose des types de données utiles (en plus des chaîne de caractères)
 - Booléen, entier, décimal, chaîne, date,...

Introduction (2)



- Un XML schema est composé :
 - **D'un prologue :**
 - indique quels espaces de noms vont être utilisés
 - Espace de noms : pointeur vers un document (pré-) définissant un vocabulaire qui pourra être utilisé dans le document courant
 - **D'un corps :**
 - Liste d'assertions d'éléments, de types simples ou complexes,...

Schémas : Schema, prologue Syntaxe



- `<?xml version="1.0" encoding="ISO-8859-1"?>`
 - Première assertion d'un document XML
- `<xs:schema`
 `xmlns:xs="http://www.w3.org/2001/XMLSchema"`
 `targetNamespace="http://www.lri.fr/~thiam/FCXML"`
 `xmlns="http://www.lri.fr/~thiam/FCXML"`
 `elementFormDefault="qualified" version="1.0">`
- Définition de l'espace de nom **xs** qui permet d'utiliser le vocabulaire prédéfini pour XML schema
 - Le vocabulaire XML schema devra être préfixé par **xs** !

Schémas : Schema, prologue

Détails



- **targetNamespace** indique à quel espace de noms appartiennent les éléments définis dans le schéma.
- **xmlns="..."** indique l'espace de noms par défaut
- **elementFormDefault** indique si les éléments doivent
 - être préfixés : **qualified**
 - non préfixés : **unqualified**par leur espace de noms

Schémas : Schema, corps



- Le corps permet de définir des éléments :
 - `<xs:element name="library" />`
- Un élément peut avoir plusieurs attributs optionnels :
 - **Type="..."** : pour définir des types **simples** ou **complexes**.
- Des restrictions de cardinalités généralisant celles des DTD :
 - `minOccurs="x"` ou `maxOccurs="x"` avec $x \in \mathbb{N}^* \cup \{\text{unbounded}\}$

Type simple et type complexe



- Un **type complexe** est un type qui peut contenir des **déclarations** d'attributs et d'éléments.
- Un **type simple** ne peut pas contenir de déclarations d'attributs ou d'éléments.



TYPES SIMPLES

Définition



- Un type simple est défini par :
 - un **ensemble de valeurs**,
 - une **représentation lexicale**,
 - un ensemble de **facettes** qui caractérise l'ensemble de valeurs.
- Un type simple peut être :
 - un **type atomique**, un **type liste** ou un **type union**,
 - primitif ou dérivé,
 - prédéfini ou défini par l'utilisateur.

Type atomique, type liste ou type union

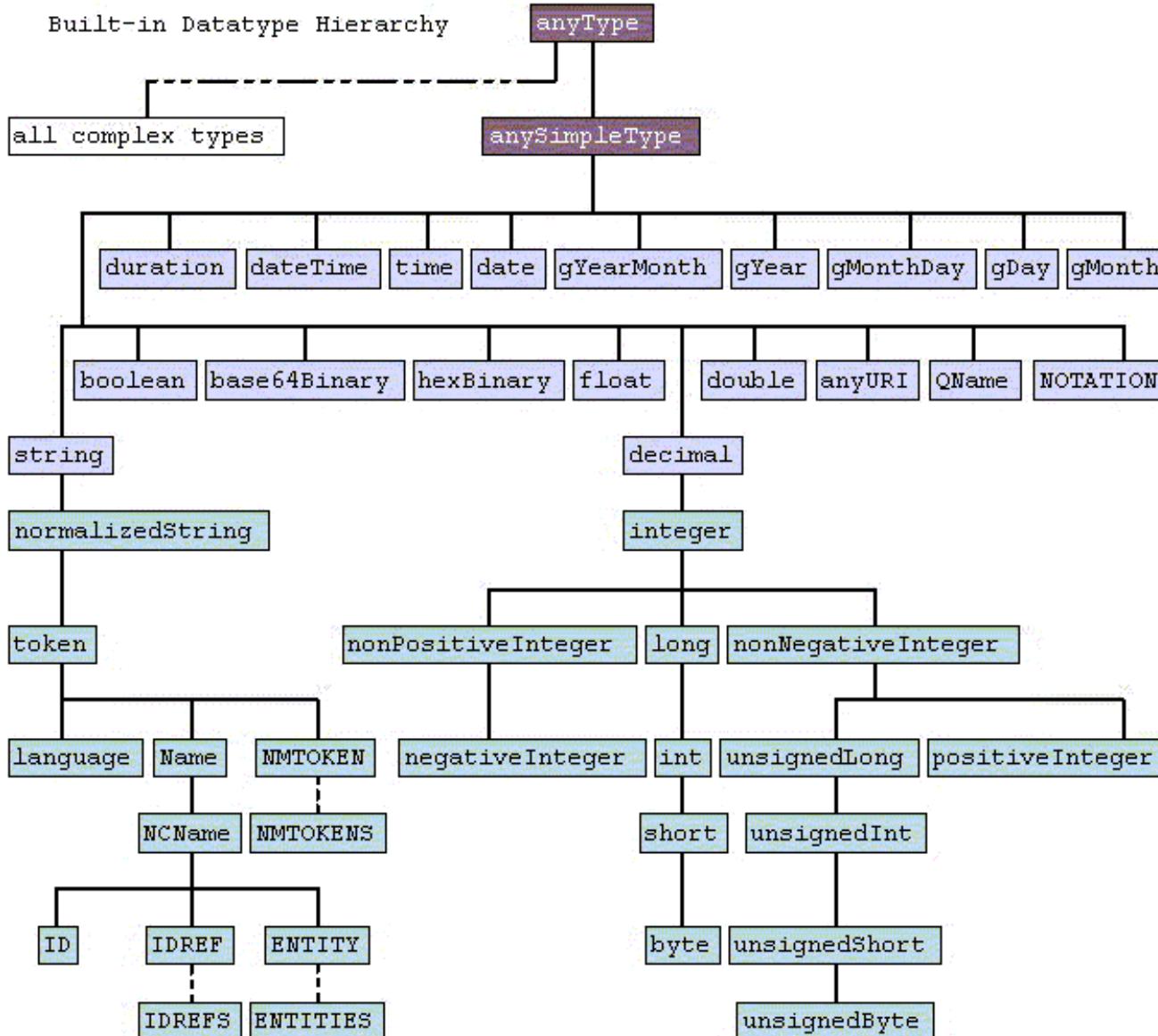


- Un type atomique est un ensemble de valeurs indécomposables dites **valeurs atomiques**.
- Un type liste est un ensemble de séquences de longueur finie de valeurs atomiques.
- Un type union est un ensemble de valeurs atomiques appartenant à plusieurs types atomiques.

Types prédéfinis ou définis par l'utilisateur

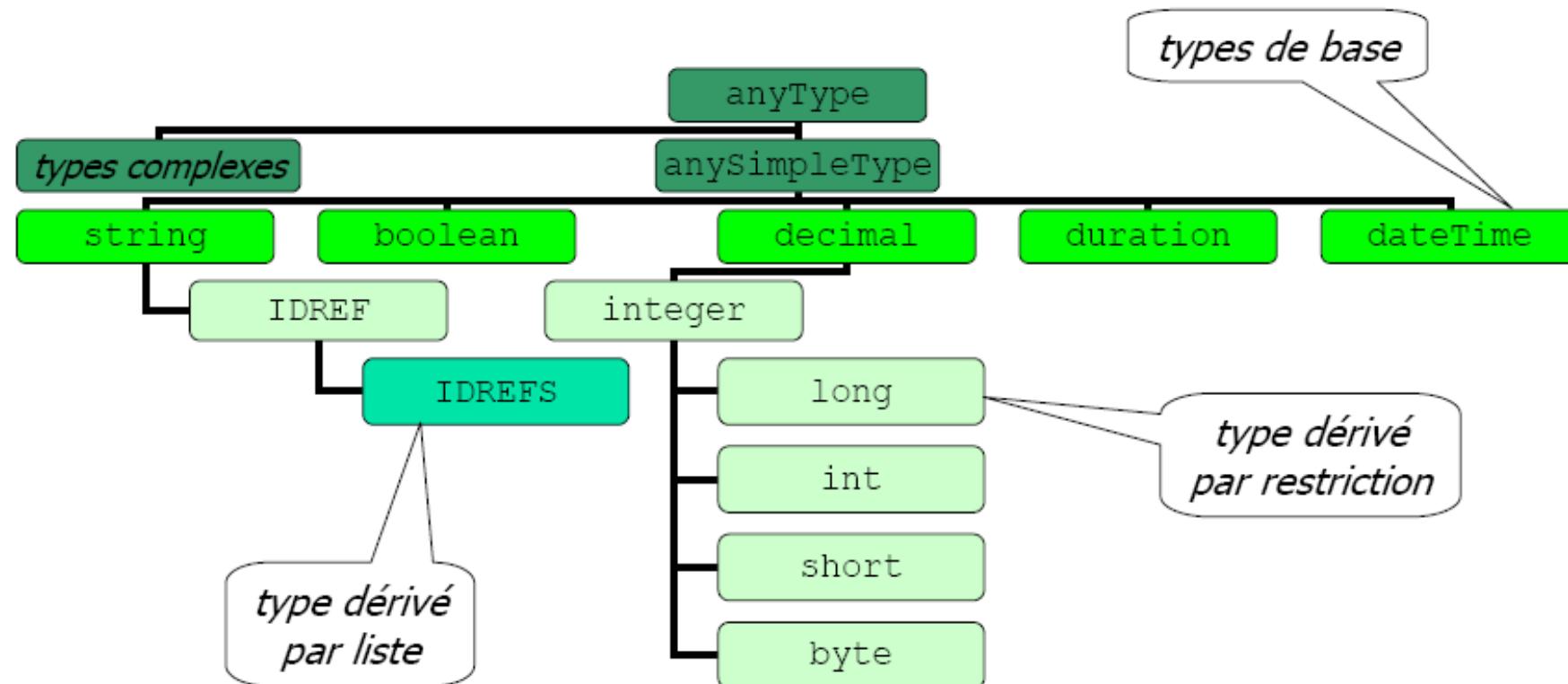


- Un type **prédéfini** peut être primitif ou dérivé.
- Un type peut être **défini** par dérivation de types existants.
- Un type défini peut être **anonyme** ou **nommé**.



- | | | | |
|--|--------------------------|---|--|
| | ur types | | derived by restriction |
| | built-in primitive types | | derived by list |
| | built-in derived types | | derived by extension or
restriction |
| | complex types | | |

Hiérarchie des types prédéfinis : ZOOM

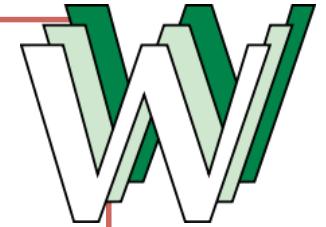


Facettes des types de base : Fondamentales



Datatype	ordered	bounded	cardinality	numeric
<u>string</u>			X	
<u>boolean</u>			X	
<u>float</u>	X	X	X	X
<u>double</u>	X	X	X	X
<u>decimal</u>	X			X
<u>duration</u>	X		X	
<u>dateTime</u>	X		X	

Facettes des types de base : constraining ou non-fondamentales



	string	boolean	decimal	float	duration	dateTime
length	x				x	x
minLength	x					
maxLength	x					
pattern	x	x	x	x	x	x
enumeration	x		x	x	x	x
whiteSpace	x	x	x	x	x	x
minInclusive			x	x	x	x
minExclusive			x	x	x	x
maxInclusive			x	x	x	x
maxExclusives			x	x	x	x
totalDigits			x			
fractionDigits			x			



Types prédéfinis temporels

duration	1 an, 2 mois, 3 jours, 10 heures, 30 minutes : P1Y2M3DT10H30M
dateTime	13h20, le 31 mai 1999 (UTC) : 1999-05-31T13:20 13h20, le 31 mai 1999 (UTC + 5h) : 1999-05-31T13:20:00-05:00
time	13h20 du jour courant (UTC) : 13:20
date	31 mai 1999 : 1999-05-31
gYear	1999 : 1999
gYearMonth	mai 1999 : 1999-05
gMonth	mois de mai de l'année courante : --05--
gMonthDay	31 mai de l'année courante : --05-31
gDay	31 du mois courant : --31

Types de base : string, boolean, float et decimal



string	Ensemble des séquences de caractères de longueur finie	
boolean	Ensemble de valeurs = {true, false, 1, 0}	
float	Ensemble de valeurs = $m \times 2^e$ où $ m \leq 2^{24}$ et $-149 \leq e \leq 104$ + zéro positif et négatif infinité positive et négative pas un nombre	-1E4 1267.43233E12 12.78e-2 12 0 -0 INF -INF NaN
decimal	Nombre décimaux de précision arbitraire. Ensemble de valeurs = $i \times 10^n$ où i et n sont des entiers et $n \geq 0$	-1.23 12678967.543233 +100000.00 210

Types prédéfinis dérivés de decimal: un extrait



integer	Dérivé du type decimal par fractionDigits = 0	-1 0 12678967543233 +100000
long	Dérivé du type integer par minInclusive = -9223372036854775808 et maxInclusive = 9223372036854775807	-1 0 12678967543233 +100000
int	Dérivé du type integer par minInclusive = -2147483648 et maxInclusive = 2147483647	-1 0 126789675 +100000
short	Dérivé du type int par minInclusive = -32768 et maxInclusive = 32767	-1 0 12678 +10000
byte	Dérivé du type short par minInclusive = -128 et maxInclusive = 127	-1 0 126 +100

Types dérivés par restriction



- Intervalle
- Énumération
- motif

Intervalle : Exemple



- Ensemble des altitudes des points de la Terre (le plus haut sommet a une altitude de 8850m).

```
<xsd:simpleType name="altitude">
    <xsd:restriction base="xsd:integer">
        <xsd:minInclusive value="0"/>
        <xsd:maxInclusive value="8850"/>
    </xsd:restriction>
</xsd:simpleType>
```



Enumération : Exemple 1

- Liste des noms des pays africains :

```
<xsd:simpleType name="nom-pays-af">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="Algérie"/>
        <xsd:enumeration value="Sénégal"/>
        <xsd:enumeration value="Congo Brazzaville"/>
        <xsd:enumeration value="Gambie"/>
        <xsd:enumeration value="Mauritanie"/>
        ...
    </xsd:restriction>
</xsd:simpleType>
```



Enumération : Exemple 2

- Liste des codes des pays :

```
<xsd:simpleType name="code-pays-af">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="AL"/>
    <xsd:enumeration value="SN"/>
    <xsd:enumeration value="CGB"/>
    <xsd:enumeration value="GB"/>
    <xsd:enumeration value="MAU"/>
    ...
  </xsd:restriction>
</xsd:simpleType>
```

Motif : Exemple 1



- Différents motifs de ISBN à 10 chiffres

```
<xsd:simpleType name="ISBN">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="\d-\d{2}-\d{6}[\dx]" />
    <xsd:pattern value="\d-\d{3}-\d{5}[\dx]" />
    <xsd:pattern value="\d-\d{4}-\d{4}[\dx]" />
    <xsd:pattern value="\d-\d{5}-\d{3}[\dx]" />
  </xsd:restriction>
</xsd:simpleType>
```

Motif : Exemple 2



- Différentes formes d'un nom XML :

```
<xsd:simpleType name="Identifier">
    <xsd:restriction base="xsd:string">
        <xsd:pattern value="[_A-Za-z][._:_0-9A-Za-z]*"/>
    </xsd:restriction>
</xsd:simpleType>
```



Types liste

- L'opérateur **xsd:list** définit un nouveau type simple
- valeurs sont les listes de valeurs du type simple donné par l'attribut **itemType**.
- pas de listes générales (# langages de programmation)
- Uniquement listes de valeurs séparées par des espaces
- Souvent utilisées comme valeurs d'attributs.
- Pas de caractères d'espacement dans type simple donné par **itemType**



Exemple 1

- Liste de pays :

```
<xsd:simpleType name="liste-pays">  
    <xsd:list itemType="nom-pays-af"/>  
</xsd:simpleType>
```

- Contenu conforme à ce type :

```
<pays-africains>  
    Sénégal Gambie Mauritanie  
</pays-africains>
```



Exemple 2

- Type pour liste d' entiers

```
<xsd:simpleType name="IntList">
    <xsd:list itemType="xsd:integer"/>
</xsd:simpleType>
```

- Type pour liste de 5 entiers

```
<xsd:simpleType name="IntList5">
    <xsd:restriction base="IntList">
        <xsd:length value="5"/>
    </xsd:restriction>
</xsd:simpleType>
```



Types union

- L'opérateur **xsd:union** définit un nouveau type simple dont les valeurs sont celles des types listés dans l'attribut **memberTypes**.



Exemple 1

- Ensemble des noms ou des codes des pays :

```
<xsd:simpleType name="nom-ou-code-pays">
    <xsd:union memberTypes="nom-pays-af code-pays-af"/>
</xsd:simpleType>
```

- Eléments dont le contenu est conforme à ce type :

```
<pays>AL</pays>
<pays>Algerie</pays>
```



Exemple 2

- Définition du type **Unbounded**

```
<xsd:simpleType name="Unbounded">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="unbounded"/>
    </xsd:restriction>
</xsd:simpleType>
```

- Définition du type **IntegerOrUnbounded**

```
<xsd:simpleType name="IntegerOrUnbounded">
    <xsd:union memberTypes="xsd:nonNegativeInteger
        Unbounded"/>
</xsd:simpleType>
```



TYPES COMPLEXES



Type complexe

- Un type complexe définit des compositions d'éléments et/ou d'attributs :

`<xsd:complexType ...>`

Suite de déclarations d'attribut, d'élément ou de composition d'éléments

`</xsd:complexType>`

- Une composition d'éléments est réalisée à l'aide de 3 constructeurs:
 - **sequence** : séquence d'éléments typés,
 - **choice** : un élément parmi une liste d'éléments possibles,
 - **all** : tas d'éléments typés.



Déclaration d'un élément

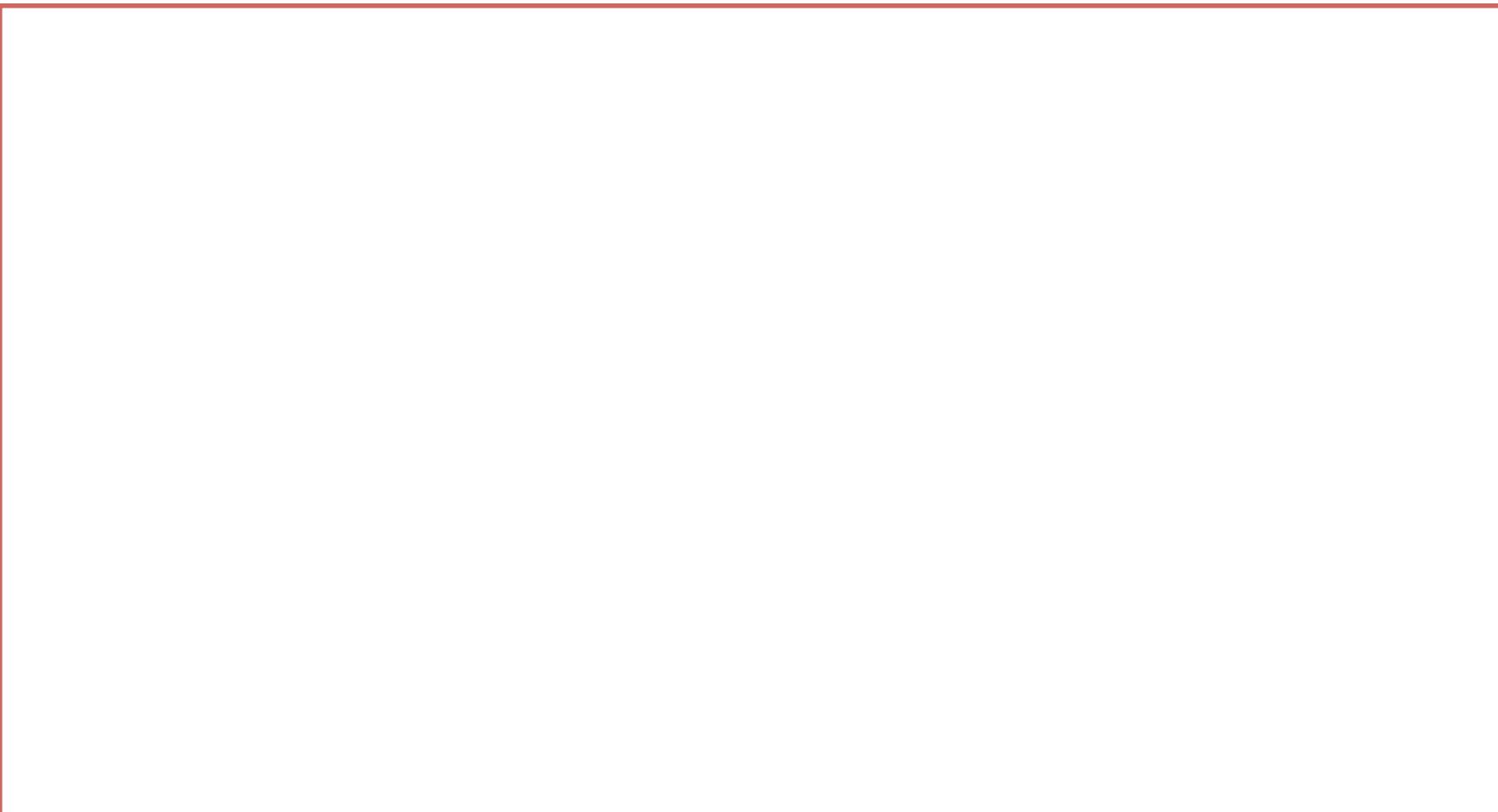
- Un élément dans la définition d'un type complexe est spécifié sous la forme d'un élément **xsd:element** dont les attributs peuvent être :
 - *name* : nom de l'élément,
 - *type* : type de l'élément,
 - *ref* : référence à un nom d'élément déclaré au niveau global (**réutilisation**),
 - *minOccurs* : nombre minimum d'occurrences de l'élément (1 par défaut),
 - *maxOccurs* : nombre maximum d'occurrences de l'élément (1 par défaut),
 - ...
 - le contenu est la définition d'un type.



Déclaration d'un attribut

- Un attribut dans la définition d'un type complexe est spécifié sous la forme d'un élément **xsd:attribute** dont les attributs peuvent être :
 - *name* : nom de l'attribut,
 - *ref* : référence à un nom d'attribut déclaré au niveau global (**réutilisation**),
 - *type* : type des valeurs de l'attribut,
 - *use* : required, optional (par défaut), ...
 - *default* : valeur par défaut,
 - *fixed* : valeur fixée,
 - ...
 - le contenu peut être : un type simple.

Éléments à contenu simple



Elément à contenu simple : exemple



- L'élément :

```
<pays>Gambie</pays>
```

- peut être déclaré :

```
<xsd:element name="pays" type="mns:nom-pays-af"/>
```

Séquence d'éléments : type anonyme



- L'élément : <**livre**>

```
    <titre>Programmer en XML</titre>
    <année>2004</année>
</livre>
```

- peut être déclaré :

```
<xsd:element name="livre">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="titre" type="xsd:string"/>
      <xsd:element name="année" type="xsd:gYear"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Eléments et attributs : type nommé



- L'élément :

```
<livre edition="2">
    <titre>Programmer en XML</titre>
    <année>2004</année>
</livre>
```
- peut être déclaré :

```
<xsd:element name="livre">
    <xsd:complexType name="livre">
        <xsd:sequence>
            <xsd:element name="titre" type="xsd:string"/>
            <xsd:element name="année" type="xsd:gYear"/>
        </xsd:sequence>
        <xsd:attribute name="édition" type="xsd:positiveInteger"/>
    </xsd:complexType>
</xsd:element>
```



Éléments à contenu mixte

- *pur* lorsqu'il ne contient que des éléments qui, eux-mêmes, peuvent à leur tour contenir du texte et/ou des éléments.
- *mixte* lorsqu'il contient du texte autre que des caractères d'espacement en dehors de ses enfants.
- attribut *mixed* de l'élément **xsd:complexType** contient la valeur true.

Exemple



- L'élément :

```
<titre>  
    La logique <sigle>FOL</sigle> est peu expressive  
</titre>
```

- peut être déclaré :

```
<xsd:element name="titre">  
    <xsd:complexType mixed="true">  
        <xsd:sequence>  
            <xsd:element name="sigle" type="xsd:string"/>  
        </xsd:sequence>  
    </xsd:complexType>  
</xsd:element>
```



Elément à contenu vide

- L'élément :

```
<prix monnaie="euros" valeur="25.5"/>
```

- peut être déclaré :

```
<xsd:element name="prix">
  <xsd:complexType>
    <xsd:attribute name="monnaie" type="xsd:string"/>
    <xsd:attribute name="valeur" type="xsd:decimal"/>
  </xsd:complexType>
</xsd:element>
```



Type nommé

- On peut nommer le type définissant un prix en euros ou en dollars :

```
<xsd:complexType name="prix-international">  
    <xsd:attribute name="monnaie" type="xsd:string"/>  
    <xsd:attribute name="valeur" type="xsd:decimal"/>  
</xsd:complexType>
```

- et y faire référence dans la déclaration de l'élément prix :

```
<xsd:element name="prix" type="prix-international"/>
```



Elément à contenu alternatif

- Les éléments :

```
<prix><euros>25.5</euros></prix>
```

```
<prix><dollars>28.75</dollars></prix>
```

- peuvent être déclarés :

```
<xsd:element name="prix">
  <xsd:complexType>
    <xsd:choice>
      <xsd:element name="euros" type="xsd:decimal"/>
      <xsd:element name="dollars" type="xsd:decimal"/>
    </xsd:choice>
  </xsd:complexType>
</xsd:element>
```

Opérateur d'ensemble : Elément contenu non ordonné



- Les éléments doivent apparaître une fois dans un ordre quelconque
- Pas d'équivalent en DTD

```
<xsd:element name="book">
  <xsd:complexType>
    <xsd:all>
      <xsd:element name="title" type="xsd:string"/>
      <xsd:element name="author" type="xsd:string"/>
      <xsd:element name="year" type="xsd:string"/>
      <xsd:element name="publisher" type="xsd:string"/>
    </xsd:all>
  </xsd:complexType>
</xsd:element>
```

Exemple



```
<?xml version="1.0" encoding="iso-8859-1"?>
<bibliography>
  <book key="Michard01" lang="fr">
    <author>Alain Michard</author>
    <title>XML langage et applications</title>
    <publisher>Eyrolles</publisher>
    <year>2001</year>
  </book>
  <book key="Zeldman03" lang="en">
    <title>Designing with web standards</title>
    <author>Jeffrey Zeldman</author>
    <year>2003</year>
    <publisher>New Riders</publisher>
  </book>
  ...
</bibliography>
```



Groupe d'éléments

- La séquence d'éléments :

```
<prenom>Jean</prenom><nom>Dupont</nom>
```

- peut être déclarée :

```
<xsd:group name="prenom-nom">
  <xsd:sequence>
    <xsd:element name="prenom" type="xsd:string"/>
    <xsd:element name="nom" type="xsd:string"/>
  </xsd:sequence>
</xsd:group>
```

- (On peut aussi grouper des attributs)

Utilisation des groupes d'éléments



- Pour déclarer des éléments décrivant des personnes décrites par leur nom et leurs prénoms, on peut définir le type suivant :

```
<xsd:complexType name="personne">  
    <xsd:group ref="prenom-nom"/>  
</xsd:complexType>
```

Dérivation d'un type complexe par extension



- A partir du type personne, on peut définir le type chercheur en ajoutant au prénom et au nom, le laboratoire et le pays :

```
<xsd:complexType name="chercheur">
  <xsd:complexContent>
    <xsd:extension base="personne">
      <xsd:sequence>
        <xsd:element name="laboratoire" type="xsd:string"/>
        <xsd:element ref="pays">
      </xsd:sequence>
    </xsd:extension>
  </complexContent>
</xsd:complexType>
```

Schéma XML



- Un schéma XML est une suite de définitions de types et de déclarations d'attributs.



Exemple du document ...

```
<livres> ...
  <livre edition="1">
    <titre>Le langage XML</titre>
    <auteur>
      <prenom>Jean</prenom><nom>Dupont</nom>
      <laboratoire>LRI</laboratoire><pays>France</pays>
    </auteur>
    <auteur>
      <prenom>Pierre</prenom><nom>Durand</nom>
      <laboratoire>LRI</laboratoire><pays>France</pays>
    </auteur>
    <année>2004</année>
    <prix monnaie="euros" valeur="25.5"/>
  </livre>...
</livres>
```

DTD



```
<!ELEMENT livres (livre*)>
<!ELEMENT livre (titre, auteur+, année, prix)>
<!ATTLIST livre edition CDATA #REQUIRED>
<!ELEMENT titre (#PCDATA)>
<!ELEMENT auteur (prenom, nom, laboratoire, pays)>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT prenom (#PCDATA)>
<!ELEMENT laboratoire (#PCDATA)>
<!ELEMENT année (#PCDATA)>
<!ELEMENT prix EMPTY>
<!ATTLIST prix monnaie CDATA #REQUIRED valeur CDATA #REQUIRED>
<!ELEMENT pays (#PCDATA)>
```



Schéma XML : exemple

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:group name="prenom-nom">
    <xsd:sequence>
        <xsd:element name="prenom" type="xsd:string"/>
        <xsd:element name="nom" type="xsd:string"/>
    </xsd:sequence>
</xsd:group>
<xsd:complexType name="personne">
    <xsd:group ref="prenom-nom"/>
</xsd:complexType>
...
...
```



Schéma XML : exemple

```
...
<xsd:simpleType name="nom-pays">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="Algerie"/>
        <xsd:enumeration value="Australie"/>
        <xsd:enumeration value="Belgique"/>
        <xsd:enumeration value="France"/>
        ...
    </xsd:restriction>
</xsd:simpleType>

<xsd:element name="pays" type="nom-pays"/>
...
```

Schéma XML : exemple



```
<xsd:complexType name="chercheur">
  <xsd:complexContent>
    <xsd:extension base="personne">
      <xsd:sequence>
        <xsd:element name="laboratoire" type="xsd:string"/>
        <xsd:element ref="pays">
      </xsd:sequence>
    </xsd:extension>
  </complexContent>
</xsd:complexType>

<xsd:complexType name="prix-international">
  <xsd:attribute name="monnaie" type="xsd:string"/>
  <xsd:attribute name="valeur" type="xsd:decimal"/>
</xsd:complexType> ...
```



Schéma XML : exemple

```
...
<xsd:element name="livre">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="titre" type="xsd:string"/>
      <xsd:element name="auteur" type="chercheur"
                    maxOccurs="unbounded"/>
      <xsd:element name="année" type="xsd:gYear"/>
      <xsd:element name="prix"
                    type="prix-international"/>
    </xsd:sequence>
    <xsd:attribute name="édition" type="xsd:positiveInteger"/>
  </xsd:complexType>
</xsd:element>
...
```

Schéma XML : exemple



```
...
<xsd:element name="livres">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="livre" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
</xsd:schema>
```

END



```
<?xml version="1.0" encoding="UTF-8" ?>
<Semestre>
  <UE ref="1">
    <Intitule>Informatique</Intitule>
    <EC>
      <Titre>Développement Web</Titre>
      <Duree>25</Duree>
      <Enseignant>Pr Togobaye Diawara</
      Enseignant>
      <Etudiants>
        <Etudiant num="X9021">
          <Bac>Sciences</Bac>
          <Note>14</Note>
        </Etudiant>
        <Etudiant num="X9022">
          <Bac>Technique</Bac>
        </Etudiant>
        </Etudiants>
      </EC>
      <EC>
        <Titre>Systèmes Distribués</Titre>
        <Duree>Vingt heures</Duree>
        <Enseignant>Pr Yaya Siby</Enseignant>
```

```
<Etudiants>
  <Etudiant num="X9023">
    <Note>13</Note>
    <Bac> Mathématiques </Bac>
  </Etudiant>
</Etudiants>
</EC>
</UE>
<UE ref="2">
  <Intitule>Réseaux Télécoms</Intitule>
  <EC>
    <Titre>Programmation réseau</Titre>
    <Duree>15</Duree>
    <Enseignant>Pr Cheikh Souare</
    Enseignant>
    <Etudiants>
      <Etudiant num="X9024">
        <Note>17</Note>
        <Bac>Mathématiques</Bac>
      </Etudiant>
    </Etudiants>
  </EC>
</UE>
</Semestre>
```



XML et ses applications : plan

1. Introduction à XML
2. Schémas de documents XML
 - a. DTD
 - b. XML Schema
3. Le modèle DOM
4. Interrogation : XPath
5. Transformation et présentation : XSLT
6. Programmation : XML et Java (API DOM, SAX, ...)
7. Application : intégration de données



DOCUMENT OBJECT MODEL

Introduction

- **DOM** standard du W3C qui décrit interface indépendante
 - langage de programmation
 - plate-forme
 - Permettant aux programmes informatiques et scripts d'accéder ou de m-a-j (contenu, structure ou style) document XML et HTML
- Possibilité de traiter le document et réincorporer résultats des traitements dans le document tel qu'il sera présenté

Évolution

- DOM 1
 - 1998 - Core + Représentation sous la forme d'arbre
- DOM 2
 - 2000 - *Core, Events, Style, View et Traversal and Range*
- DOM 3
 - 2004 - Support **Xpath, clavier et interface de sérialisation de documents XML**
- DOM 4 – draft depuis février 2014

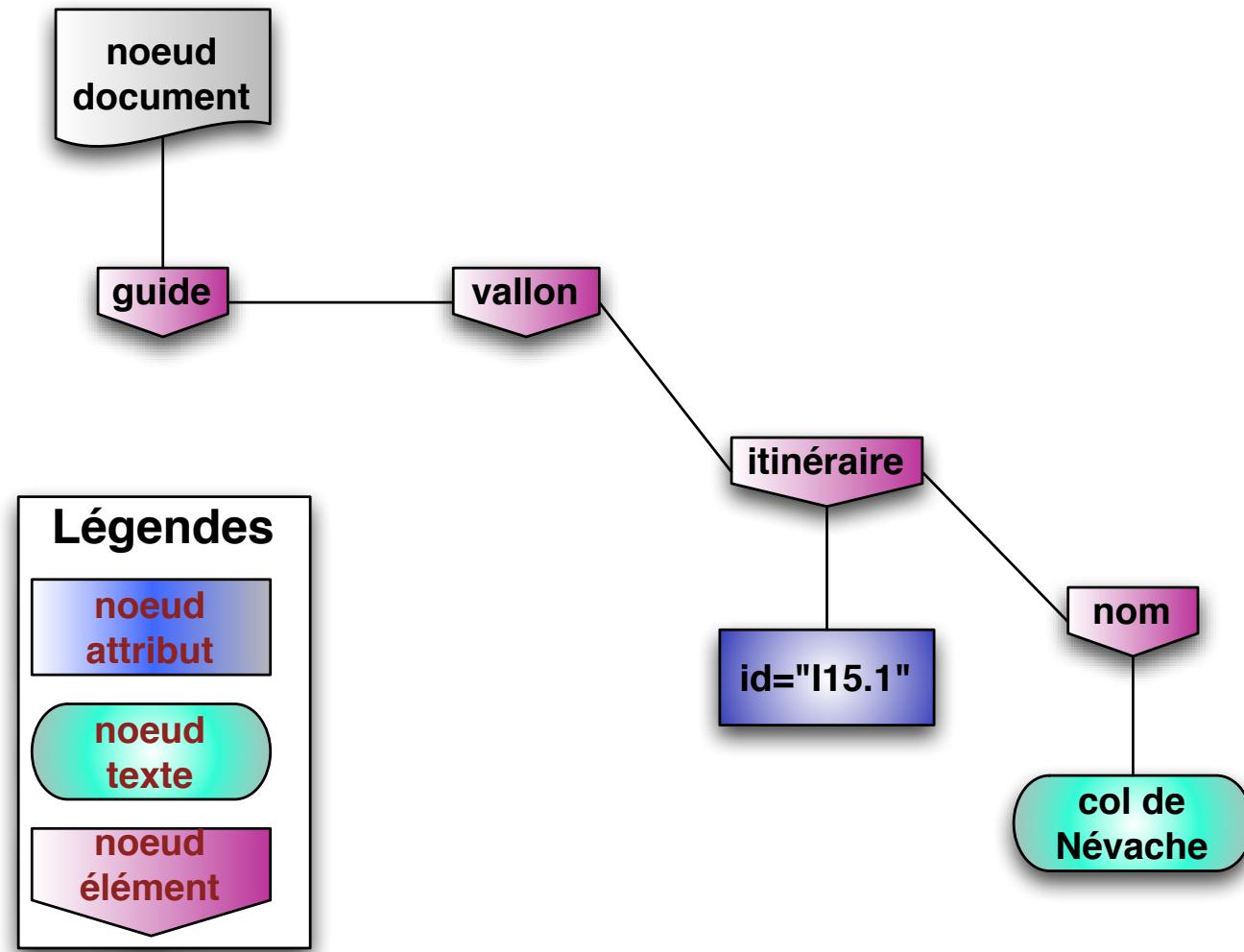
Évènements capturant du DOM

- Page et fenêtre
 - *Onabort, onerror, onload, onresize, etc ...*
- Souris
 - *Onclick, ondblclick, onmousedown, etc ...*
- Clavier
 - *Onkeydown, onkeypress, onkeyup*
- Formulaire
 - *Onblur, onchange, onfocus, onreset, onselect, etc..*

Représentation d'un document XML

- Les principaux langages de manipulation de documents XML :
 - **XPath**,
 - **XSLT**,
 - **XQuery**...
 - opèrent sur un modèle de données commun : le modèle **XDM** (XML Data Model)
- Dans ce modèle, un document XML est représenté sous forme d'un **arbre de document**.

Arbre du document itinéraire



Sortes de noeuds

- L'arbre d'un document comporte **7** sortes de noeuds associés à chaque constituant d'un document
 - **document** qui est le noeud racine,
 - **élément**,
 - **texte**,
 - **attribut**,
 - **espace de noms**,
 - **commentaire**,
 - **instruction de traitement**.

Relations entre noeuds (1)

● Parents

- Tous noeuds, sauf le noeud ***document***, ont un **père** (*parent*) qui est un noeud élément.
- Le noeud ***document*** doit avoir
 - un seul noeud ***fils*** qui est de sorte **élément**
 - autres noeuds (probablement) ***fils*** qui sont de sorte **commentaire** ou **instruction de traitement**.

Relations entre noeuds (2)

- Éléments

- Un noeud **élément** peut avoir des noeuds **fils** qui peuvent être de sorte
 - *attribut,*
 - *élément,*
 - *texte,*
 - *espace de noms,*
 - *commentaire ou*
 - *instruction de traitement.*

Relations entre noeuds (3)

- **Fils**
 - Les noeuds **fils** d'un noeud **document** ou **élément** qui sont de sorte
 - élément,
 - texte,
 - commentaire ou
 - instruction de traitement
 - sont appelés les **enfants (*children*)** de ce noeud.
 - Les enfants d'un noeud ne peuvent donc être ni des noeuds attribut, ni des noeuds espace de noms.

Ordre du document

- Les noeuds de l'arbre d'un document sont ordonnés
 - **l'ordre du document** = ordre de lecture, dans le document XML, des constituants représentés par chaque noeud.
- Cet ordre correspond à un parcours **préfixe** de l'arbre du document.

Identité d'un noeud

- Tout noeud a un identificateur unique différent des identificateurs des autres noeuds.
- Un noeud ne doit pas avoir d'attributs ou d'enfants qui ont la même identité.

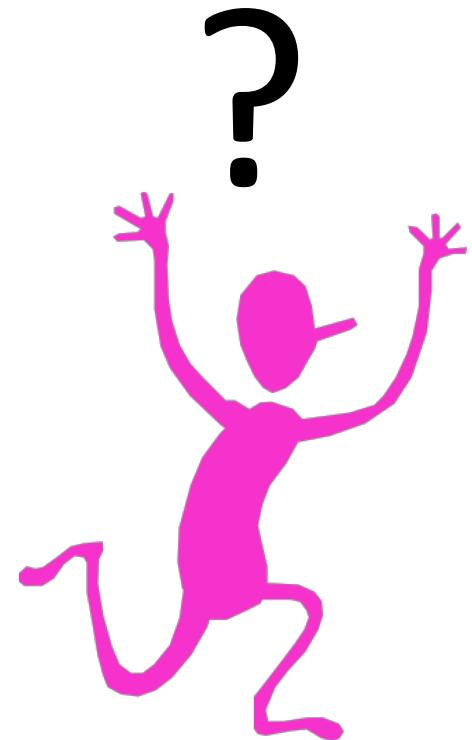
Contraintes sur les noeuds texte

- Un noeud ne doit pas avoir
 - 2 enfants consécutifs qui sont des noeuds **texte**.
 - d'enfants de sorte **texte** dont le contenu est vide.

Valeurs textuelles

- Tout noeud a une **valeur textuelle** :
 - noeud **document** = concaténation des valeurs textuelles de ses descendants de sorte texte dans l'ordre du document,
 - noeud **élément** = concaténation des valeurs textuelles de ses descendants de sorte texte dans l'ordre du document,
 - noeud **texte** = chaîne de caractères constituant ce texte contenu dans ce noeud,
 - noeud **attribut** = chaîne de caractères constituant la valeur de cet attribut.

END





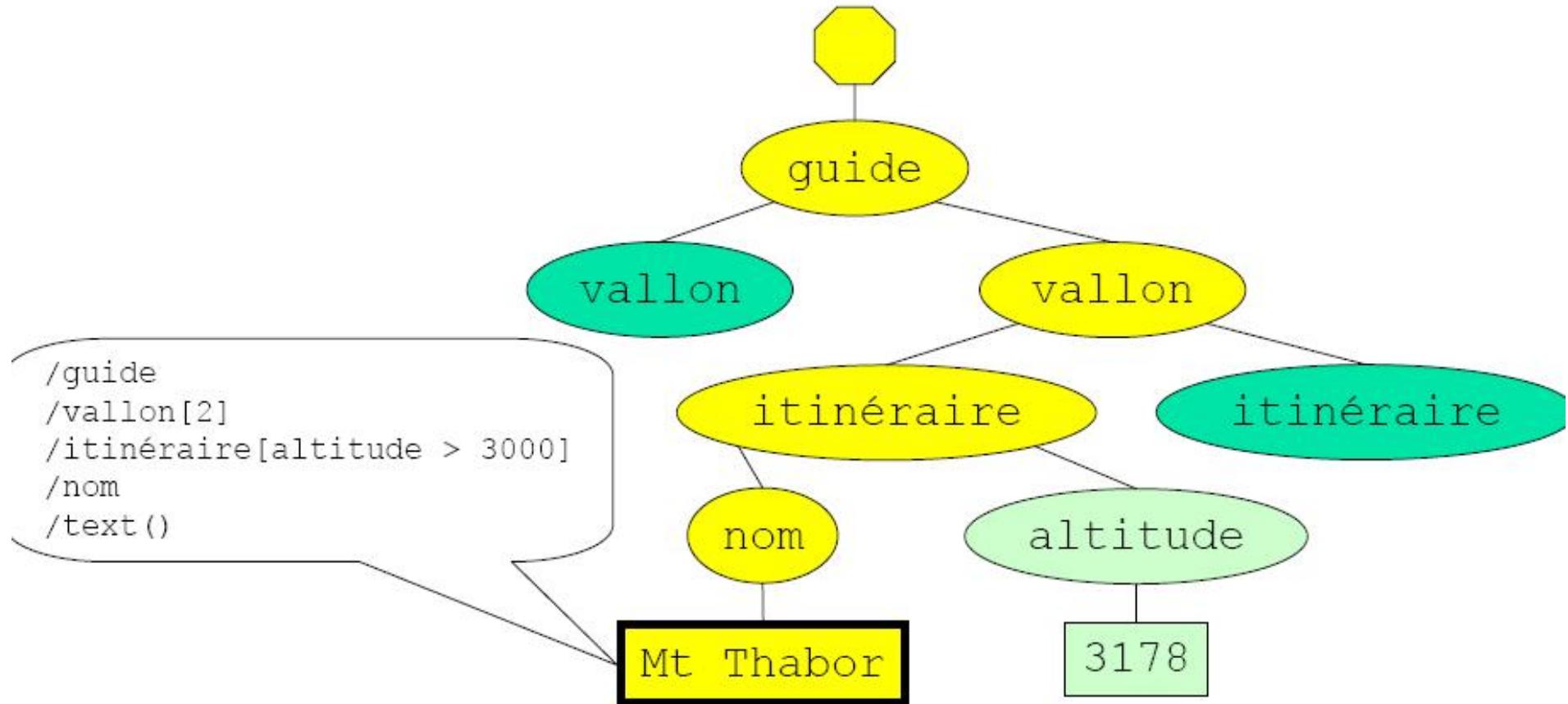
XML et ses applications : plan

1. Introduction à XML
2. Schémas de documents XML
 - a. DTD
 - b. XML Schema
3. Le modèle DOM
4. Interrogation : XPath
5. Transformation et présentation : XSLT
6. Programmation : XML et Java (API DOM, SAX, ...)
7. Application : intégration de données

Introduction

- Le langage XPath opère sur l'arbre d'un document.
- Le langage XPath permet de sélectionner l'ensemble des noeuds que l'on peut atteindre en suivant, à partir d'un noeud donné de l'arbre d'un document, tous les chemins conformes à un modèle appelé **chemin de localisation**.
- Le langage XPath est un langage fonctionnel et donc un langage d'expressions.

Exemple de path (chemin de localisation)



Définition

- Une **expression XPath** a une valeur qui a l'un des 4 types suivants :
 - **ensemble de noeuds**,
 - **chaîne** de caractères UCS,
 - **nombre**,
 - **booléen**

Construction (1)

- Une **expression XPath** est construite à partir :
 - de constantes littérales de type chaîne de caractères ou nombre,
 - de noms de variable,
 - de chemins de localisation (*location path*),
 - d'opérateurs sur les ensembles de noeuds :
parcours de chemin, filtrage, union, ...

Construction (2)

- Une **expression XPath** est construite à partir :
 - d'opérateurs arithmétiques : **+ - * mod div ...**
 - d'opérateurs de comparaison : **= != > >= < <= ...**
 - de connecteurs logiques : **and, or, not ...**
 - d'opérateurs sur les chaînes de caractères :
contains, ...
 - d'appels de fonctions prédéfinies.

Contexte d'évaluation

- Une expression XPath est évaluée dans un **contexte** constitué par :
 - un noeud : le **noeud contexte**,
 - deux entiers :
 - ✓ la **position contexte** et
 - ✓ la **taille contexte**,
 - une bibliothèque de fonctions prédéfinies,
 - les déclarations d'espaces de noms visibles dans l'expression.

Chemin de localisation (1)

- **chemin de localisation** = expression qui représente un modèle de chemin.
- Deux (2) types : **absolu** ou **relatif**.
- chemin de localisation **relatif** commence au noeud contexte et est constitué d'une suite de pas de localisation.
- **Syntaxe :**
 - *pas de localisation₁/.../pas de localisation_n*

Chemin de localisation (2)

- Un chemin de localisation **absolu** est un chemin de localisation relatif qui commence à la racine du document.
- **Syntaxe :**
 - //
 - *chemin de localisation relatif*
- La valeur d'un chemin de localisation est l'ensemble des noeuds extrémité des chemins conforme à ce modèle.

Pas de localisation

- Un **pas de localisation** est caractérisé par :
 - un **axe**,
 - un **test de noeud**,
 - une suite éventuellement vide de **prédicts**.
- **Syntaxe** :
 - *axe::test de noeud prédict₁... prédict_n*

Axes de localisation

- Un axe sélectionne, dans l'arbre du document et à partir du noeud contexte, l'ensemble des noeuds qui peuvent être atteints en suivant une certaine direction.
- On distingue 13 axes :
 - **self, child, descendant, descendant-or-self, parent, ancestor, ancestor-or-self, following-sibling, preceding-sibling, following, preceding, attribute, namespace.**

Sens des axes de localisation

- Un axe a un **sens** : **avant ou arrière**.
 - **Avant** : noeuds sont soit le noeud contexte, soit des noeuds **qui suivent** le noeud contexte dans l'ordre du document.
 - **Arrière** : noeuds sont soit le noeud contexte, soit des noeuds **qui précèdent** le noeud contexte dans l'ordre du document.
- Un axe a une **sorte de noeud principal (element, attribute, ...)**

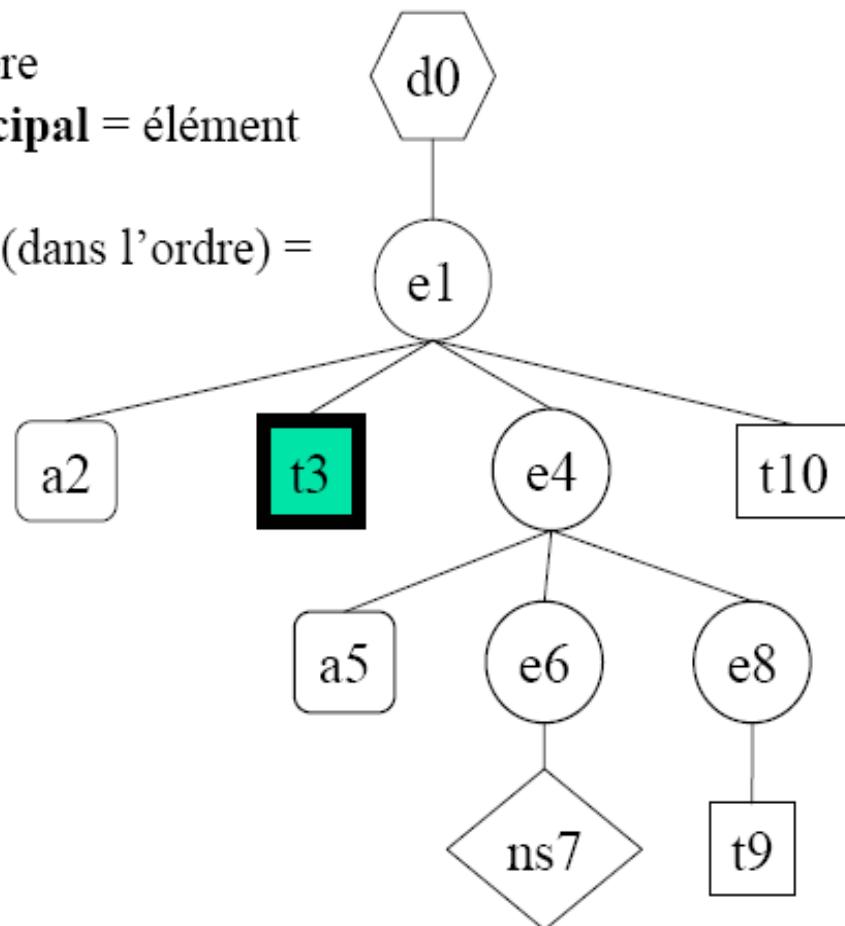
Axe : self

Sens = avant ou arrière

Sorte de nœud principal = élément

Noeud contexte = e1

Nœuds sélectionnés (dans l'ordre) =
e1



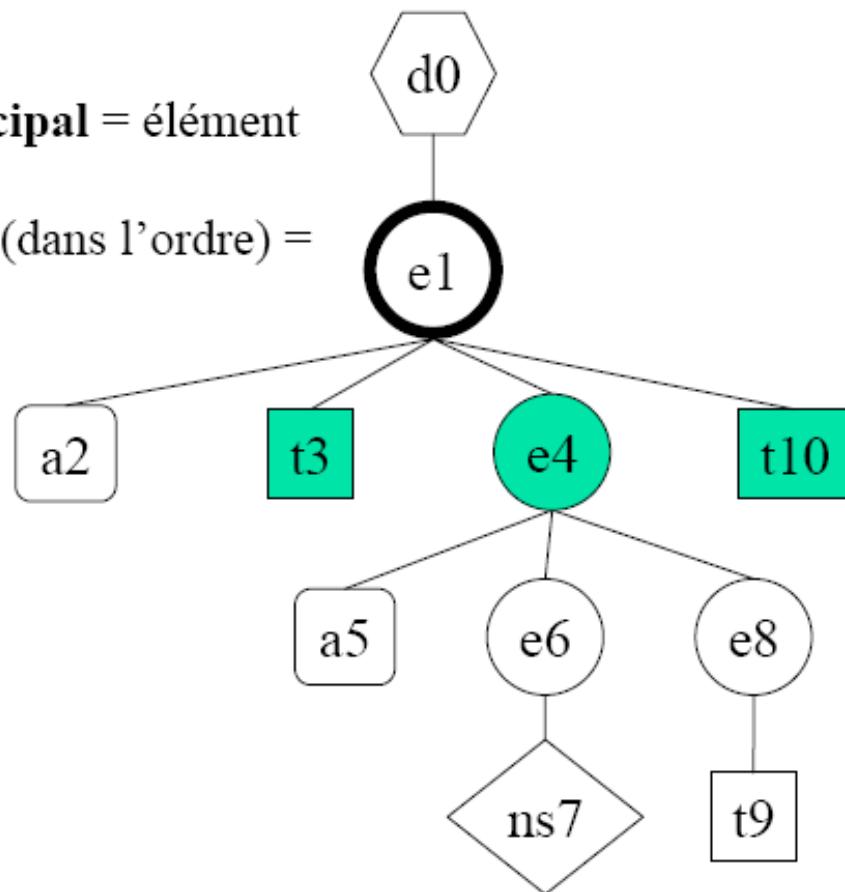
Axe : child

Sens = avant

Sorte de nœud principal = élément

Noeud contexte = e1

Nœuds sélectionnés (dans l'ordre) =
t3, e4, t10



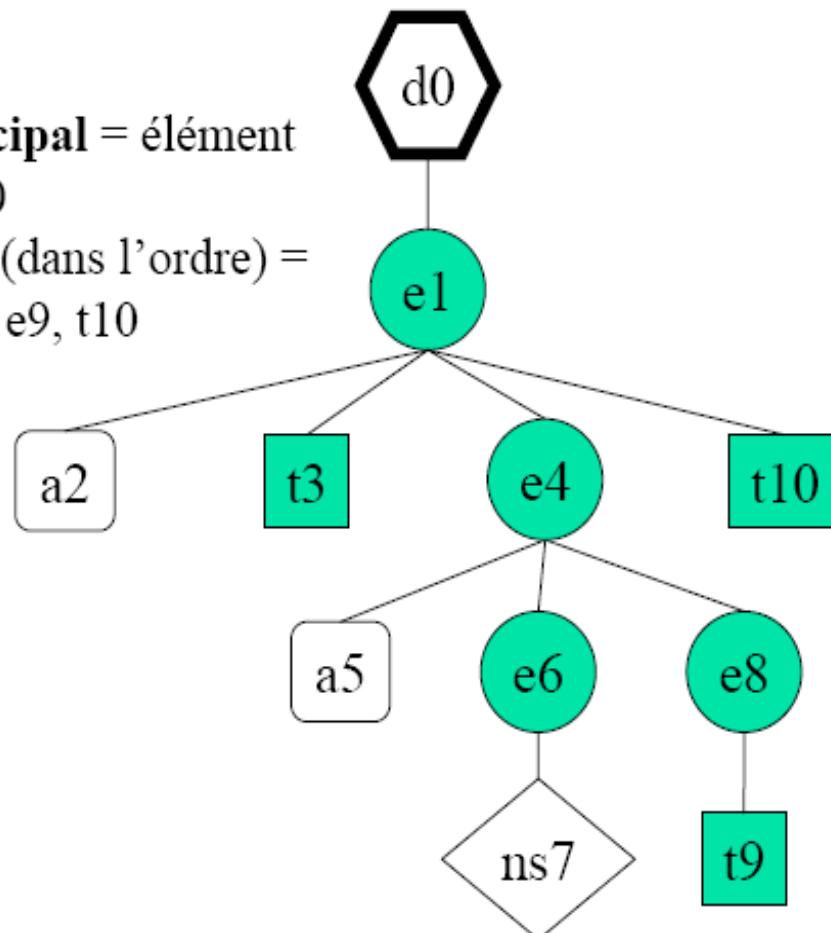
Axe : descendant

Sens = avant

Sorte de nœud principal = élément

Noeud contexte = d0

Nœuds sélectionnés (dans l'ordre) =
e1, t3, e4, e6, e8, e9, t10



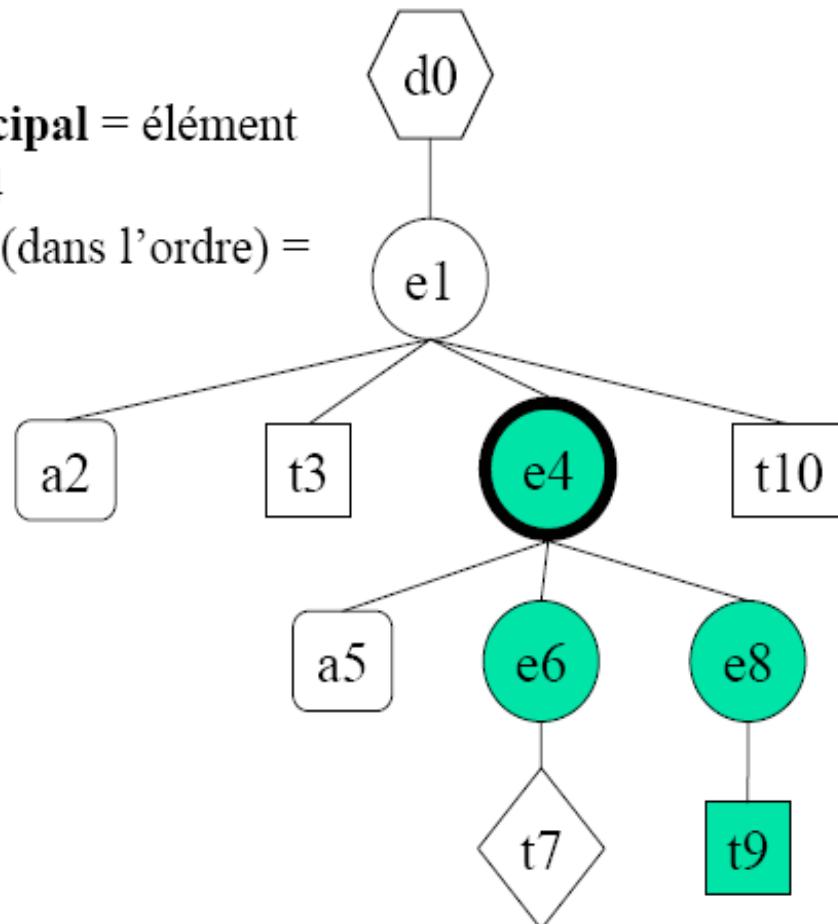
Axe : descendant-or-self

Sens = avant

Sorte de nœud principal = élément

Noeud contexte = e4

Nœuds sélectionnés (dans l'ordre) =
e4, e6, e8, e9



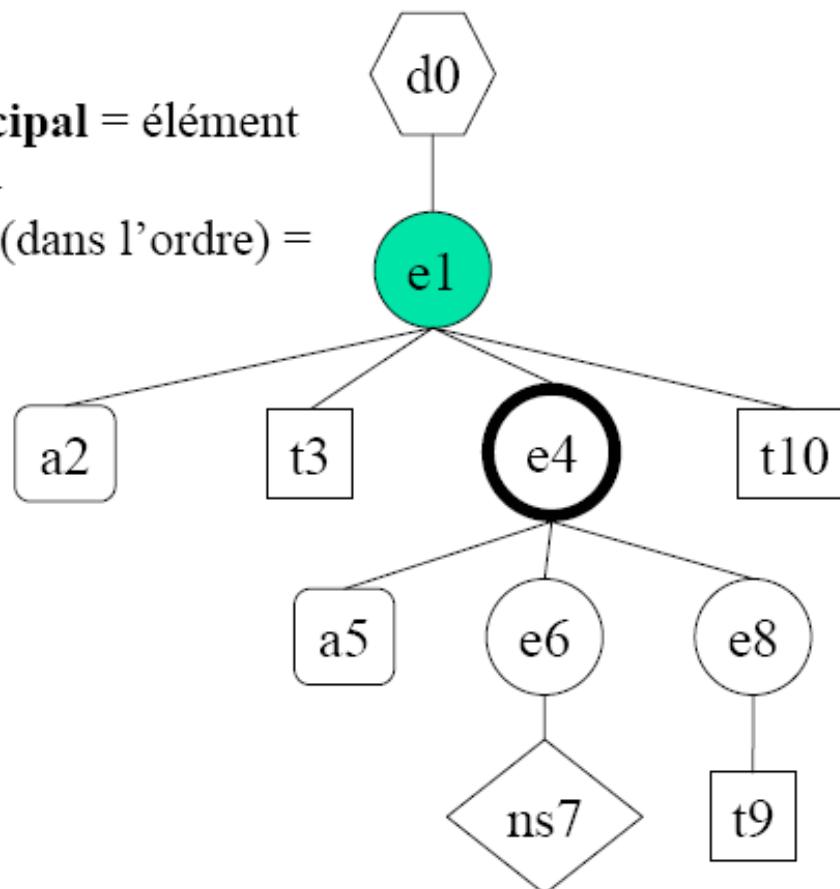
Axe : parent

Sens = arrière

Sorte de nœud principal = élément

Noeud contexte = e4

Nœuds sélectionnés (dans l'ordre) =
e1



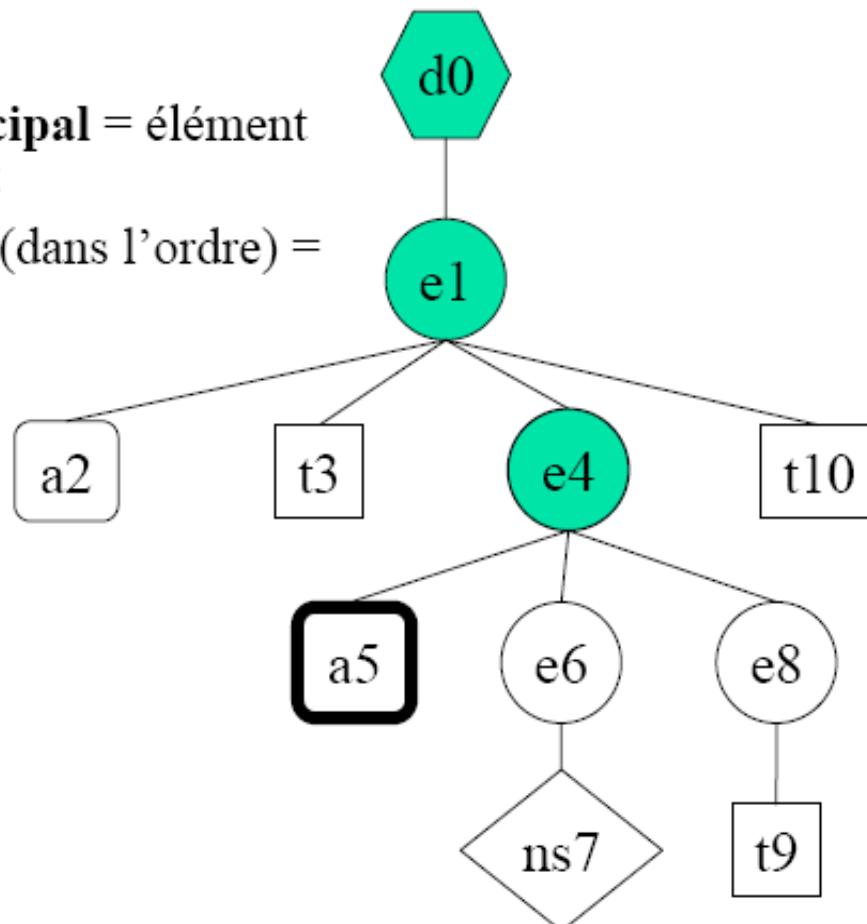
Axe : ancestor

Sens = arrière

Sorte de nœud principal = élément

Noeud contexte = a5

Nœuds sélectionnés (dans l'ordre) =
e4, e1, d0



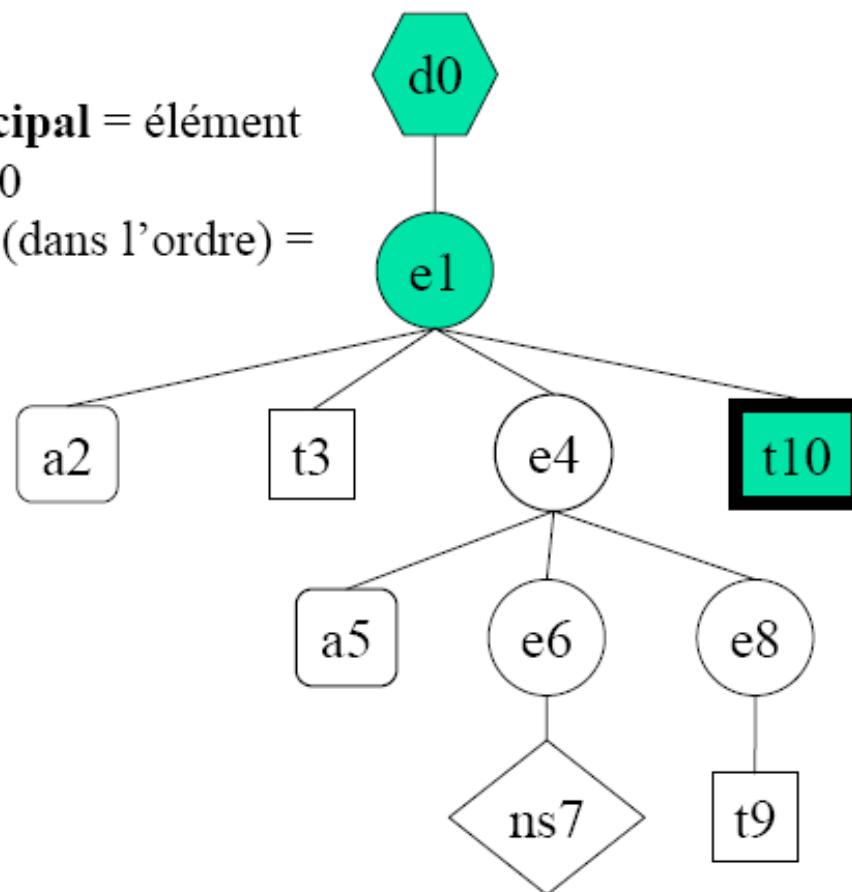
Axe : ancestor-or-self

Sens = arrière

Sorte de nœud principal = élément

Noeud contexte = t10

Nœuds sélectionnés (dans l'ordre) =
t10, e1, d0



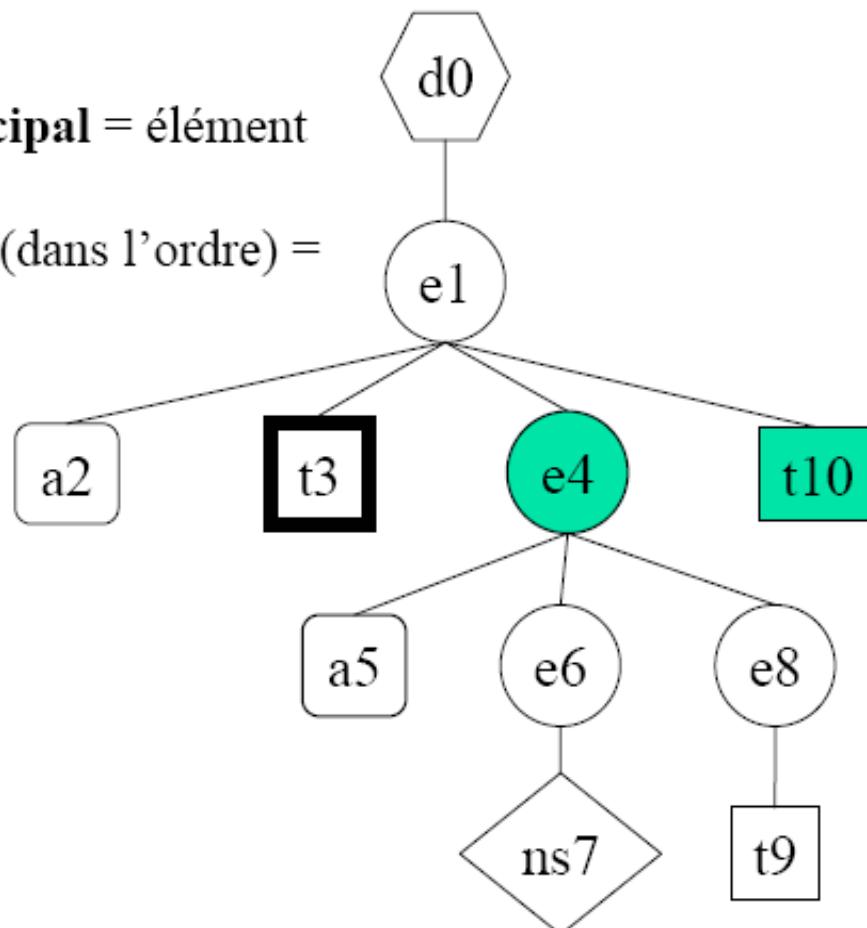
Axe : following-sibling

Sens = avant

Sorte de nœud principal = élément

Noeud contexte = t3

Nœuds sélectionnés (dans l'ordre) =
e4, t10



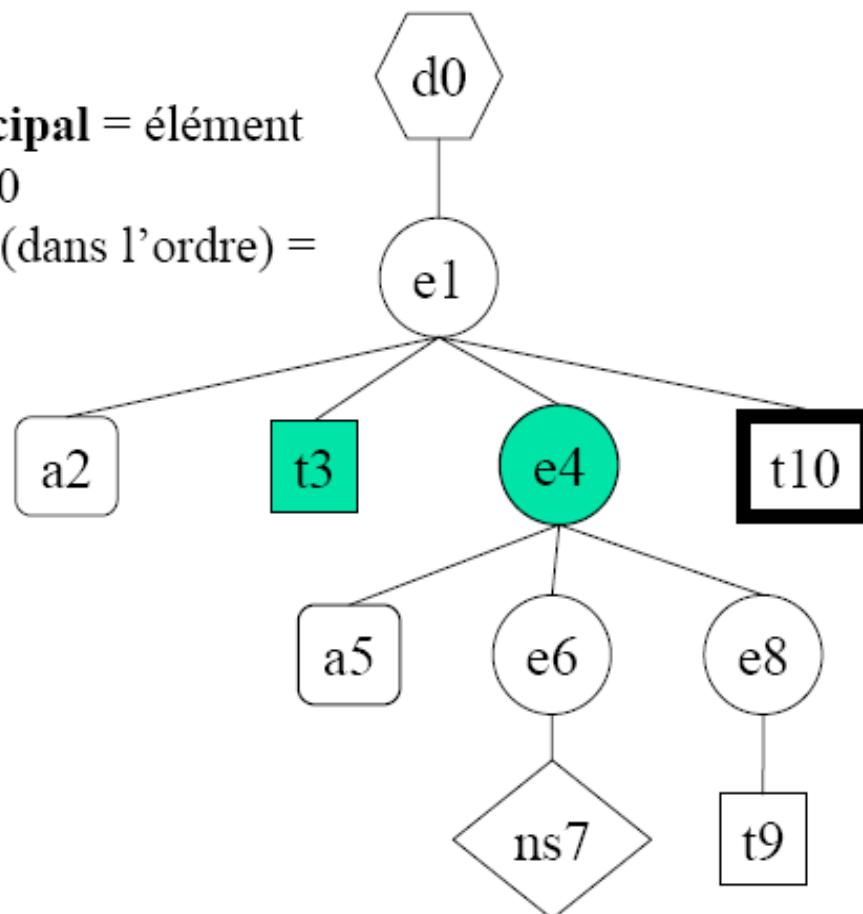
Axe : preceding-sibling

Sens = arrière

Sorte de nœud principal = élément

Noeud contexte = t10

Nœuds sélectionnés (dans l'ordre) =
e4, t3



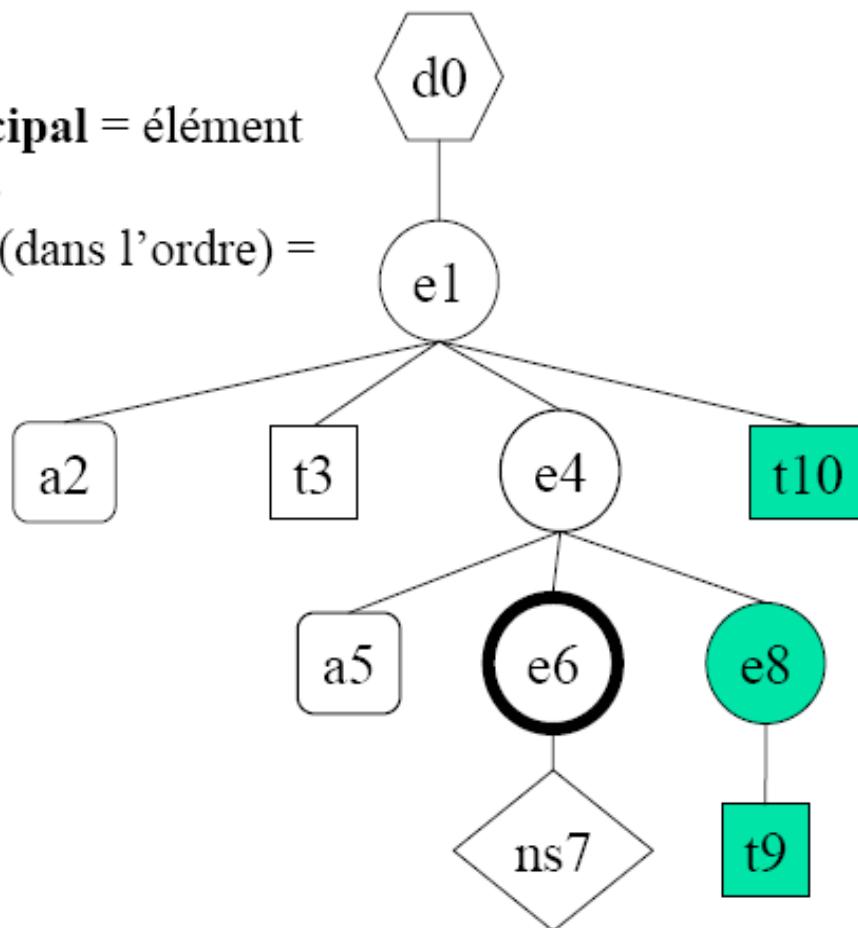
Axe : following

Sens = avant

Sorte de nœud principal = élément

Noeud contexte = e6

Nœuds sélectionnés (dans l'ordre) =
e8, e9, e10



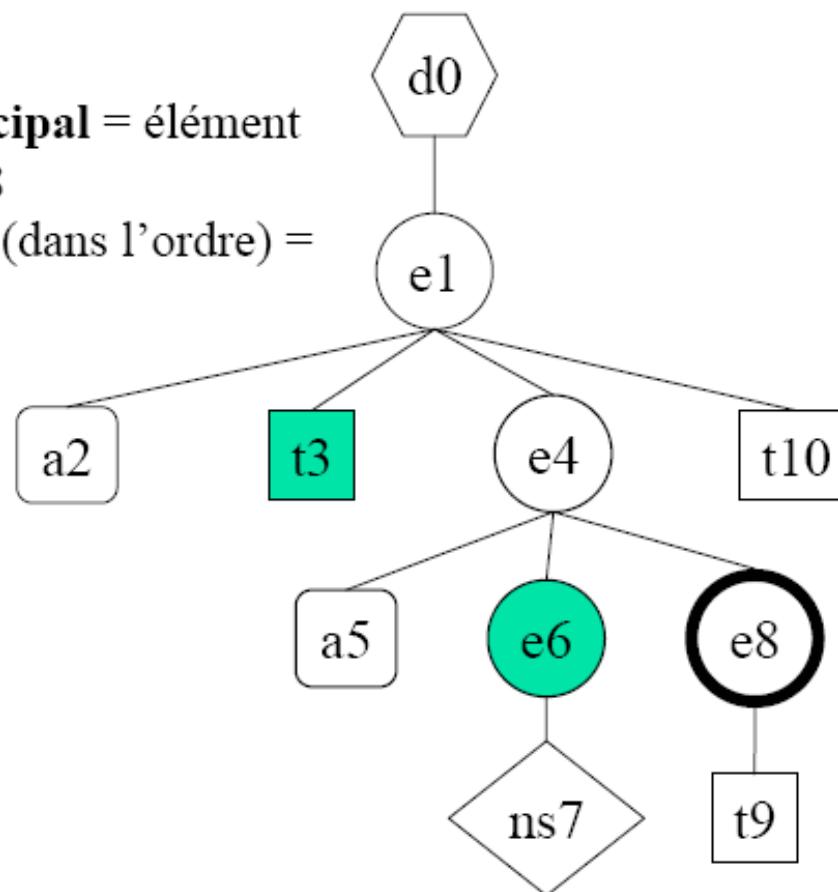
Axe : preceding

Sens = arrière

Sorte de nœud principal = élément

Noeud contexte = e8

Nœuds sélectionnés (dans l'ordre) =
e6, t3



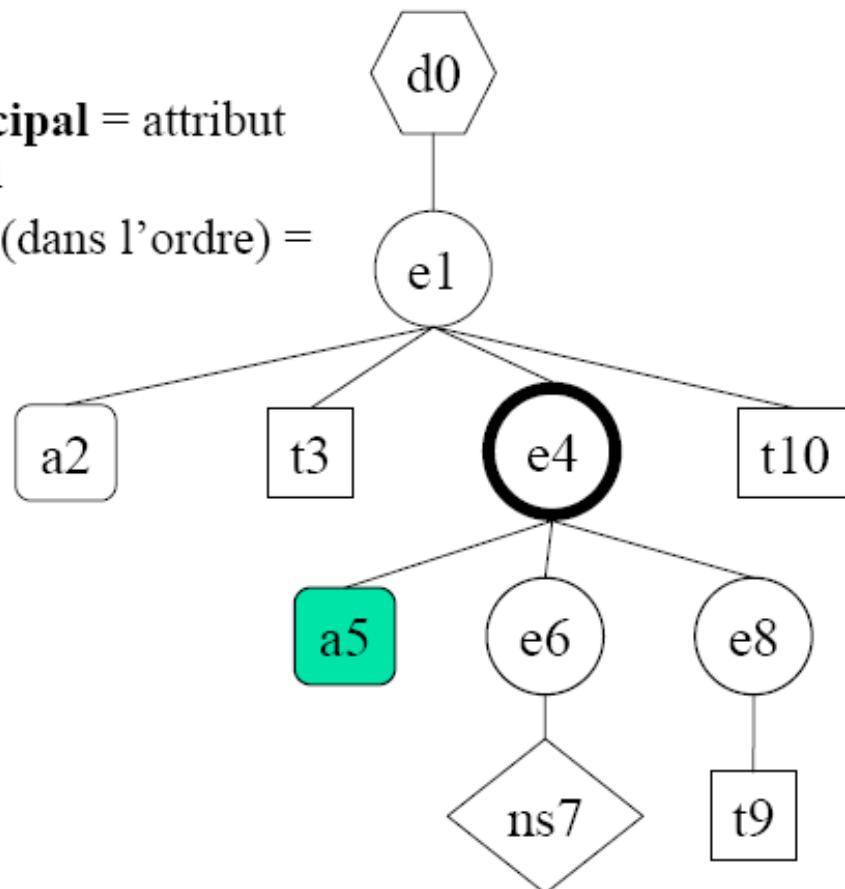
Axe : attribute

Sens = avant

Sorte de nœud principal = attribut

Noeud contexte = e4

Nœuds sélectionnés (dans l'ordre) =
a5



Axe : namespace

Sens = avant

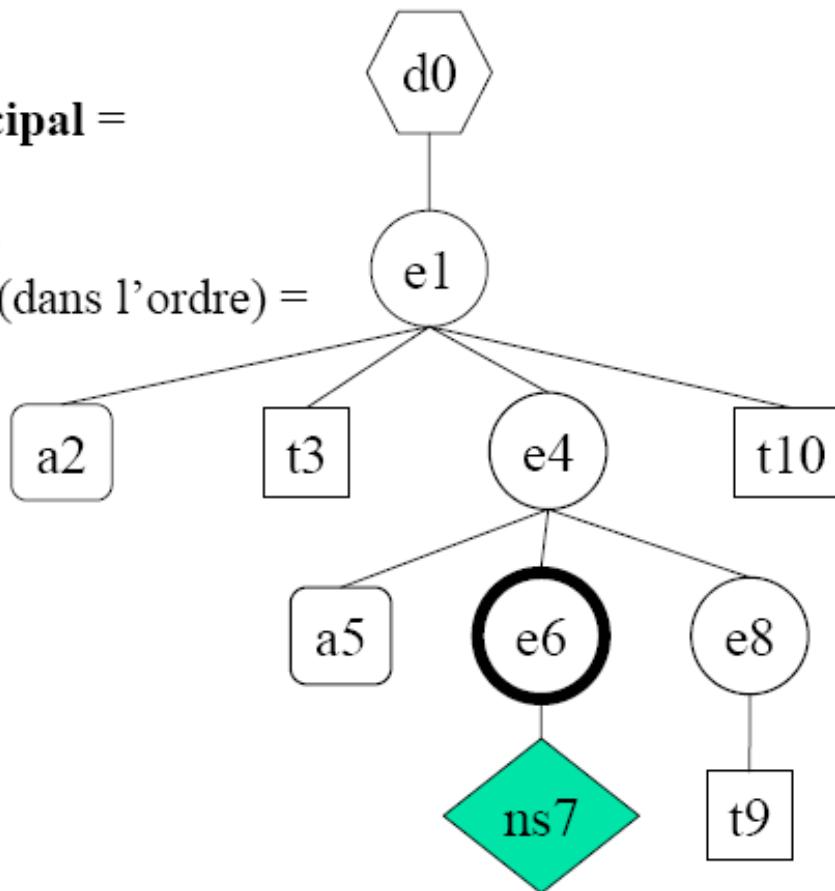
Sorte de nœud principal =

espace de nom

Noeud contexte = e6

Nœuds sélectionnés (dans l'ordre) =

ns7



Pas de localisation = Rappel

- Un **pas de localisation** est caractérisé par :
 - un **axe**,
 - un **test de noeud**,
 - une suite éventuellement vide de **prédictats**.
- **Syntaxe** :
 - *axe::test de noeud prédictat₁... prédictat_n*

Test des nœuds (1)

- Un **test de noeud** sélectionne parmi les noeuds de l'axe, ceux qui sont d'un certain type.
- Un test de noeud a l'une des formes suivantes :
 - ***n*** où *n* est un nom : sélectionne les noeuds de l'axe de même sorte que la sorte principale de l'axe et dont le nom étendu est égal au nom étendu de *n* ;
 - ***** : sélectionne les noeuds de l'axe de même sorte que la sorte principale de l'axe;

Test des nœuds (2)

- Un test de noeud a l'une des formes suivantes :
 - **node()** : sélectionne tout noeud de l'axe ;
 - **text()** : sélectionne tout noeud de l'axe de sorte texte ;
 - **comment()** : sélectionne tout noeud de l'axe de sorte commentaire ;
 - **processing-instruction(*n*)** : sélectionne tout noeud de l'axe représentant une instruction de traitement de nom *n*.

Notations

- Pour faciliter la lecture des chemins de localisation, les abréviations suivantes sont autorisées :
 - **child** est l'axe par défaut : **t** est l'écriture abrégée de **child::t**
 - **@** est l'écriture abrégée de **attribute::**
 - **.** est l'écriture abrégée du pas de localisation **self::node()**
 - **..** est l'écriture abrégée du pas de localisation **parent::node()**
 - **//** est l'écriture abrégée de **/descendant-or-self::node()//**

Exemples (1)

- Nom des itinéraires du vallon des Muandes cotés ** et ne comportant pas de notes de prudence.
 - /guide/vallon[nom = "Vallon des Muandes"]
 - /itinéraire[cotation = "***" and not (para/note[@type = "prudence"])]
 - /nom
 - /text()
- Nom des vallons possédant au moins deux itinéraires cotés **** ?
 - /guide
 - /vallon[count(itinéraire[cotation = "****"]) > 1]
 - /nom/text()

Exemples (2)

- Quel est le second itinéraire ** du Vallon des Muandes ?
 - /guide/vallon[nom = "Muandes"]
 - /itinéraire[cotation = "***"]**[2]**
 - /nom/text()
- Cet exemple montre l'intérêt des prédictats successifs lors de l'extraction d'un noeud par rapport à sa position de proximité. Ici on cherche le **deuxième** des itinéraires **.
- Quels sont les noms des itinéraires du vallon des Muandes décrits après celui de la Pointe de Névache ?
 - /vallon[nom = "Muandes"]
 - /itinéraire[nom = "Pointe de Névache"]
 - /following-sibling::itinéraire

Exemples (3)

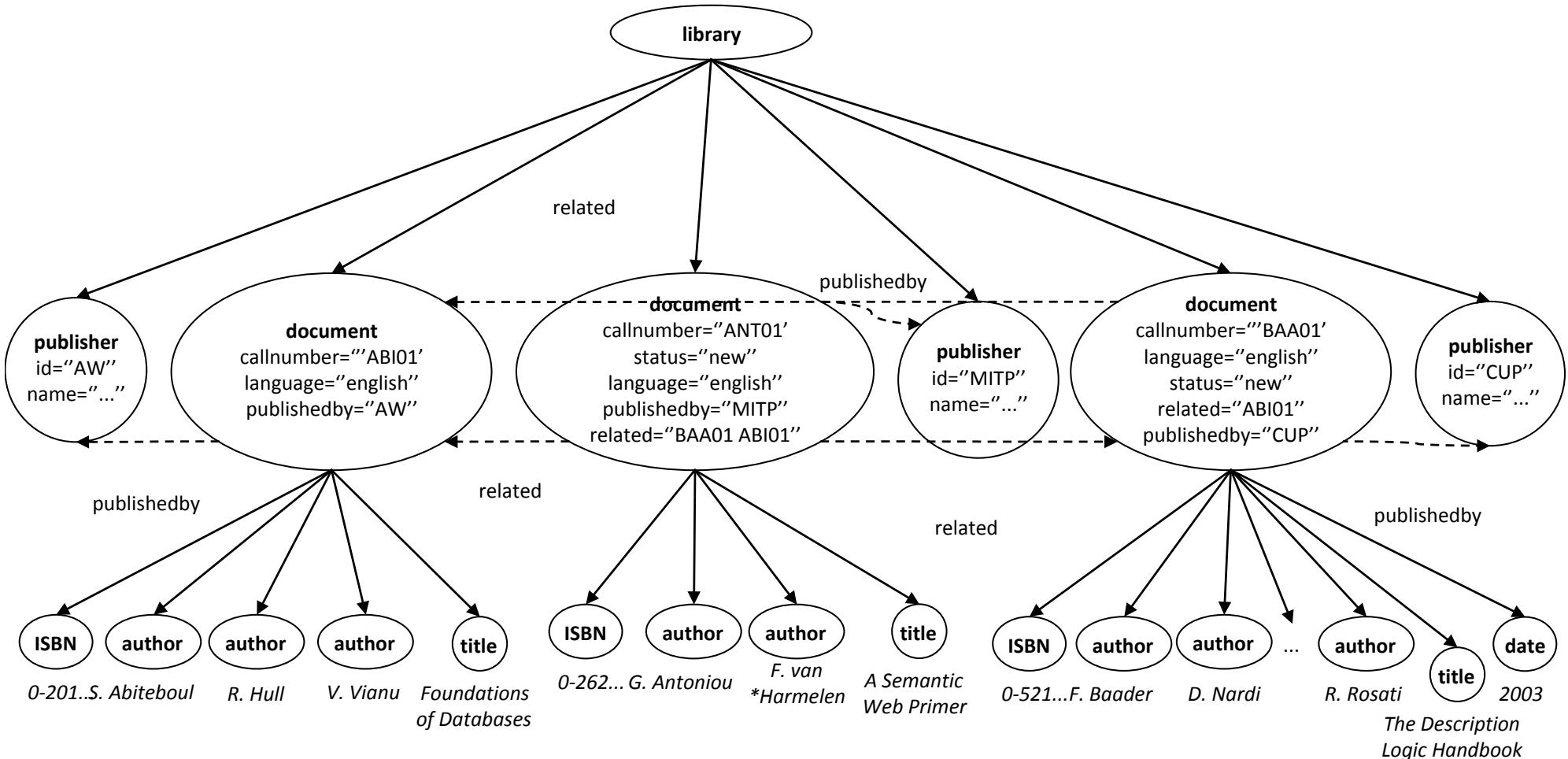
- Quel est le nom du dernier vallon du guide ?
/guide/vallon[last()]/nom
- Quels sont les textes de toutes les notes de type « prudence » ?
//note[@type = "prudence"]/text()
- Quels sont les noms des itinéraires cités dans la description de l'itinéraire de la Pointe de Névache ?
//itinéraire[contains(text(), "Pointe de Névache ")]
- Quel est le nom du vallon dans lequel se trouve l'itinéraire **I15.1** ?
//itinéraire[@id = "I15.1"]/parent::vallon/nom
- ou bien :
//itinéraire[@id = "I15.1"]/..//nom

Fonctions XPath

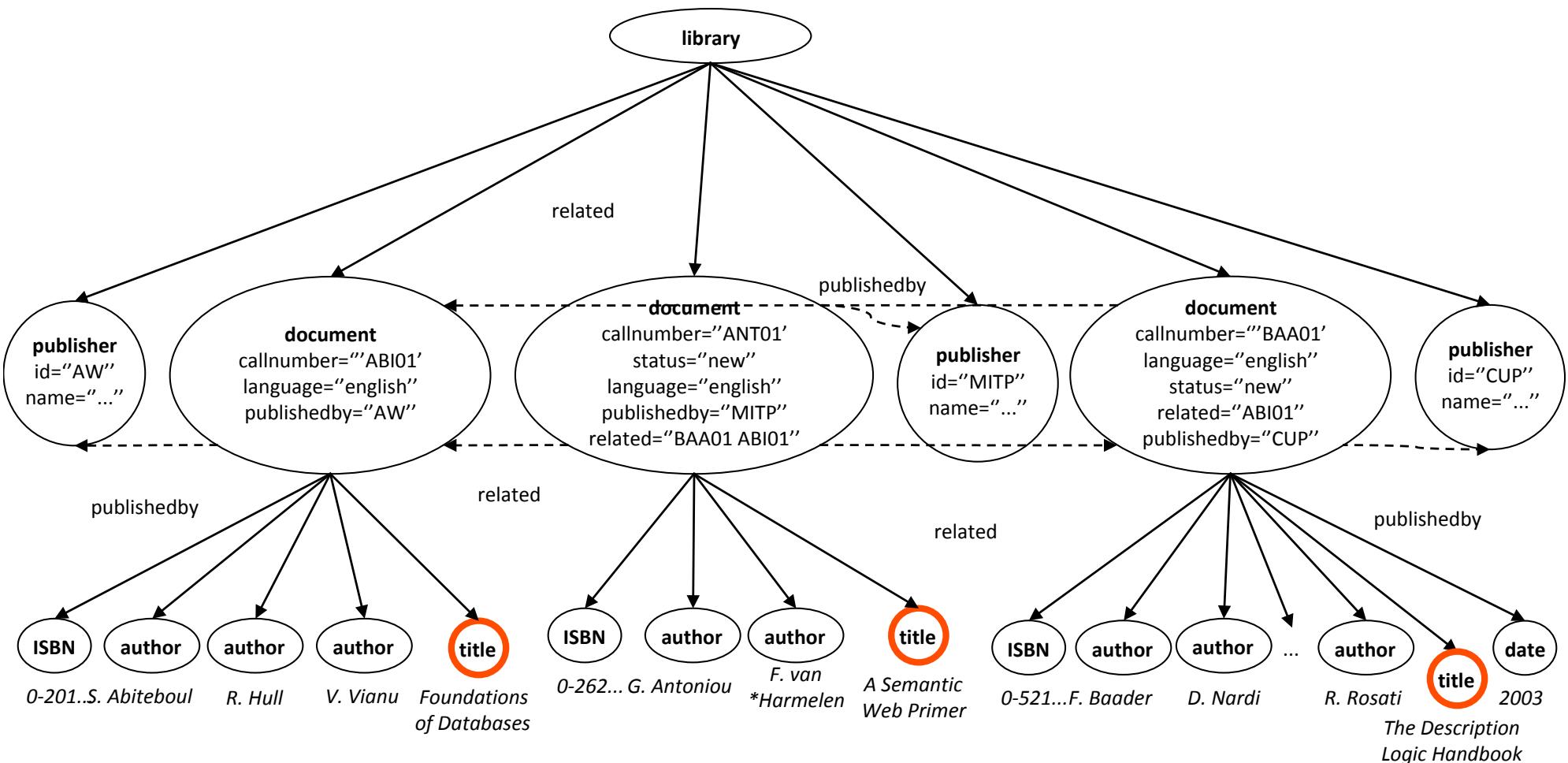
- **last** → nombre égal à la dimension contextuelle issue du contexte d'évaluation de l'expression
- **position** → nombre égal à la position contextuelle issue du contexte d'évaluation de l'expression
- **count** → nombre de nœuds de l'ensemble de nœuds passé en argument

EXEMPLES EN IMAGES

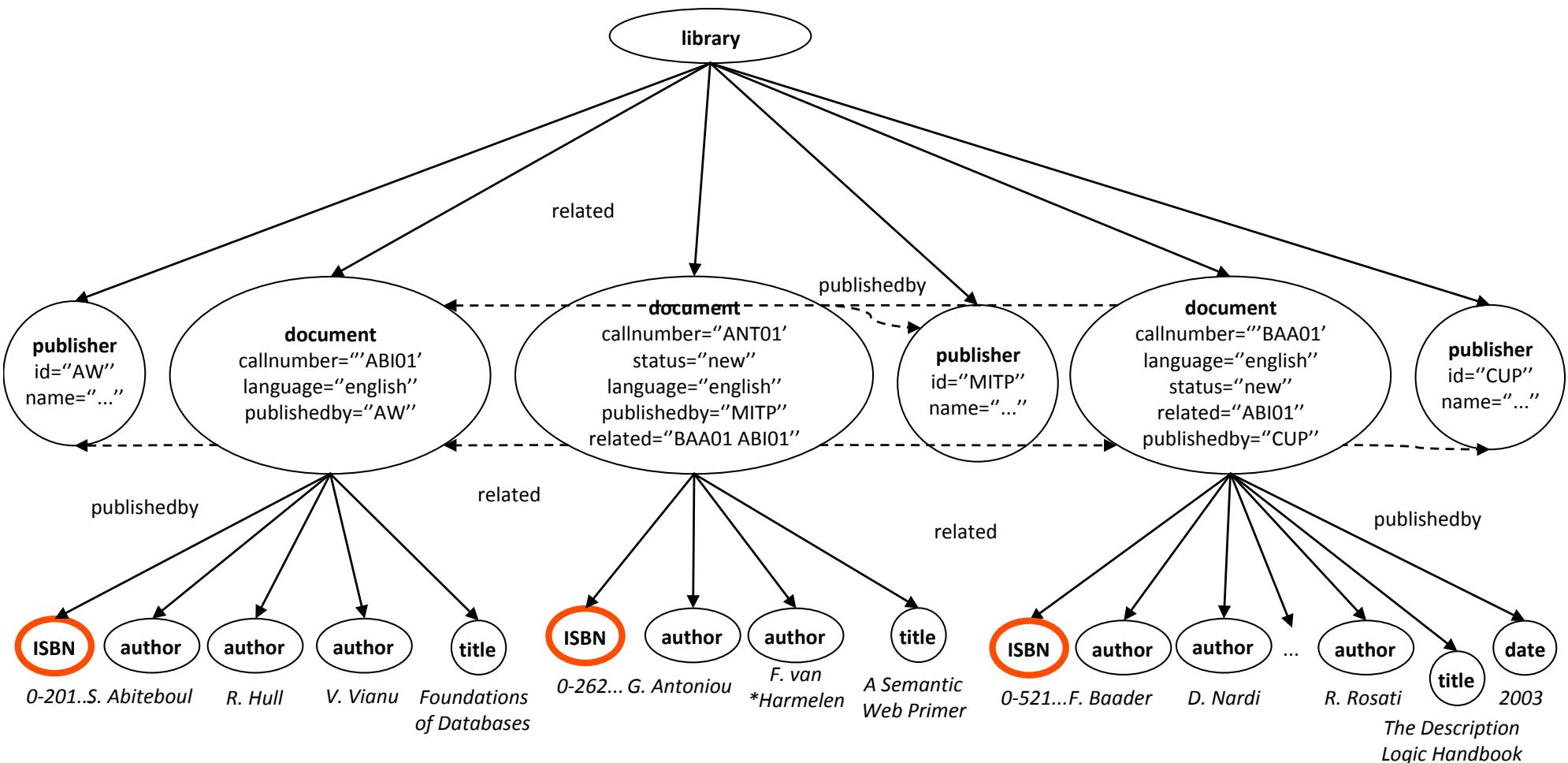
Arbre XML



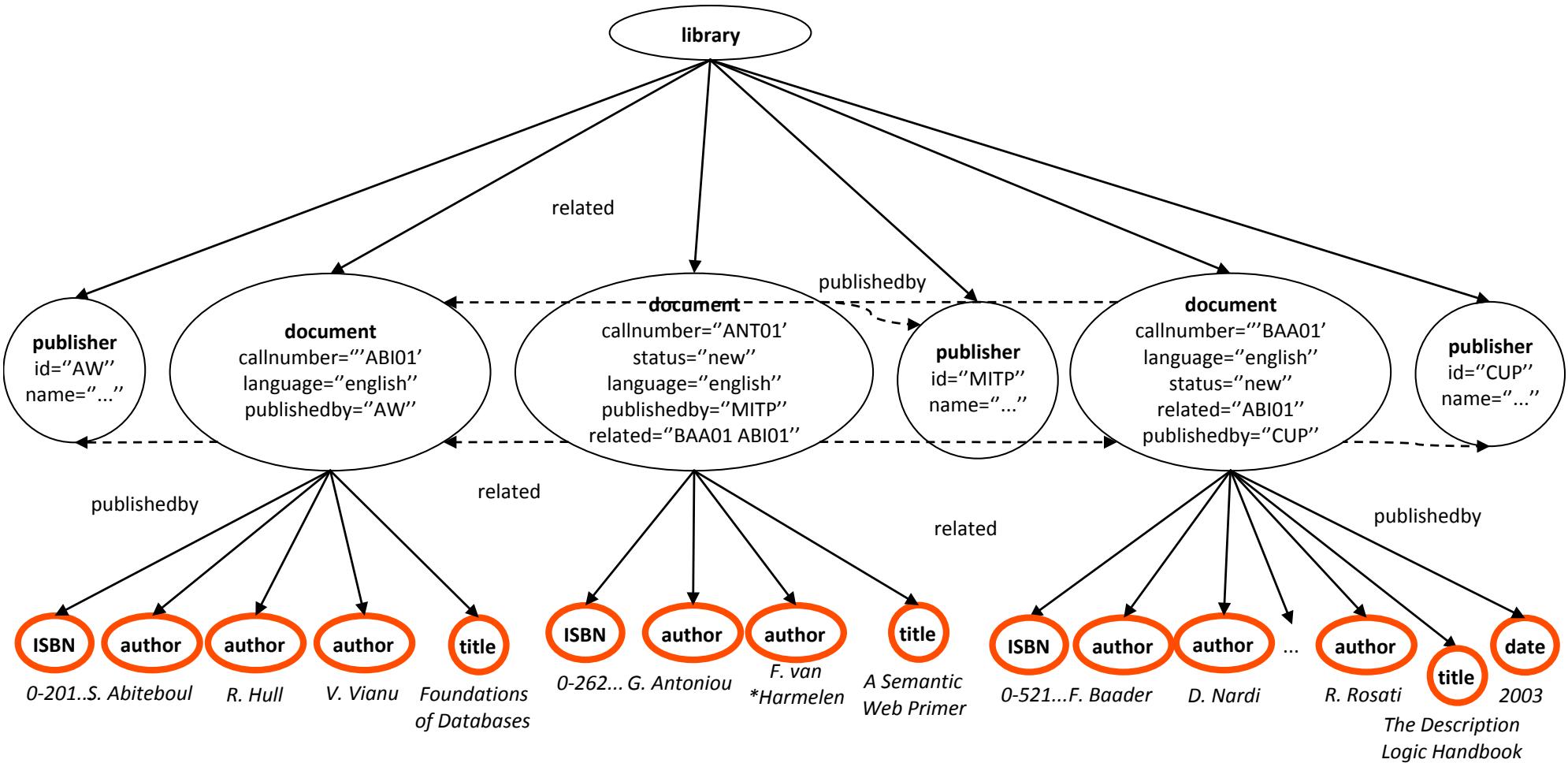
/library/document/title



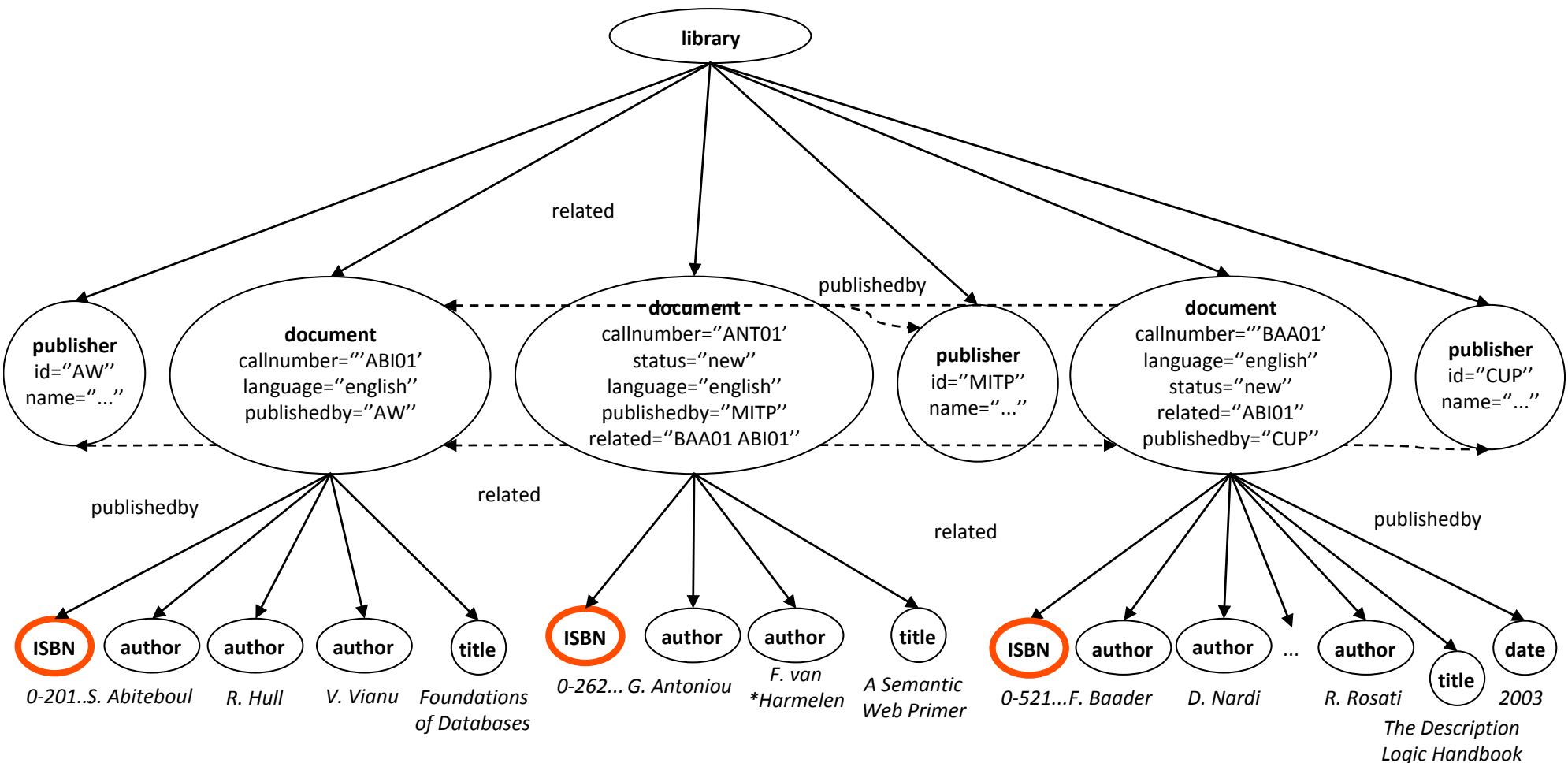
//ISBN



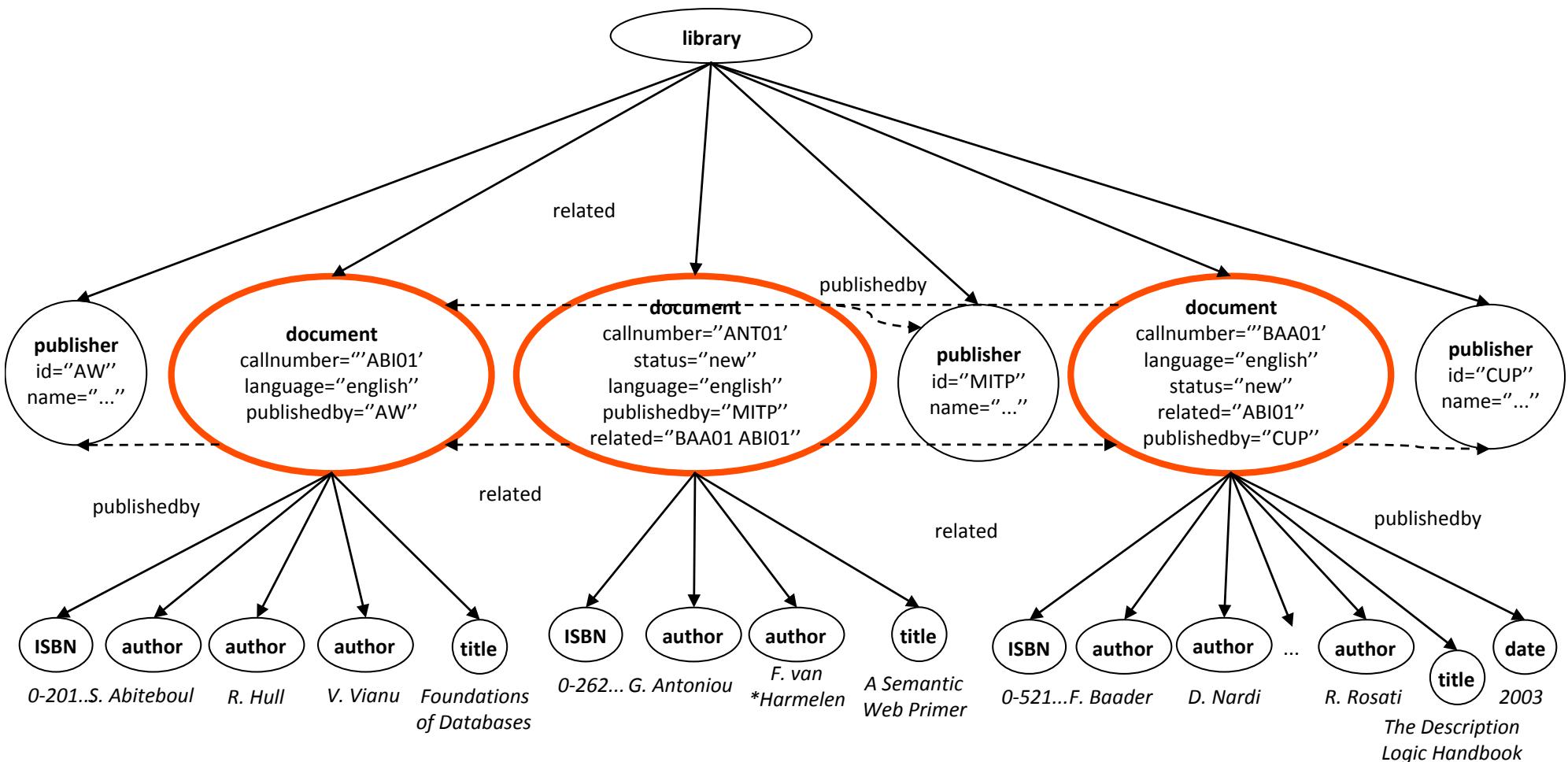
/library/document/*



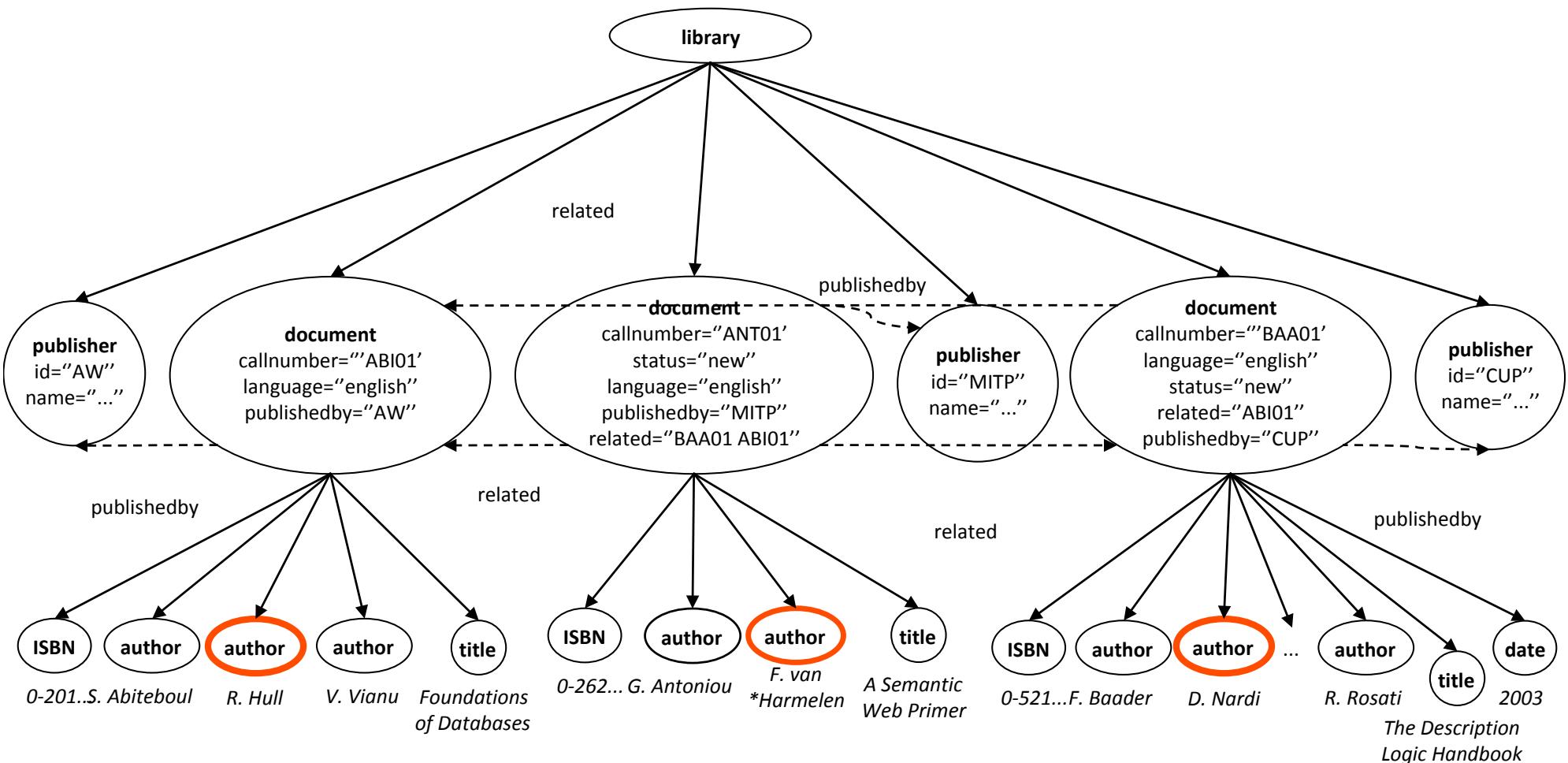
/*/*/ISBN



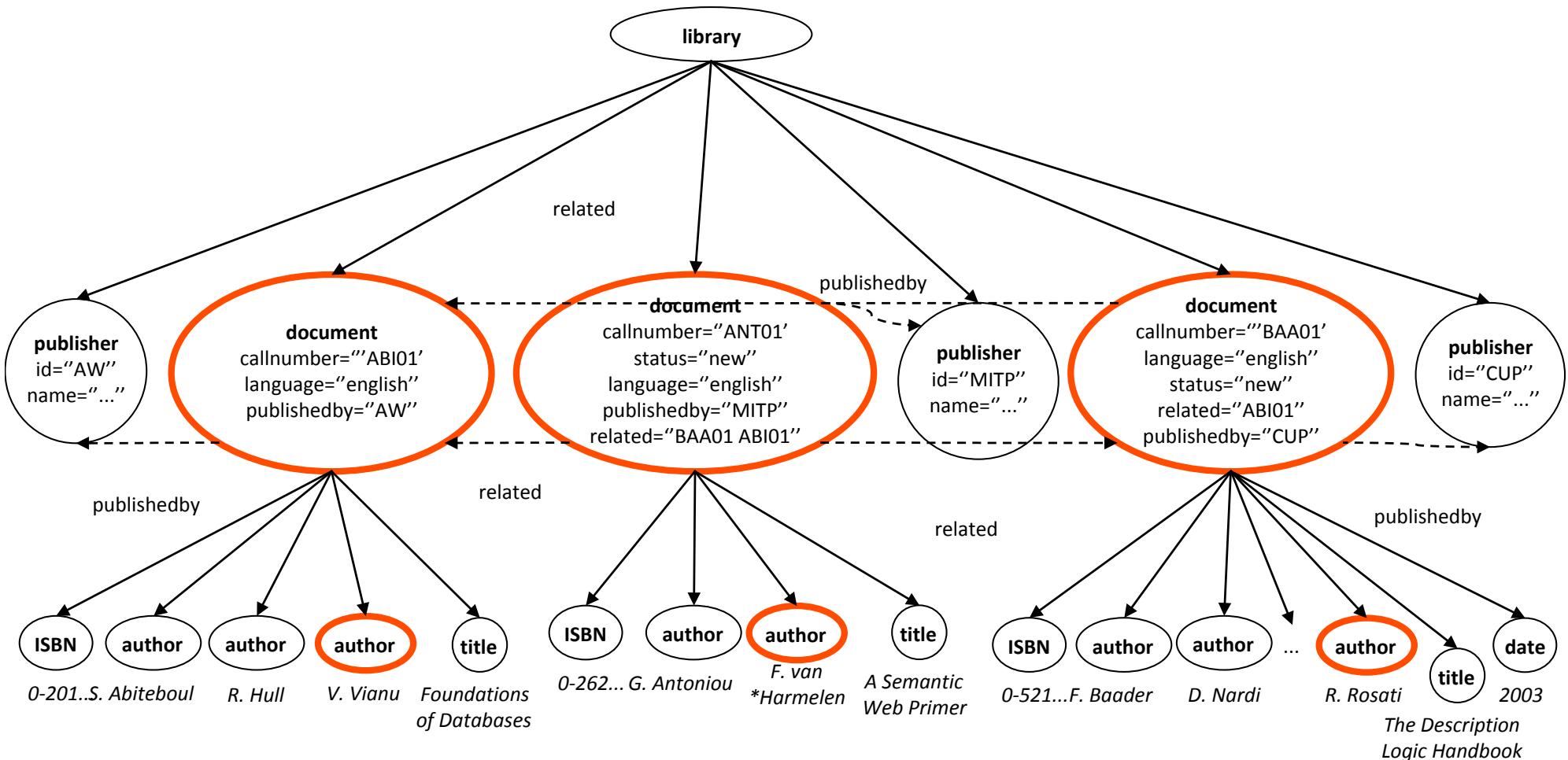
/*/*/*/..



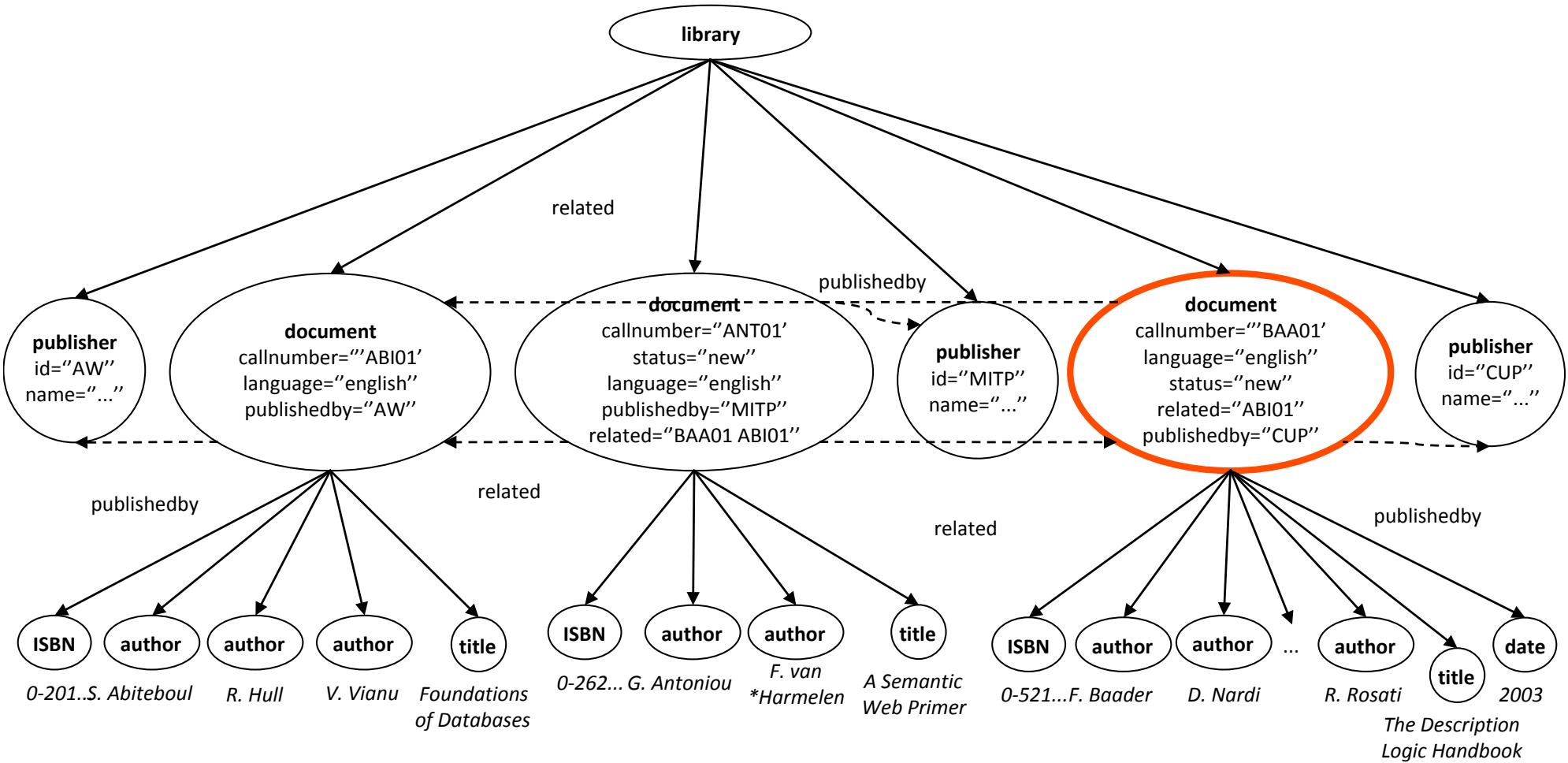
/library/document/author[2]



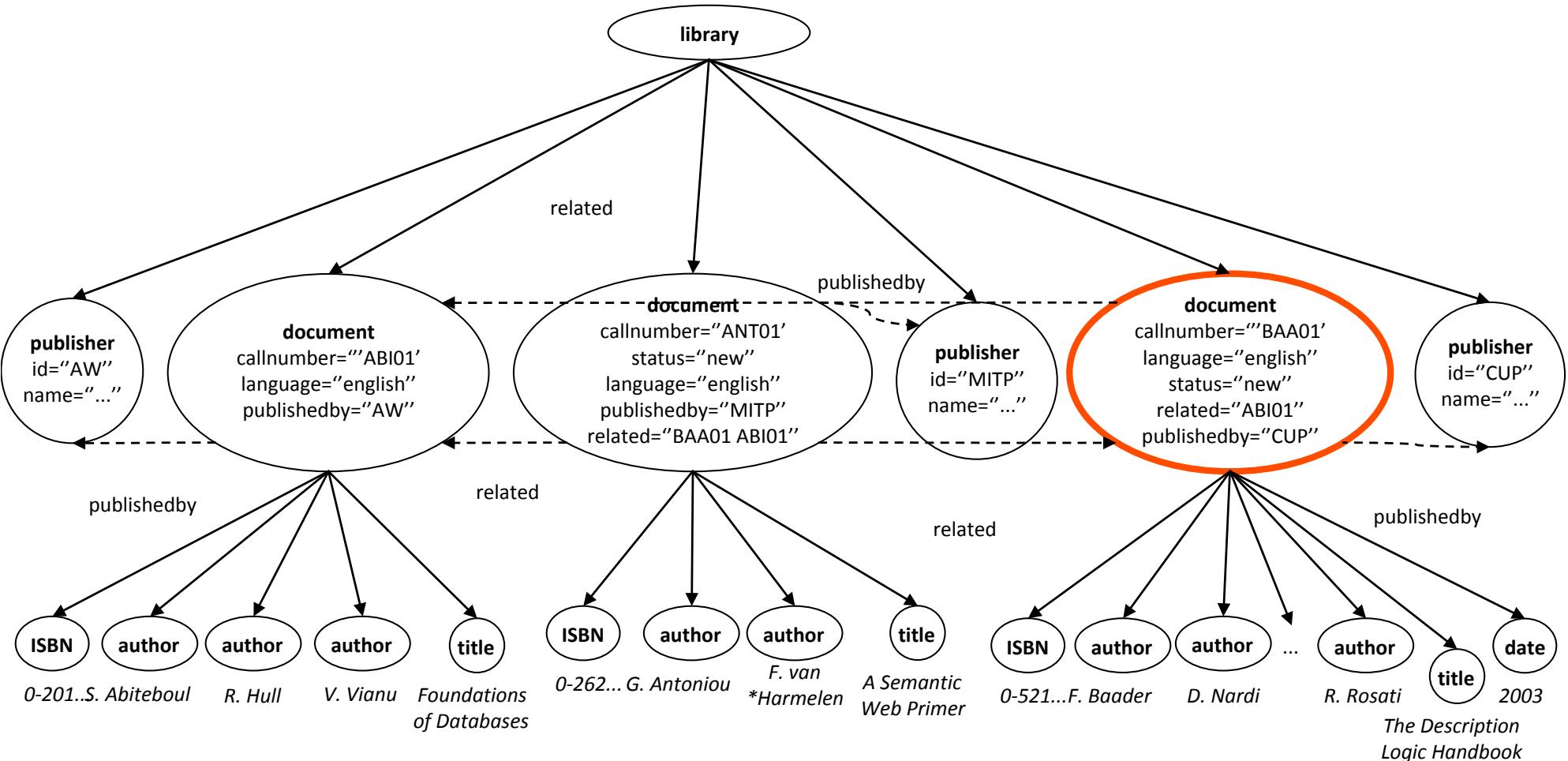
///*/*/.. | /library/document/ author[last()]



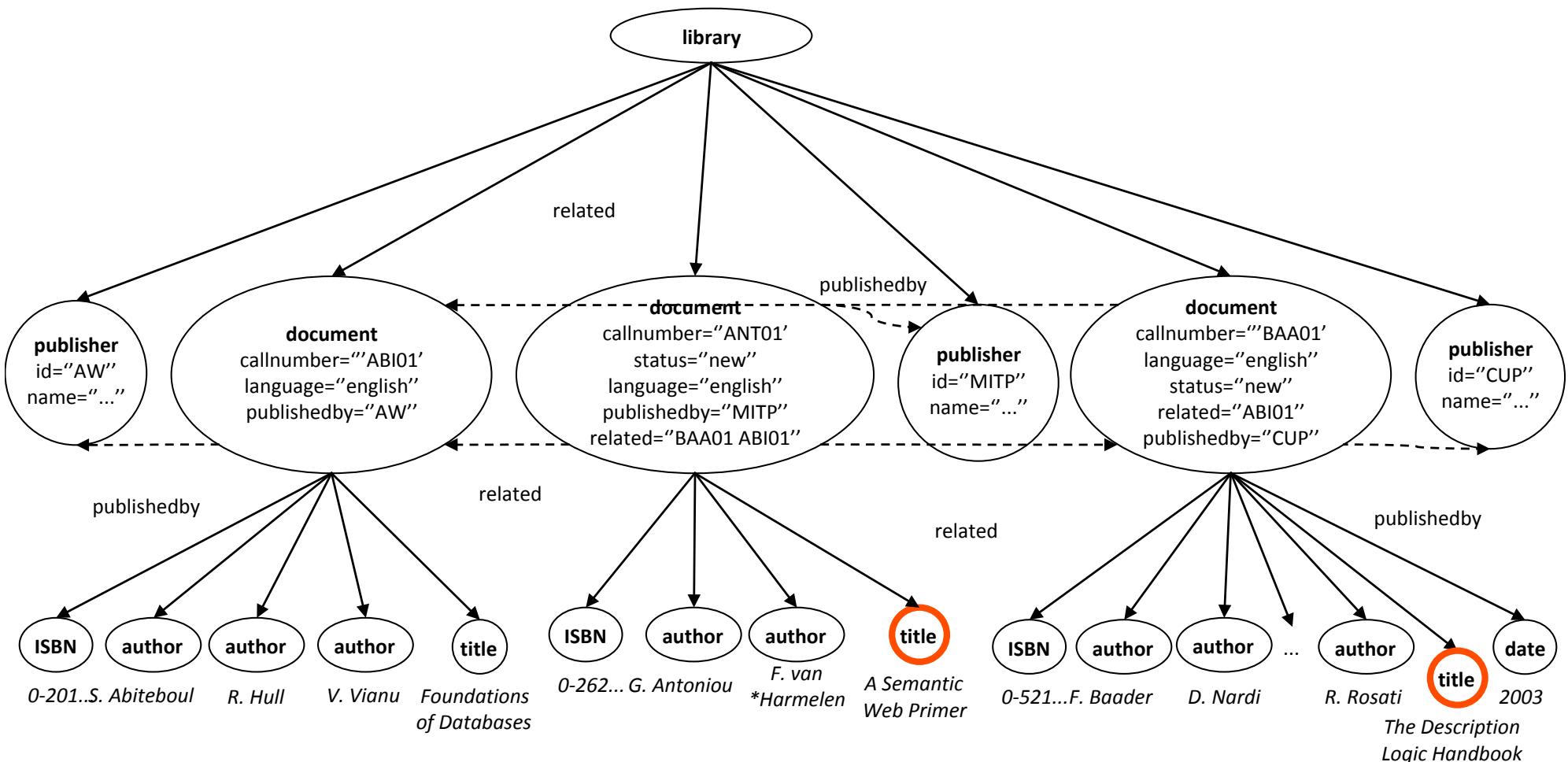
/library/document[date]



/library/document[ISBN!='0-262-01210-3' and date]



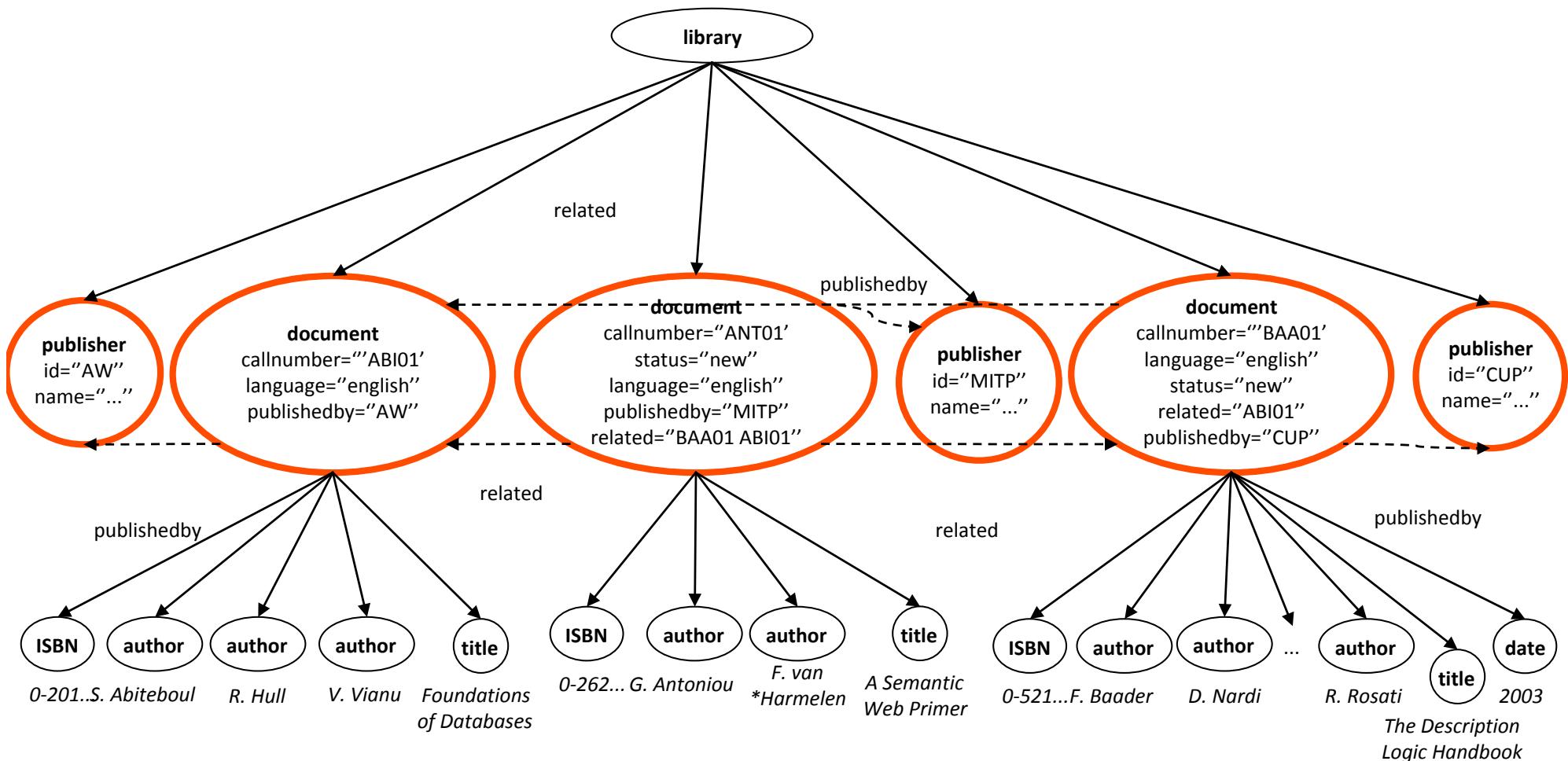
/library/document[ISBN='0-262-01210-3' or date]/title



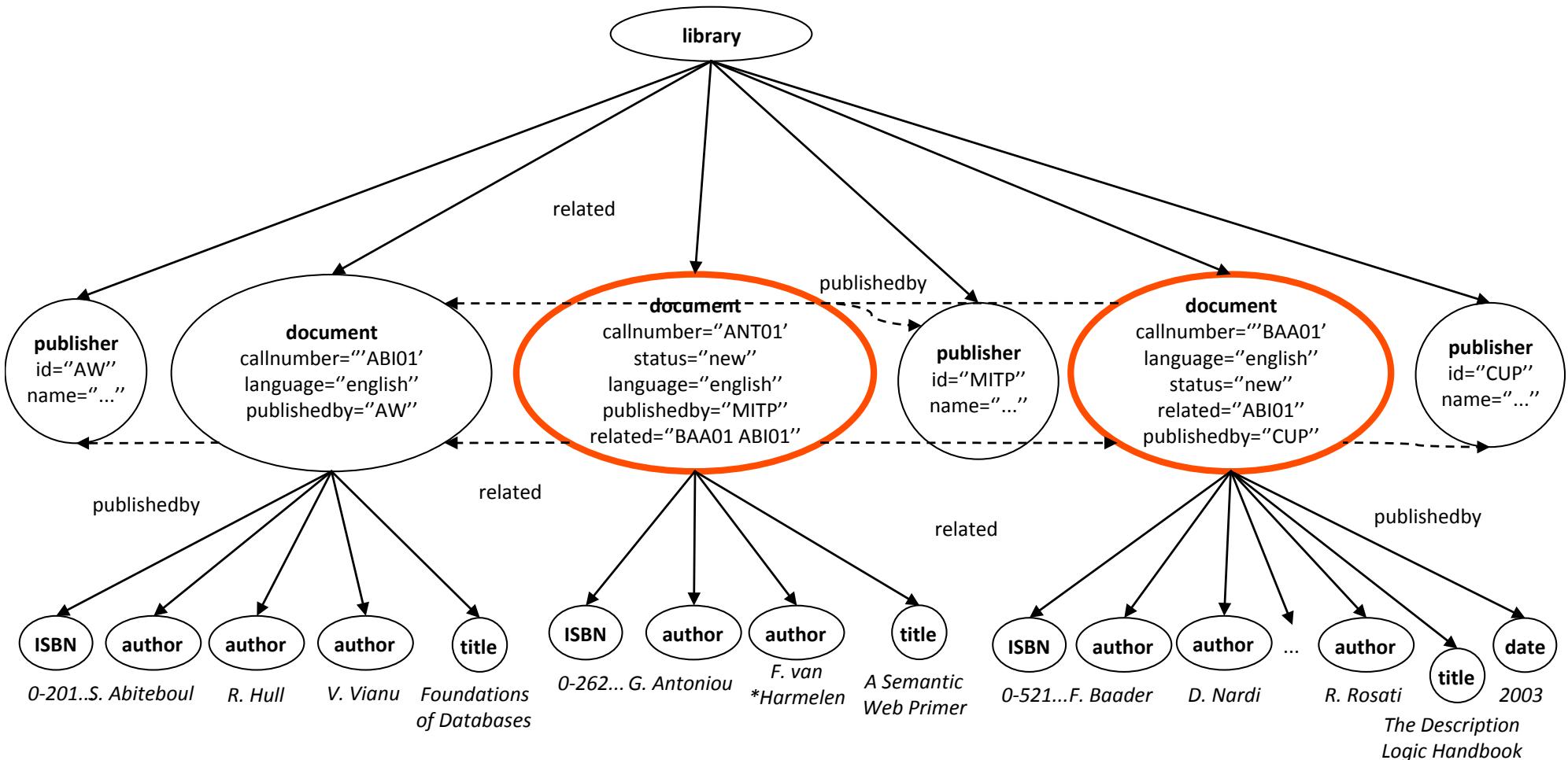
Interrogation XML : XPath

- Éléments qui possèdent des attributs
 - `//*[@*]`
- Éléments ayant un attribut donné
 - `//document[@related]`
- Éléments ayant des attributs donnés
 - `//document[@related and @language]`
- Éléments ayant des attributs donnés avec une valeur donnée
 - `//document[@publishedby='AW' or @status='new']`
- Nièmes Éléments ayant une valeur donnée
 - `/library/document[@language='english'][2]`
- Nièmes Éléments s'ils ont la valeur donnée
 - `/library/document[2][@language='english']`

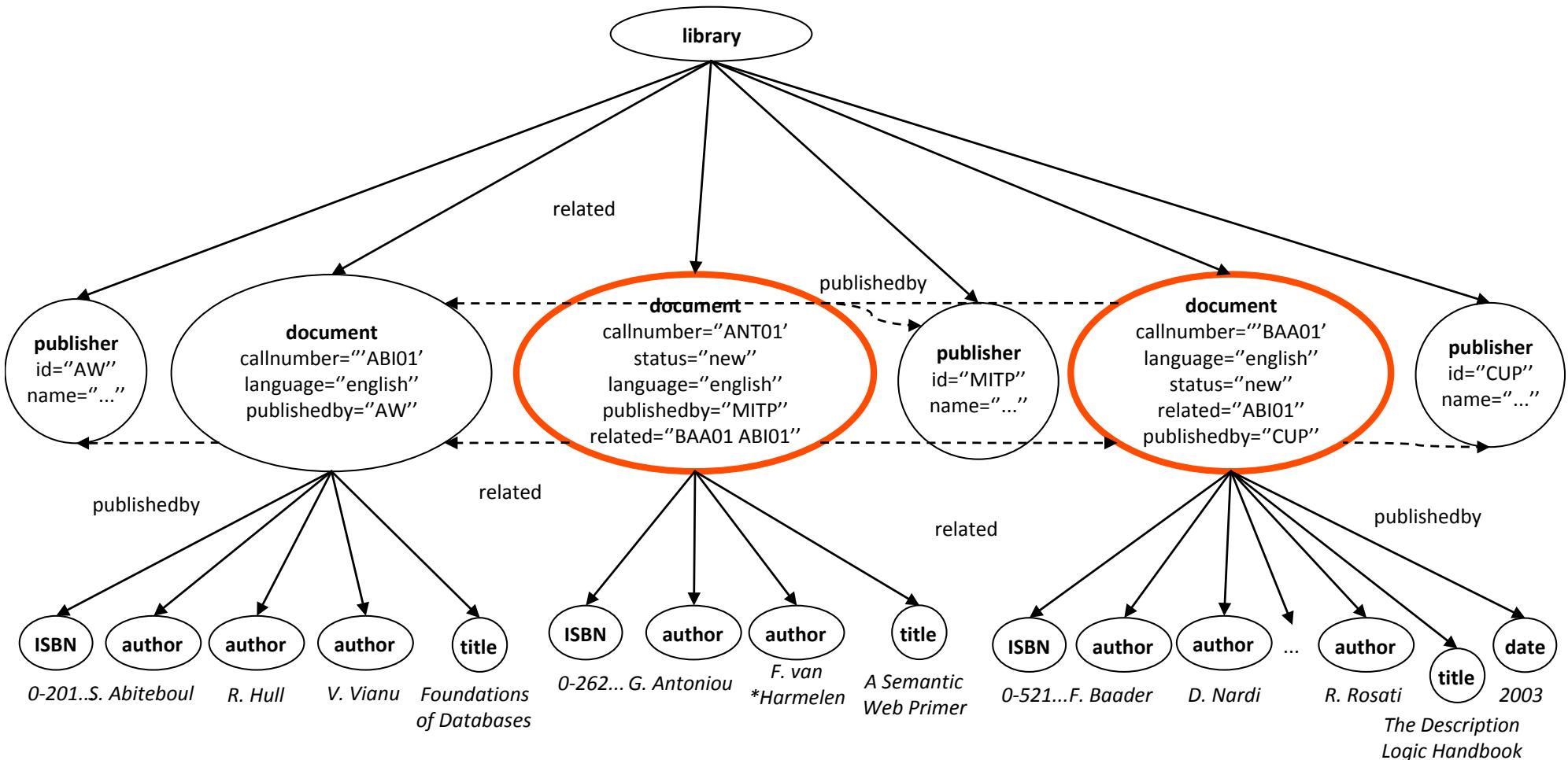
//*[@*]



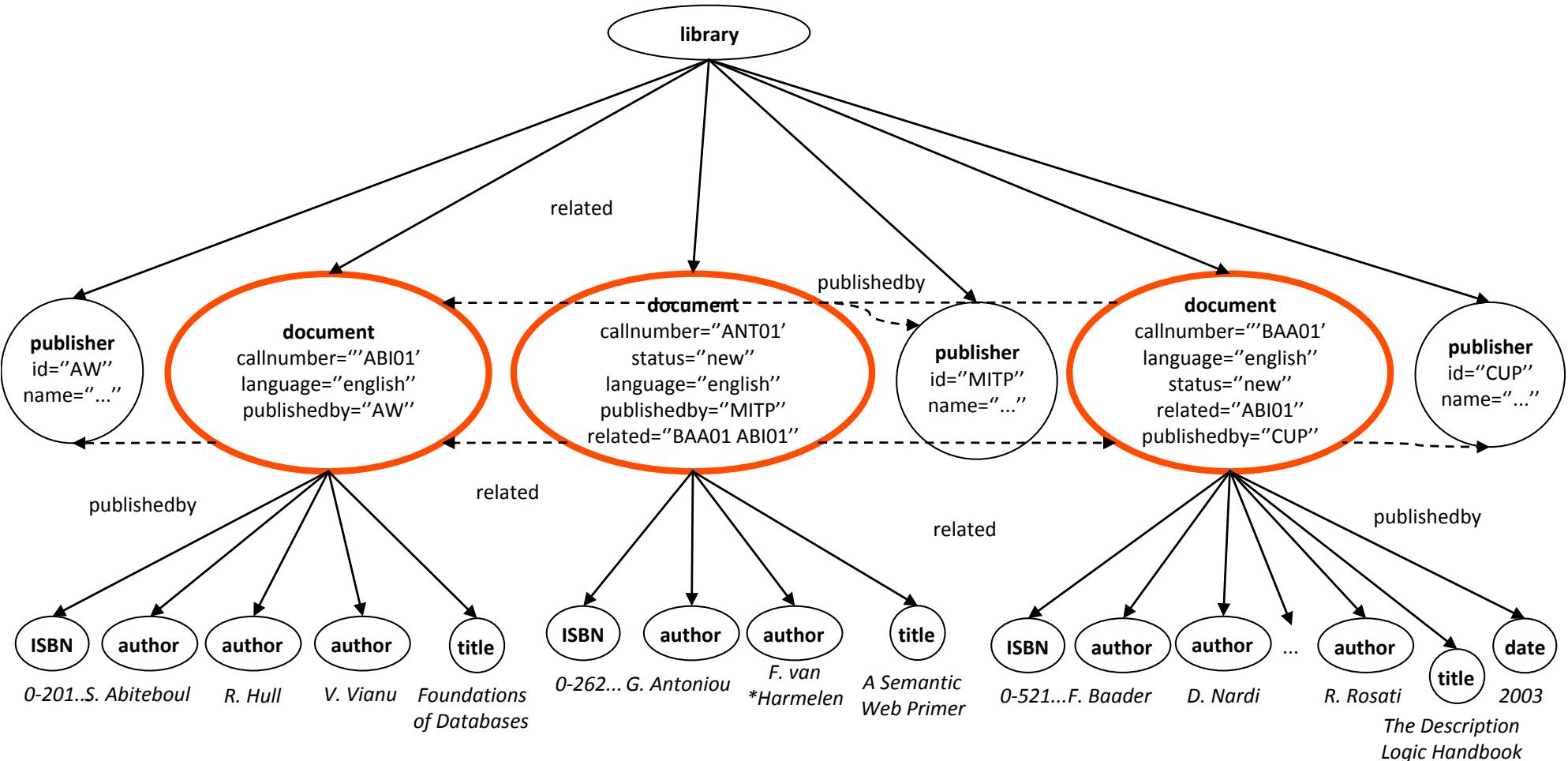
//document[@related]



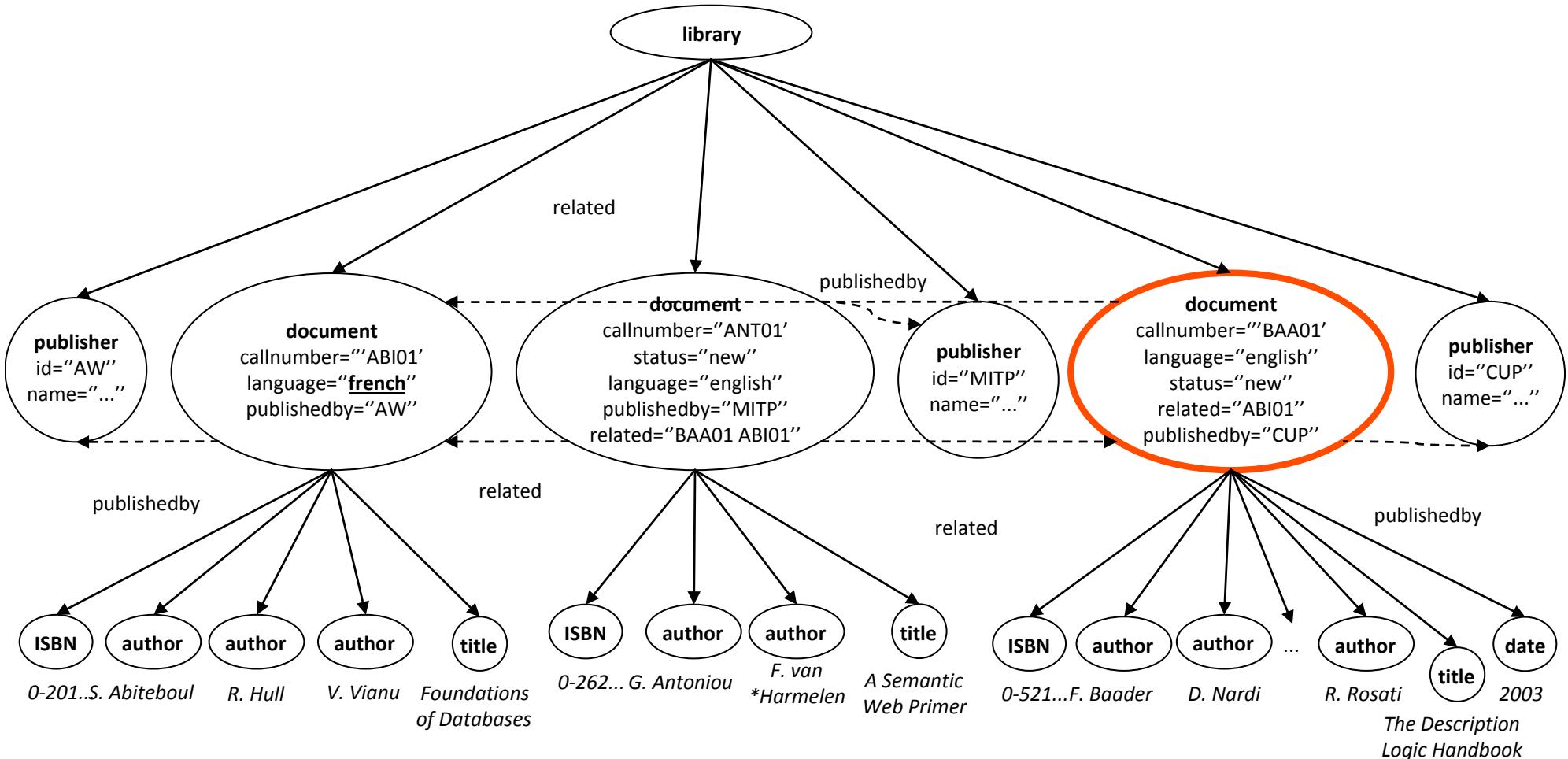
//document[@related and @language]



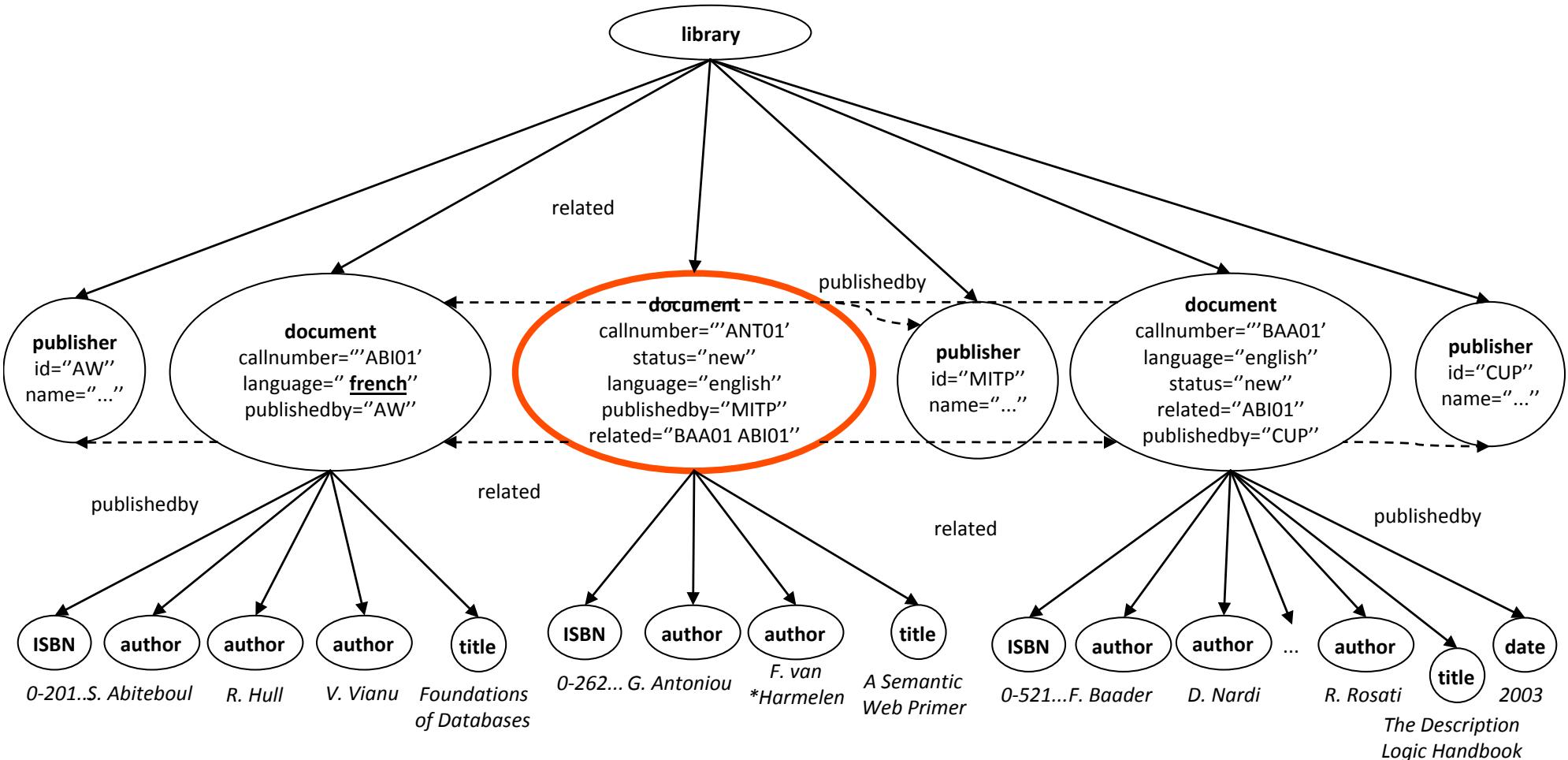
//document[@publishedby='AW' or @status='new']



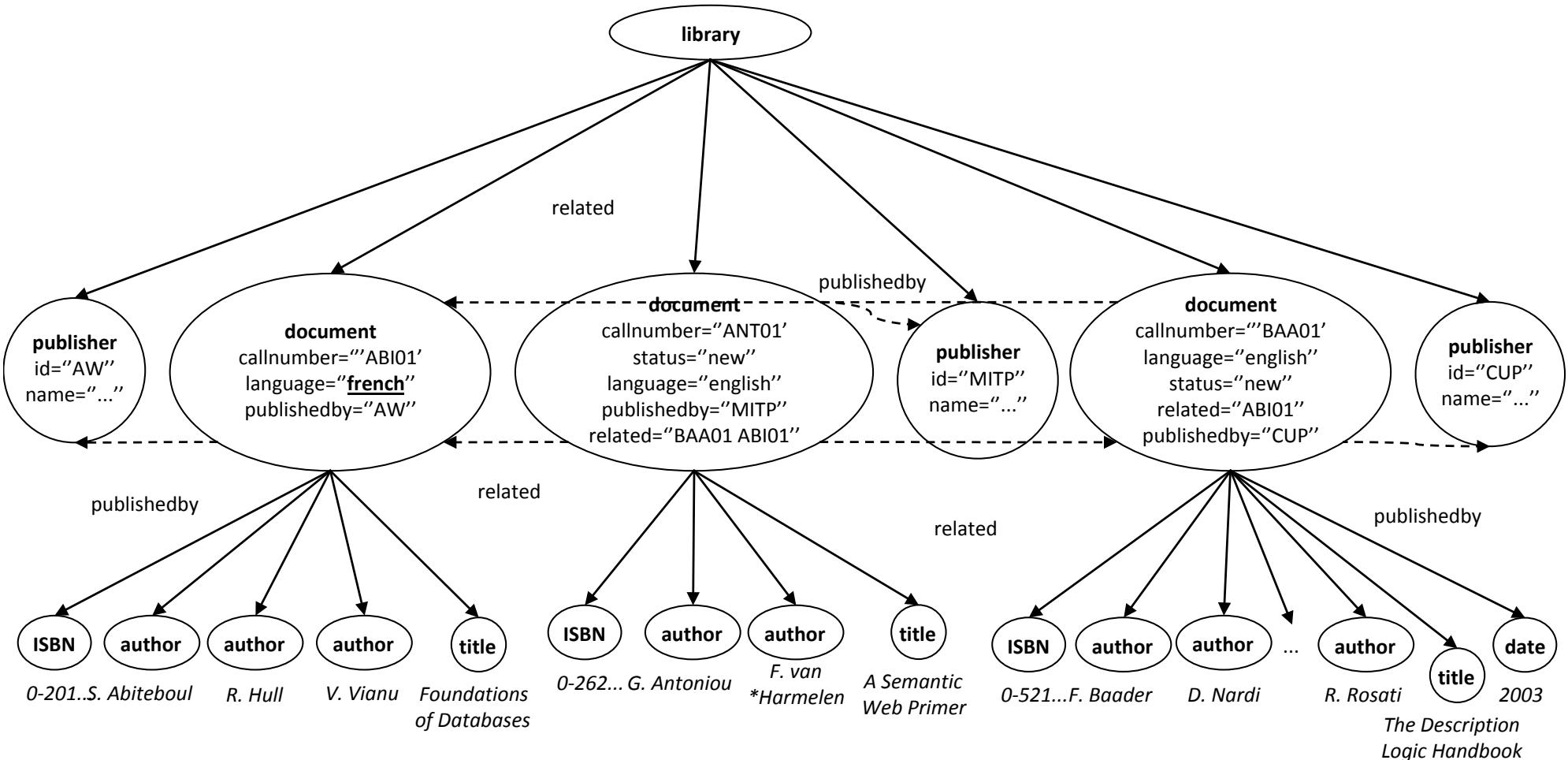
/library/ document[@language='english'][2]



/library/document[2] [@language='english']



/library/document[1] [@language='english']



XPath - Synthèse

Pattern	Exemple	Signification
Nom	section	Sélectionne les éléments de nom donné
Nom[0]	section[0]	Sélectionne le premier élément ayant le nom donné
Nom[end()]	section[end()]	Sélectionne le dernier élément ayant un nom donné
	Droite Gauche	Indique une alternative (un nœud OU bien l'autre (ou les deux))
/	/	Sélectionne le nœud racine d'une arborescence
/arbre/Nom	/livre/chapitre	Sélectionne les nœuds descendants par la balise de nom donné de l'arbre
*	*	Motif "joker" désignant n'importe quel élément
//	//personne	Indique tous les descendants d'un nœud
.	.	Caractérise le nœud courant
..	..	Désigne le nœud parent. Permet de remonter d'un niveau dans l'arborescence
@	@nom	Indique un attribut caractéristique (@ <i>nom</i> désigne la valeur de l'attribut). La notation @* désigne tous les attributs d'un élément
text()	text()	Désigne le contenu d'un élément (le texte contenu entre ses balises)
ID()	ID('a2546')	Sélectionne l'élément dont l'identifiant (la valeur de l'attribut ID) est celui spécifié en paramètre
Comment()	Comment()	Désigne tous les nœuds commentaires
Node()	Node()	Désigne tous les noeuds

- `string-length(...)` : longueur d'une chaîne ;
- `starts-with(chaîne1, chaîne2)` : tester si chaîne1 commence par chaîne2 ;
- `substring(chaîne1, position1, longueur)` : extraction d'une sous-chaîne ;
- `normalize-space(chaîne)` : normalisation des occurrences de blancs à 1 blanc ; suppression des blancs d'en-tête et de fin ;

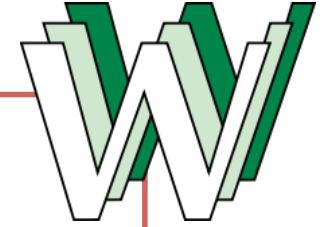
- `translate(chaîne, caractères source, caractères destination)` : convertit dans la chaîne tous les caractères source par leur correspondance (en fonction de la position) dans le dernier argument ;
- `number(chaîne)` : conversion en nombre ;
- `string(expression)` : conversion en chaîne ;

- concat(chaîne1, chaîne2) : concaténation ;
- contains(chaîne1, chaîne2) : tester si chaîne1 contient chaîne2 ;
- floor(nombre décimal) : arrondi inférieur (10.9 devient 10, par exemple) ;
- ceil(nombre décimal) : arrondi supérieur (10.1 devient 11, par exemple) ;
- round(nombre décimal) : arrondi au plus juste (10.4 devient 10 et 10.6 devient 11, par exemple) ;
- count(NodeSet?) : nombre de nœuds ;
- position() : position courante commençant par 1 ;
- last(NodeSet?) : dernière position ;

- `name(NodeSet?)` : nom du nœud (tag s'il s'agit d'un élément) avec préfixe éventuel ;
- `local-name(NodeSet?)` : nom du nœud sans préfixe ;
 - `namespace-uri(NodeSet?)` : espace de noms ;
 - `generate-id(NodeSet?)` : génération d'un identifiant unique.
- Vous trouverez l'ensemble des fonctions sur le site du W3C à l'adresse :
- *<http://www.w3.org/TR/xpath#corelib>*

END

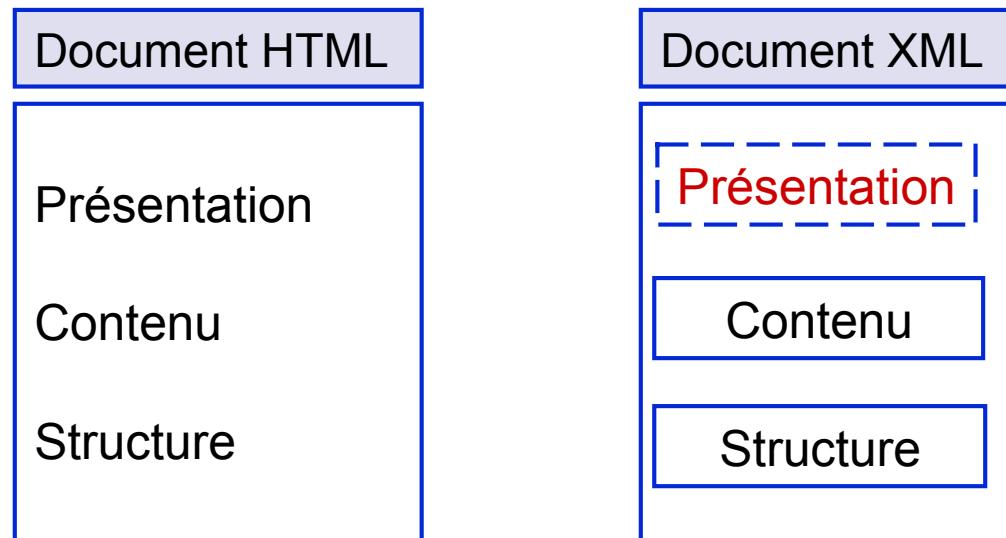




XML et ses applications : plan

1. Introduction à XML
2. Schémas de documents XML
 - a. DTD
 - b. XML Schema
3. Le modèle DOM
4. Interrogation : XPath
5. Transformation et présentation : XSLT
6. Programmation : XML et Java (API DOM, SAX, ...)
7. Application : intégration de données

HTML versus XML



Objectifs du langage XSLT

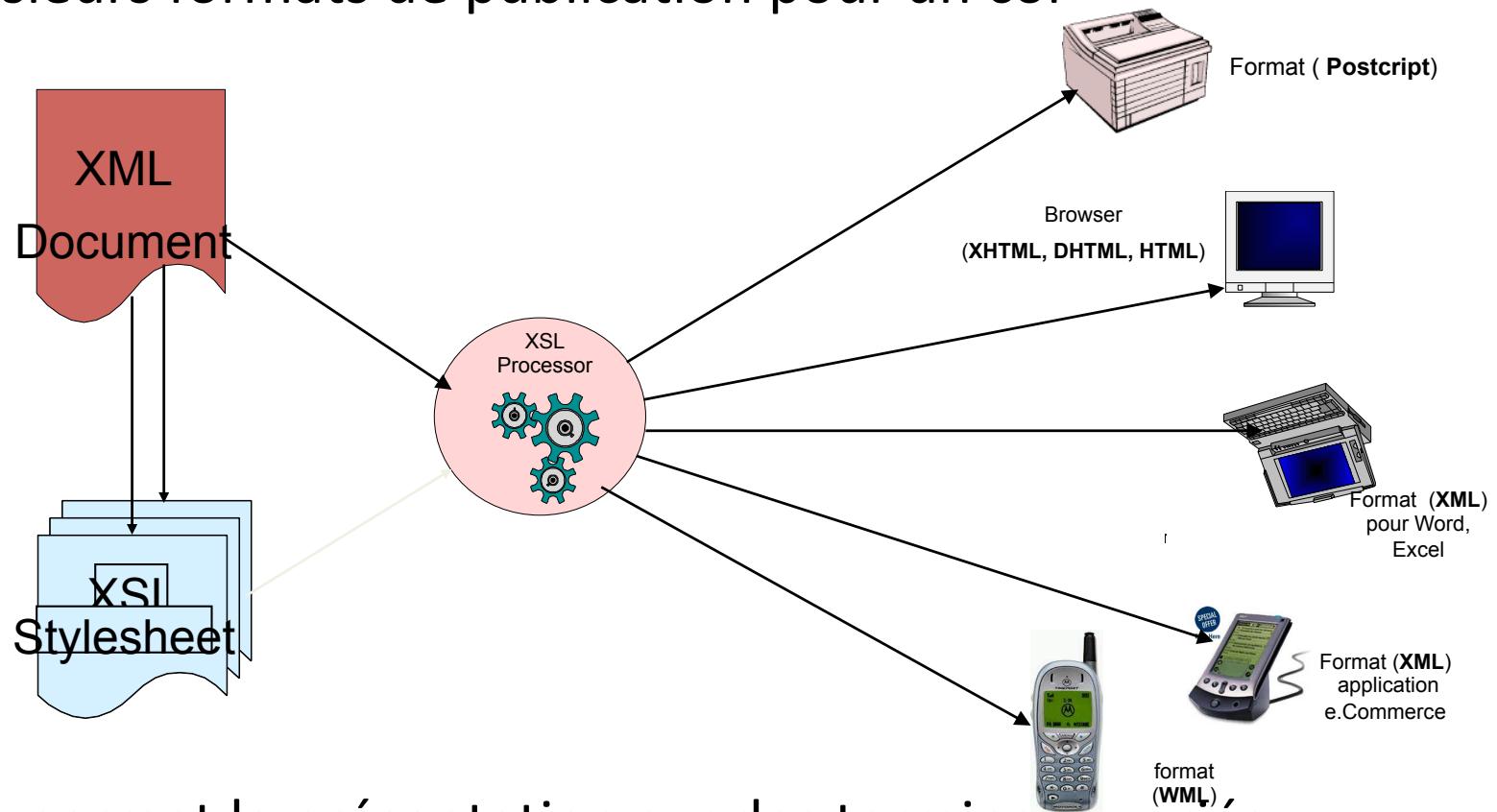
- XSLT permet de produire un nouveau document (XML, HTML, ...) par **transformation** d'un document XML existant.
- Une transformation s'exprime sous la forme d'une **feuille de style** qui est un document XML composé d'un ensemble de **règles** de transformation.
 - **Une règle comporte deux parties :**
 - un modèle de chemin exprimé en termes du langage **XPath**,
 - une transformation sous la forme d'une suite d'**instructions**.

Objectifs du langage XSLT

- L'espace de noms associé au langage XSLT est :
<http://www.w3.org/1999/XSL/Transform>
- dont le préfixe usuel est **xsl**.

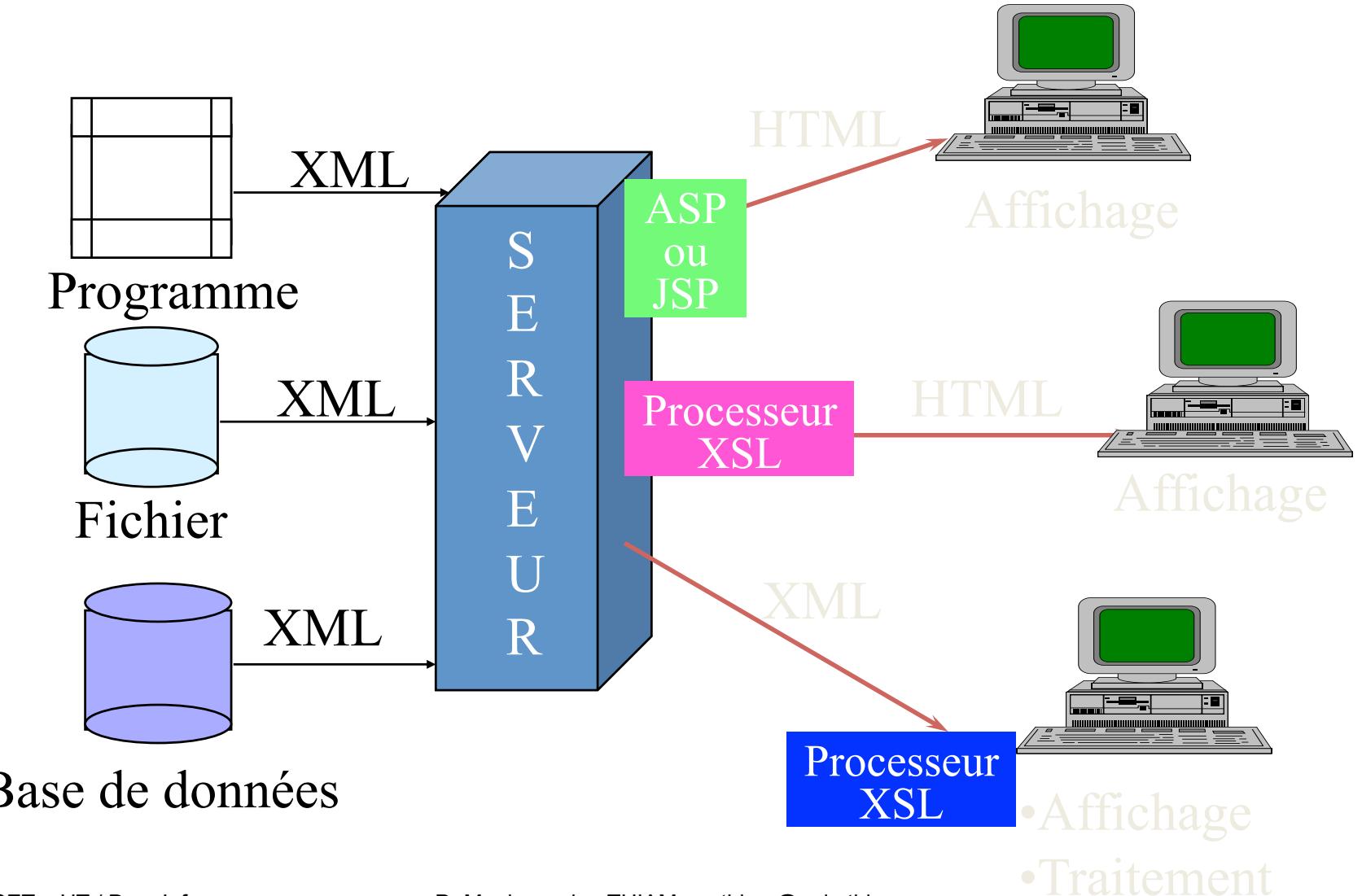
Publications avec XSL

- Plusieurs formats de publication pour un contenu

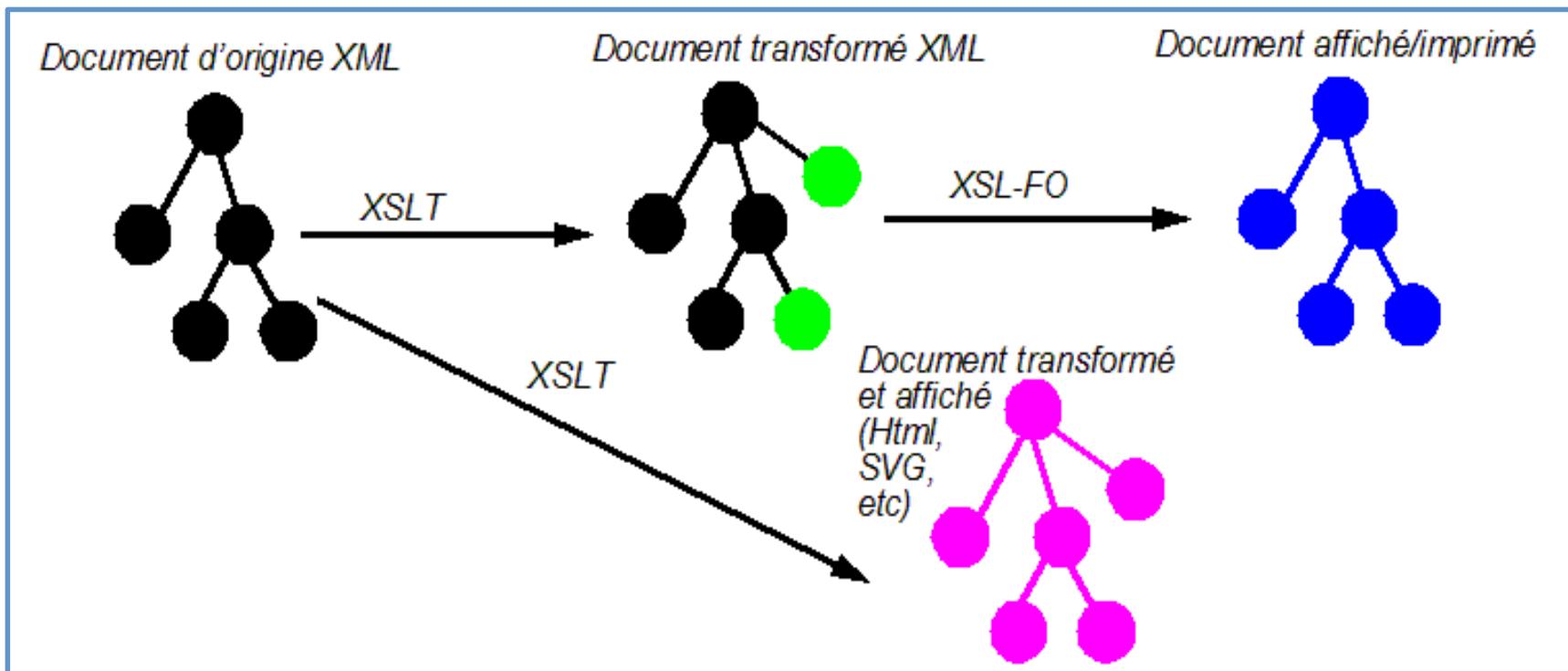


- XSL permet la présentation sur des terminaux variés

Architectures



Utilisation de XSLT



Travaux pratiques

- **Usine à gaz**
 - Utilisez vos IDE
 - NetBeans
 - Eclipse
 - XML Editor
- **SAXON**
 - Installer le processeur
 - En ligne tapez
 - **Transform -s:source -xsl:stylesheet -o:output**

Le document XSLT

- Un document XSLT est un document XML, on y trouve donc
 - Un prologue
 - Un corps
 - (Un épilogue)
- Le prologue
 - ✓ <?xml version="1.0" encoding="UTF-8"?>

Structure d'un document XSLT

• Le corps

- `<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">`
... liste de templates (règles)
`</xsl:stylesheet>`

Ou la formulation équivalente

- `<xsl:transform version="1.0"`
`xmlns:xsl="http://www.w3.org/1999/XSL/Transform">`
... liste de templates (règles)
`</xsl:transform>`

Règles

- Une règle (**template**) est décrite sous forme d'un élément XML :

```
<xsl:template match="p">  
    Transformation  
</xsl:template>
```

- où :
 - *p* est un chemin de localisation : une requête **XPath**,
 - **Transformation** une séquence de caractères ou d'éléments dont certains sont des instructions **XSLT**.

Règles

- D'autres attributs permettent :
 - de nommer une règle pour l'invoquer par ce nom,
 - de donner une priorité à une règle
 - Plus ce nombre est grand plus la règle a une priorité forte, ...

```
<xsl:template match="XML_tag_name"> ← Selector of XML elements  
    .... contents to produce (i.e. HTML) ← to transform  
    .... further instructions ← Contenus à produire  
</xsl:template> + autres instructions
```

Instruction

- Une instruction est décrite par un élément XML prédéfini dans le langage XSLT. Par exemple :

```
<xsl:text>  
    Bonjour  
</xsl:text>
```

- XSLT possède un jeu d'instructions très complet :
 - application d'une règle,
 - extraction de la valeur textuelle d'un élément,
 - instructions de branchement et de contrôle (IF, Then, ELSE)
 - instructions de tri et de groupement,
 - définitions de variables,
 - ...

XSLT : du XML vers le XHTML

- On crée un template pour la racine du document

```
<xsl:template match="/">
  <html>
    <head><title>TITRE</title></head>
    <body>
      HTML
      + constructions XSLT
      + des <xsl:template match= "requête XPath">
        ...
      </xsl:template>
    </body>
  </html>
</xsl:template>
```

Exemple

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform>
  <xsl:template match="/bdtheque/personnes">
    <html>
      <head>
        <title>Liste des personnes de la bdthèque</title>
      </head>
      <body>
        <h2>Les personnes </h2>
        <xsl:apply-templates select="personne" />
      </body>
    </html>
  </xsl:template>
  <xsl:template match="personne">
    <h2> <xsl:value-of select=".,"></h2>
  </xsl:template>
</xsl:stylesheet>
```

Moteur de transformation XSLT

- Le moteur de transformation :
 - opère sur :
 - l'arbre du document XML à transformer,
 - une liste de noeuds à traiter,
 - un noeud à traiter : le **noeud contexte** ;
 - produit un **flux de sortie** : en général un document XML ou HTML.
- Lancement du moteur :
 - La liste des noeuds à traiter est constituée d'un noeud unique : le noeud racine du document à traiter.
 - Appliquer les règles.

Application des règles

- Pour chaque noeud de la liste des noeuds à traiter, appelé **noeud contexte** :
 - Chercher les règles `<xsl:template match="p">t</...>` telles que le noeud contexte est l'extrémité du chemin *p*.
 - S'il y en a plusieurs choisir la plus spécifique c.-à-d. celle dont le chemin *p* est le plus précis.
 - Par exemple, la règle `<... match="vallon/nom">` est plus spécifique que la règle `<... match="vallon">`.
 - puis la plus prioritaire.
 - Appliquer la transformation *t* en parcourant son texte et :
 - en recopiant dans le flot de sortie tout caractère qui n'appartient pas à une instruction XSLT,
 - en exécutant les instructions XSLT.

Règles prédéfinies (par défaut)

- Les règles prédéfinies s'appliquent en l'absence de règles applicables définies dans la feuille de style. Elles ont une priorité + faible que celles-ci.
- Les deux principales sont :
 - règle prédéfinie **pour les noeuds racine et éléments**, provoquant la relance du traitement sur les noeuds fils du noeud contexte :

```
<xsl:template match= "*|/*">  
    <xsl:apply-templates/>  
</xsl:template>
```

- règle prédéfinie **pour les noeuds textes et attributs**, produisant la recopie de leurs valeurs dans le flot de sortie :

```
<xsl:template match= "text()|@*">  
    <xsl:value-of select=". . ."/>  
</xsl:template>
```

Quelques instructions classiques

- **<xsl:apply-templates/>**
 - La liste des noeuds à traiter est constituée des noeuds fils du noeud contexte.
 - Appliquer les règles.
- **<xsl:apply-templates select="p"/>**
 - La liste des noeuds à traiter est constituée des noeuds atteints par le chemin *p* depuis le noeud contexte.
 - Appliquer les règles.
- **<xsl:value-of select="p"/>**
 - Recopier dans le flot de sortie la valeur-chaîne de chaque noeud atteint par le chemin *p* depuis le noeud contexte.
- **<xsl:copy-of select="p"/>**
 - Recopier dans le flot de sortie le fragment du document à transformer dont la racine est le noeud atteint par le chemin *p* depuis le noeud contexte.
- **<xsl:text>t</xsl:text>**
 - Recopier dans le flot de sortie le texte *t*.

Encore des instructions XSL

1. Les fondamentaux

- a) xsl:stylesheet
- b) xsl:output
- c) xsl:template
- d) xsl:value-of

2. Ajout d'éléments et d'attributs

- a) xsl:element
- b) xsl:attribute
- c) Syntaxe courte

Encore des instructions XSL

1. Gestion des boucles

- a) xsl:for-each
- b) xsl:sort
- c) xsl:number

2. Conditions de test

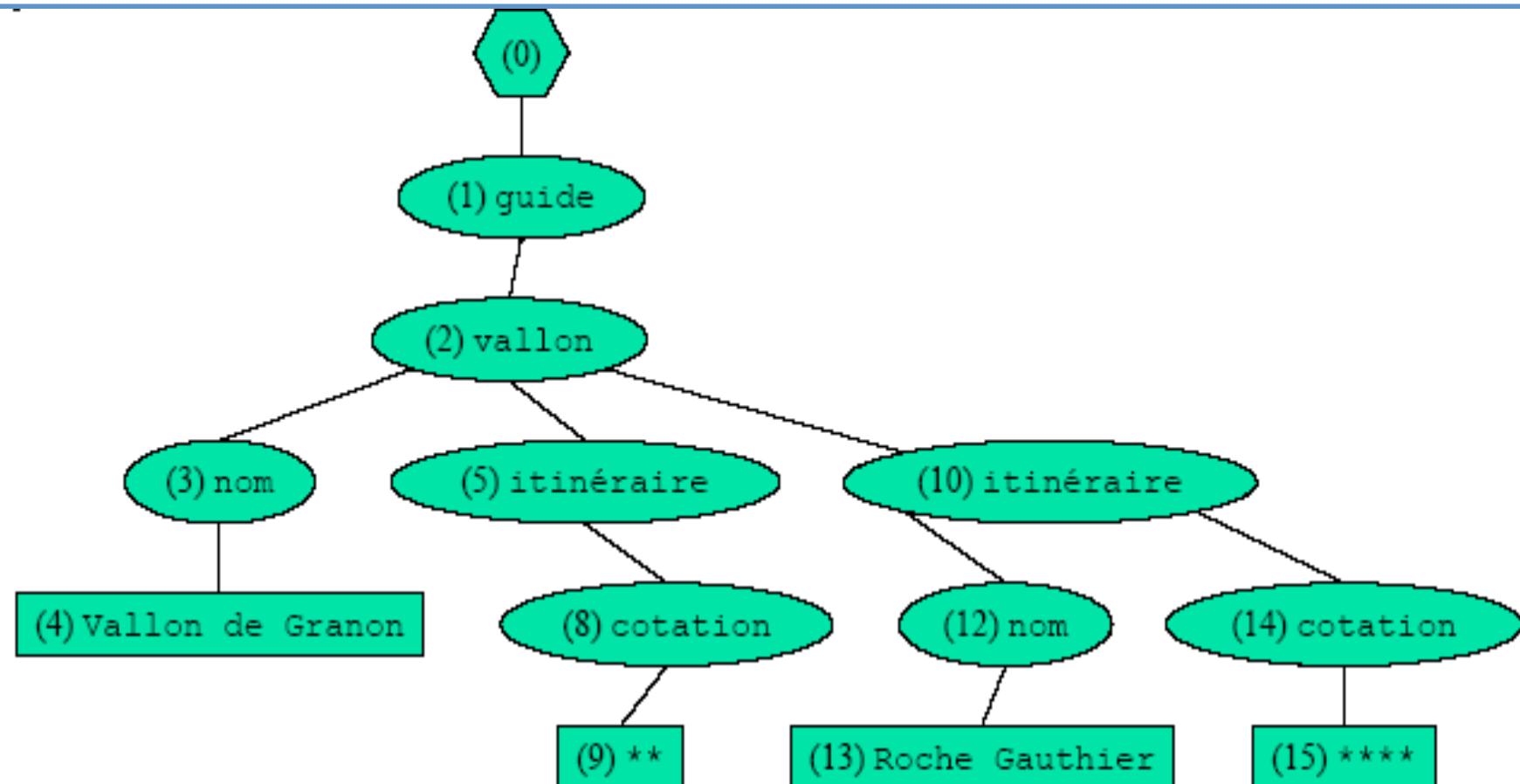
- a) xsl:if
- b) xsl:choose

Ex 1: Itinéraires les plus difficiles

- On veut produire un document XML listant les itinéraires **** avec pour chacun, son nom et le nom du vallon dans lequel il se déroule :

```
<best-of>
    <nom>Roche Gauthier (Vallon de Granon)</nom>
    <nom>Le Guyon (Vallon des Acles)</nom>
    <nom>Roche des prés (Vallon des Acles)</nom>
    <nom>Crête de Marapa (Vallon des Acles)</nom>
    <nom>Pointe des Rochers-Charniers (Vallon des Acles)</nom>
    <nom>Pic du Lauzin (Vallon des Acles)</nom>
    <nom>Pointe de Pécé (Vallon des Acles)</nom>
    ...
</best-of>
```

Ex 1 : fragment de l'arbre du document à transformer



Ex 1 : la feuille de style

```
1. <xsl:stylesheet version="1.0"
2.                         xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
3. <xsl:output method="xml" indent="yes"/>
4. <xsl:template match="/">
5.   <best-of>
6.     <xsl:apply-templates select="//itinéraire[cotation='****']"/>
7.   </best-of>
8. </xsl:template>
9. <xsl:template match="itinéraire">
10. <nom>
11.   <xsl:value-of select="nom"/>
12.   <xsl:text> (</xsl:text>
13.   <xsl:value-of select="ancestor::vallon/nom"/>
14.   <xsl:text>)</xsl:text>
15. </nom>
16. </xsl:template>
17. </xsl:stylesheet>
```

Ex 1: la transformation (sur l'extrait)

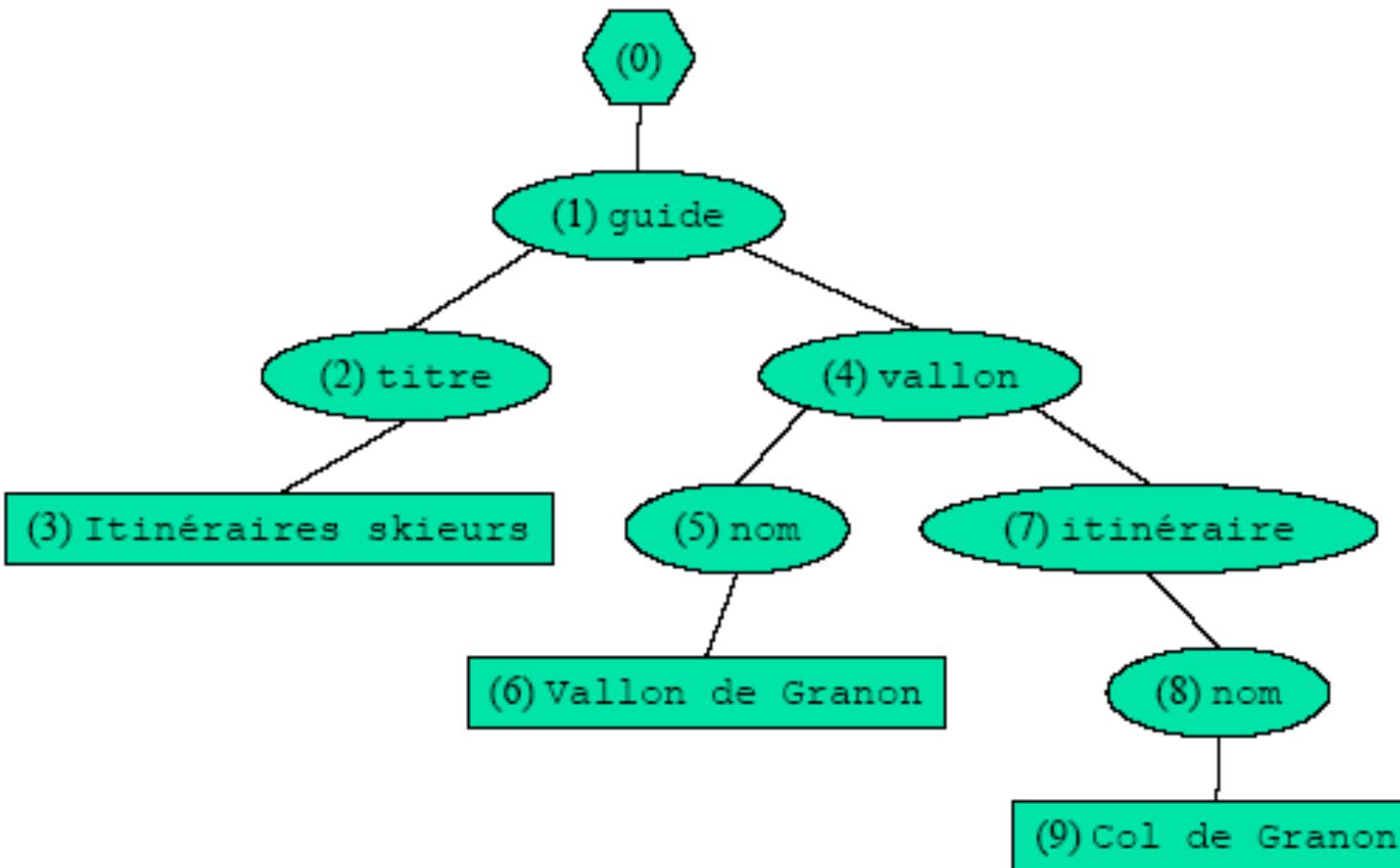
- La **règle 4** s'applique au noeud 0 :
 - on écrit <best-of> dans le flot de sortie (ligne 5).
 - on applique les règles aux noeuds vérifiant :
itinéraire[cotation='****'] soit 5.
- La règle 9 s'applique au noeud 5 :
 - on écrit <nom> dans le flot de sortie (ligne 10),
 - on écrit Roche Gauthier dans le flot de sortie (instruction 11),
 - on écrit (dans le flot de sortie (ligne 12),
 - on écrit Vallon de Granon dans le flot de sortie (instruction 13),
 - on écrit) dans le flot de sortie (ligne 14),
 - on écrit </nom> dans le flot de sortie (ligne 15).
- On écrit </best-of> dans le flot de sortie (ligne 7).

Ex 2 : liste des vallons et des itinéraires

- On veut produire un document XML listant les noms des vallons et de leurs itinéraires

```
<vallons>
  <vallon>
    <nom>Vallon de Granon</nom>
    <itinéraires>
      <nom>Col de Granon</nom>
      ...
    </itinéraires>
  </vallon>
  <vallon>
    <nom>Vallon des Acles</nom> ...
  </vallon>
  ...
</vallons>
```
- Cet exemple illustre l'emploi des règles prédéfinies.

Ex 2 : fragment de l'arbre du document à transformer



Ex 2 : la feuille de style

```
<xsl:stylesheet version="1.0"
2.           xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
3. <xsl:output method="xml" indent="yes"/>
4.<xsl:template match="/">
5.     <vallons>
6.         <xsl:apply-templates/>
7.     </vallons>
8. </xsl:template>
9. <xsl:template match="vallon">
10.    <xsl:copy-of select="nom"/>
11.    <itinéraires>
12.        <xsl:apply-templates/>
13.    </itinéraires>
14. </xsl:template>
15. <xsl:template match="itinéraire/nom">
16.    <xsl:copy-of select=". "/>
17. </xsl:template>
18. <xsl:template match="text()|@*"/>
19. </xsl:stylesheet>
```

Cette règle cache la règle pré définie pour les noeuds texte ou attribut. Elle empêche leur valeur d'être écrite dans le flot de sortie.

Ex 2 : la transformation (sur l'extrait)

- applique les règles aux noeuds 1 fils du noeud 0.
- La règle prédefinie sur les noeuds racine et élément s'applique au noeud 1 : il est ignoré puis on applique les règles aux noeuds 2 et 4 fils du noeud 1.
- La règle prédefinie sur les noeuds racine et élément s'applique au noeud 2 : il est ignoré et l'application des règles est relancée sur le noeud 3 fils du noeud 2.
- La règle 18 s'applique au noeud 3 : aucune action n'est réalisée.
- La règle 9 s'applique au noeud 4 : on écrit <nom>Vallon de Granon</nom> (instruction 10) puis <itinéraires> (ligne 11) dans le flot de sortie, puis on applique les règles aux noeuds 5, 7 et 11 fils du noeud 4.
- La règle prédefinie sur les noeuds racine et élément s'applique au noeud 5 : il est ignoré et l'application des règles est relancée sur le noeud 6 fils du noeud 5. La règle 18 s'applique au noeud 6 : aucune action n'est réalisée.
- La règle 15 s'applique au noeud 8 : on écrit <nom>Col de Granon</nom> dans le flot de sortie (instruction 16).
- On écrit </itinéraires> dans le flot de sortie (ligne 13).
- On écrit </vallons> dans le flot de sortie (ligne 7).

Autres instructions : boucles

- La boucle

```
<xsl:for-each select="requête XPath">
```

...

```
</xsl:for-each>
```

- Le corps du **for-each** est relatif à la racine du résultat de la requête XPath ou de la liste de valeurs du résultat

- Exemple

```
<xsl:for-each select='//personnes/personne'>
```

```
    <li><xsl:value-of select='personne' /> </li>
```

```
</xsl:for-each>
```

Autres instructions : tri, condition

- Les constructions XSLT

- Le tri

- `<xsl:sort select="requête XPath"/>`

- Les éléments sont atteints en fonction du critère de tri (valeur d'élément ou d'attribut) désigné par la requête

- La condition

- `<xsl:if test="condition XPath">`
...
`</xsl:if>`

Autres instructions : choix

- Les constructions XSLT

- Le choix

- <xsl:choose>

- <xsl:when test="condition XPath">

- ...

- </xsl:when>

- ...

- <xsl:otherwise>

- ...

- </xsl:otherwise>

- </xsl:choose>

XSL:CALL-TEMPLATE

- ```
<xsl:call-template name="nom_template">
 <xsl:with-param name="nom_param" select="expression" />
</xsl:call-template>
```

# Transformation XSL via un navigateur

- Pour transformer un document XML en utilisant une feuille de style XSL, ajouter l'instruction suivante après le prolog du document XML

<?xml version=....> :

<?xml-stylesheet type="text/xsl" href="fichier.xsl"?>

# Résumé XSLT

- XSLT est un langage très puissant de transformation d'un arbre XML en un autre
- XSLT permet en particulier de publier des données XML sur le Web par transformation en document HTML standard
- XSLT est très utilisé :
  - Pour publier des contenus XML
  - Pour transformer des formats (EAI, B2B)

END





# XML et ses applications : plan

1. Introduction à XML
2. Schémas de documents XML
  - a. DTD
  - b. XML Schema
3. Le modèle DOM
4. Interrogation : XPath
5. Transformation et présentation : XSLT
6. Programmation : XML et Java (API DOM, SAX, ...)
7. Application : intégration de données

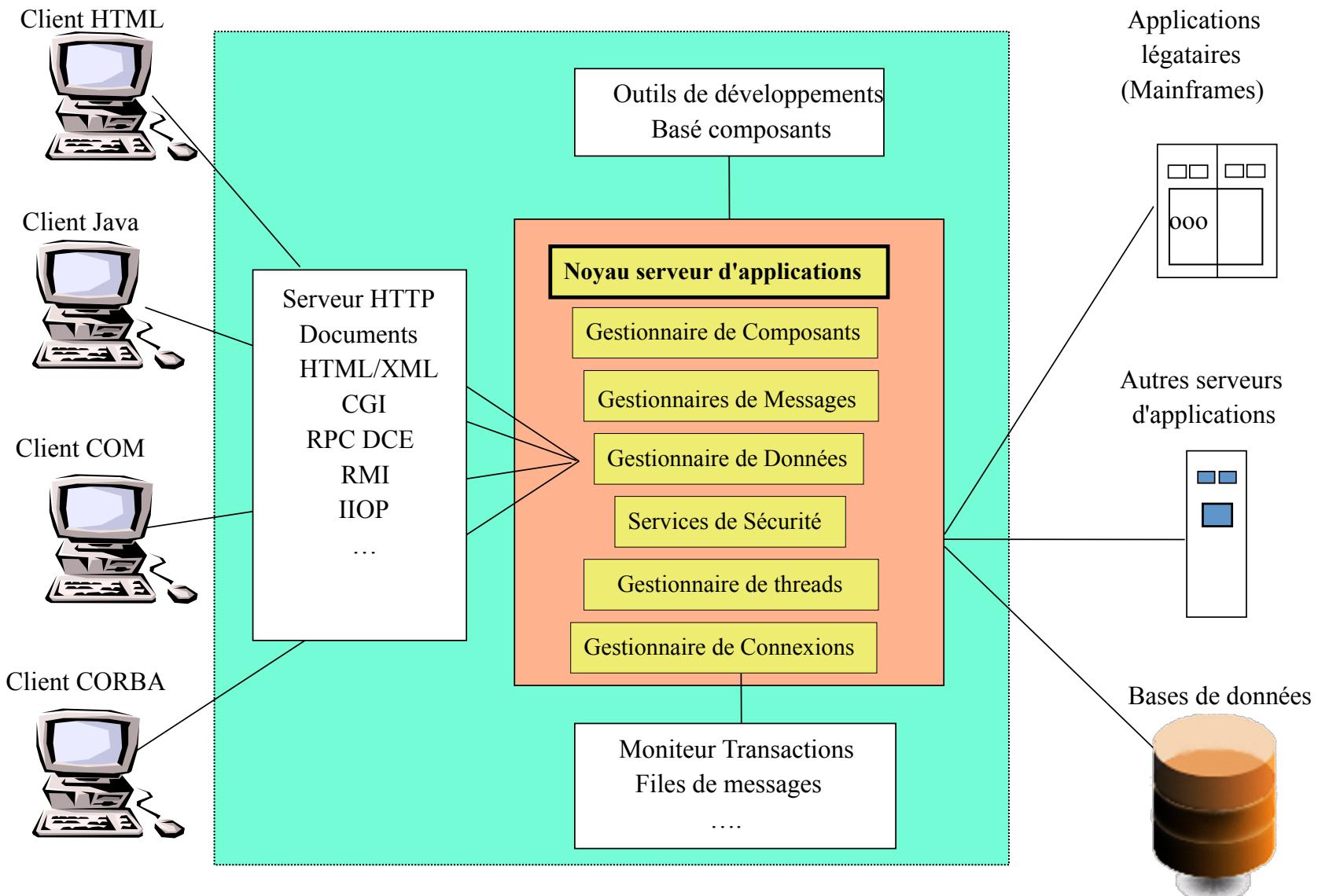
# Programmation : XML et Java

1. Introduction
2. Interface DOM
3. Interface SAX
4. Mapping objet - XML
5. Conclusion

# Introduction

- Java, un langage portable
- Mais aussi une plate-forme distribuée (J2EE)
  - XML déjà le standard de description des composants
  - XML devient le standard de communication
- .Net la plate-forme concurrente de Microsoft
- Avantages de XML:
  - échange sécurisé sur HTTP
  - un message porte plusieurs objets
  - interopérabilité des Services Web

# Architecture d'un serveur d'appli



# Principaux serveurs d'appli

- WebLogic de BEA
- WebSphere Application Server d'IBM
- Domino Application Server de Lotus -IBM
- iPlanet Application Server de Sun
- Oracle Application Server
- Sybase Enterprise Application
- Tomcat+JBoss (Open Source)
- Tomcat+Jonas (Open Source)
- Microsoft .NET, Vista & Indigo

# XML et la POO

- Les services sont souvent programmés en langage objet
  - Java, C++, C#, VB, PHP ...
- Java
  - langage pur objet portable et sûr, semi-interprété
  - compilateur produit un fichier de byte code par classe
  - possibilité de chargement dynamique de classe
  - support du standard de composants JB et EJB
- XML
  - messages XML doivent être traduits en objets
  - différents niveau d'interface :
    - Flux d'événements (SAX)
    - Traduction en objet génériques (DOM)
    - Mapping sur objets métiers (Data Binding)

# XML et autres langages

- VB, C#
  - Possibilités similaires à Java (parseurs MS)
- C++
  - Possibilités similaires à Java (parseurs libres)
  - Chargement dynamique de classe impossible
- PHP
  - Existence des parseurs libres

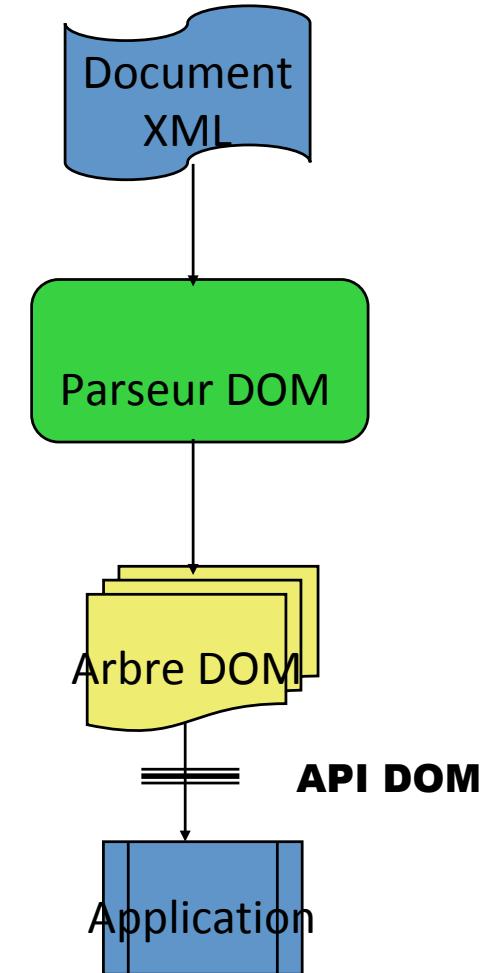
# **INTERFACE DOM**

# Présentation

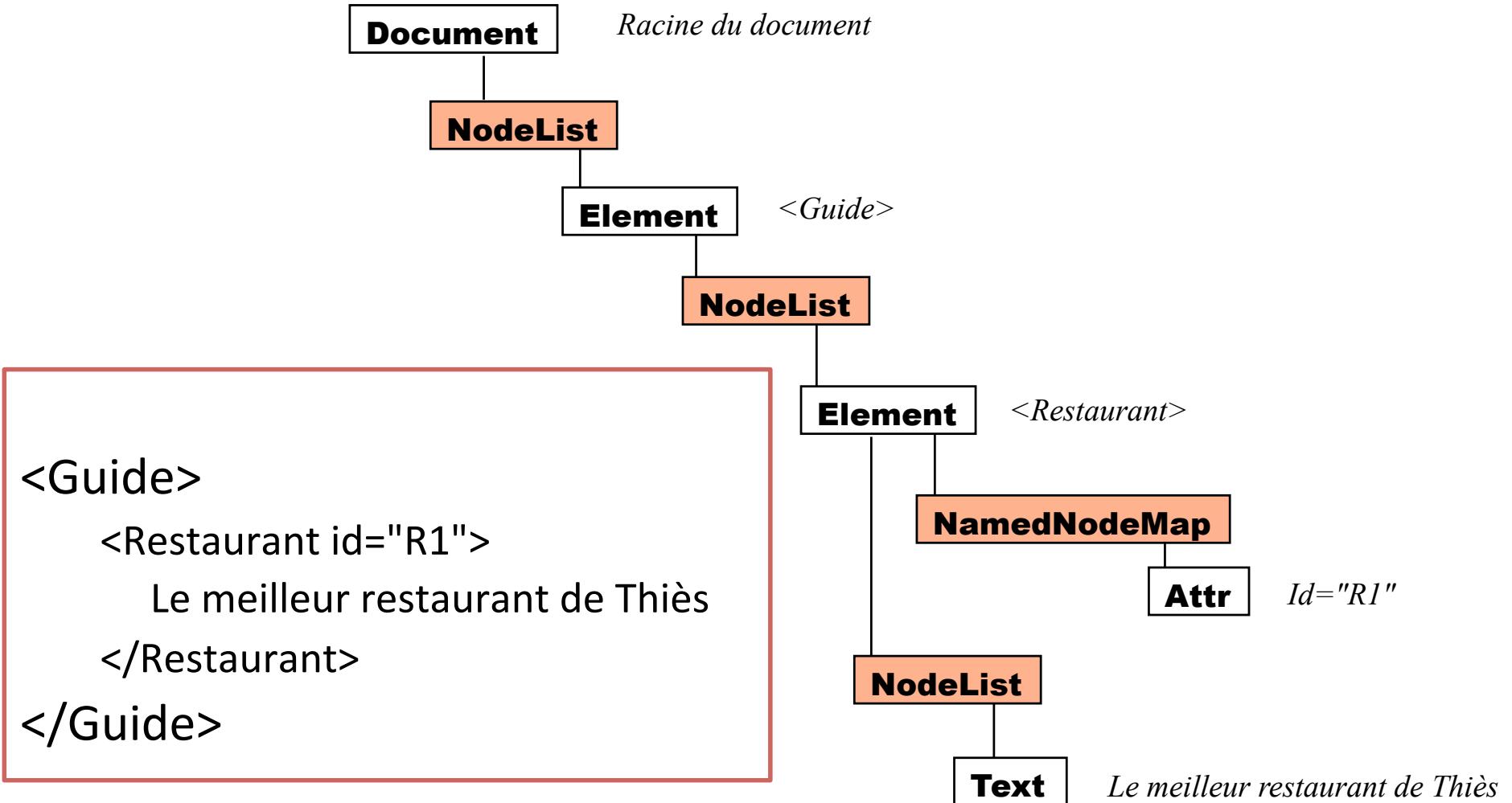
- Standard **W3C** fait pour **HTML** et **XML**
- Structure d'objets pour représenter un document
  - Résultat d'un "**parser**"
  - Arbre d'objets (reliés entre eux)
- Interface objet pour naviguer dans un document
  - Orientée objet
  - Peut être utilisée en:
    - Java, C++, C#, VB, Python, PHP

# Principaux parseurs

|            |                    |
|------------|--------------------|
| Xerces     | Apache (Java, C++) |
| MSXML      | Microsoft          |
| SDK Oracle | Oracle             |
| JAXP J2EE  | Sun, ...           |

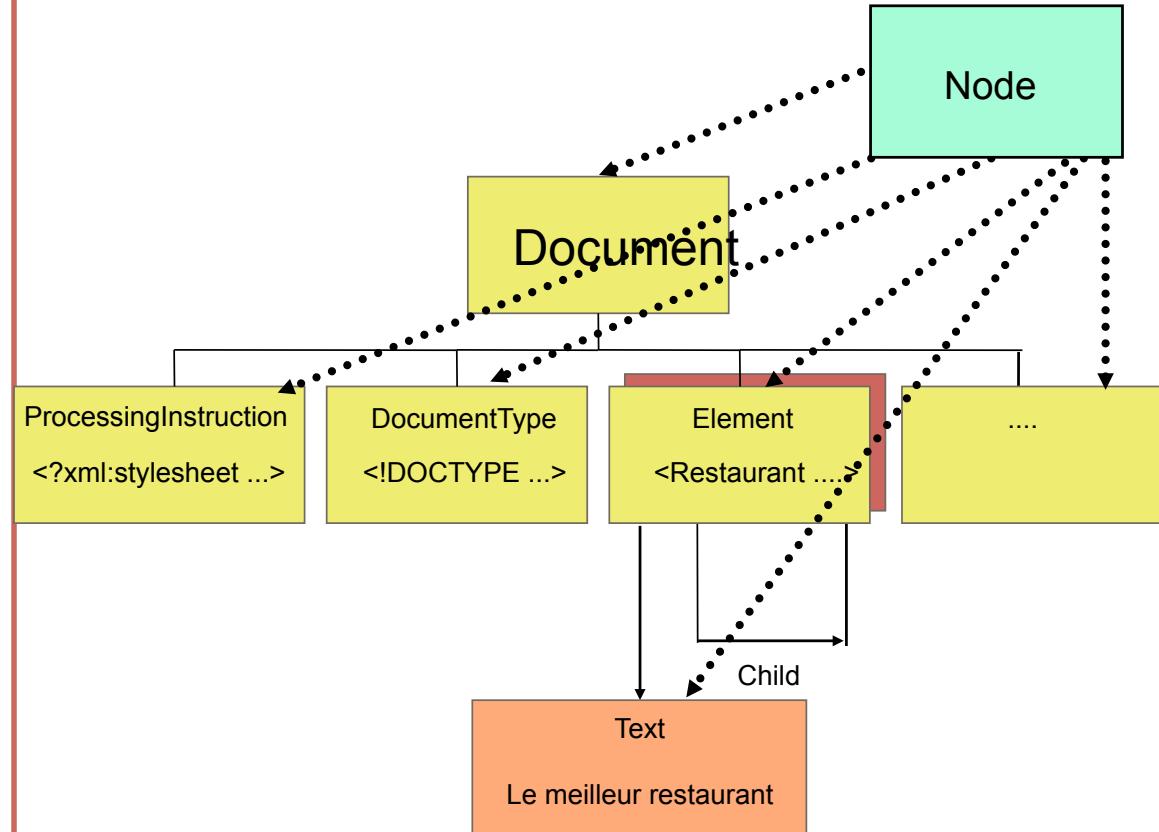


# Exemple d'arbre DOM



# Gestion de l'arbre DOM

- Navigation via un arbre générique de nœuds
  - Node
  - NodeList (Parent/Child)
  - NamedNodeMap
- Tout nœud hérite de Node



# Interface Document

- **createElement** (*Nom\_Element*):
  - créer un élément avec le nom spécifié en paramètre.
- **createComment** (*commentaire*):
  - créer une ligne de commentaires dans le document avec le texte *commentaire*.
- **createAttribute** (*Nom\_Attribut*):
  - créer un attribut avec le nom pris en paramètre.
- **getElementsByTagName** (*nom\_Tag*):
  - retourne tous les descendants des éléments correspondants au *Nom\_Tag*.

# Interface Node

- **insertBefore** (*Nouveau\_Noeud, Noeud\_Reference*):
  - insère un nouveau nœud fils avant le " nœud référence" déjà existant.
- **replaceChild** (*Nouveau\_Noeud, Ancien\_Noeud*):
  - remplace le nœud "*Ancien\_Noeud*" par le nœud "*Nouveau\_Noeud*".
- **removeChild** (*Nœud*):
  - supprime le nœud entré en paramètre de la liste des nœuds.
- **appendChild** (*Nouveau\_Noeud*):
  - Ajoute un nouveau nœud à la fin de la liste des nœuds.
- **hasChildNodes()**:
  - Retourne *VRAI* si le nœud possède un enfant et *FAUX* sinon

# Interfaces fondamentales

- DOMImplementation
- Document
- Comment
- DocumentFragment
- Element
- Attr(ibute)
- NamedNodeMap
- CharacterData
  - Comment
  - Text

# Autres étendues XML

- ProcessingInstruction
- DocumentType
- CDATASection
- Notation
- Entity
- EntityReference

# Exemple simple

- ```
import javax.xml.parsers.*;
import org.w3c.dom.*;
```
- ```
public class DomExample {
 public static void main(String args[]) {
 try {
 ...programme principal est inséré ici...
 } catch (Exception e) {
 e.printStackTrace(System.out);
 }
 }
}
```

# Création du parseur

- Appel de “**DocumentBuilder**” pour créer un parseur DOM
- Parseur non construit par constructeur mais avec une *méthode statique*
  - Bonne technique en programmation avancée en Java
  - Facilite le changement de parseur

```
DocumentBuilderFactory factory =
 DocumentBuilderFactory.newInstance();
```

```
DocumentBuilder builder =
 factory.newDocumentBuilder();
```

# Chargement du document XML

- Ajouter la ligne suivante au programme  
`Document document =  
builder.parse("hello.xml");`
- Notes:
  - `document` contient la totalité du fichier XML (sous forme d'arbre); c'est le Document Object Model
  - En ligne de commande le fichier XML dans le même répertoire
  - Si `java.io.FileNotFoundException`, c'est lié à sa visibilité

# Lecture du texte du document XML

- Element root =  
document.getDocumentElement();
- Node textNode = root.getFirstChild();
- System.out.println(textNode.getNodeValue());

# Programme entier

```
import javax.xml.parsers.*;
import org.w3c.dom.*;

public class SecondDom {
 public static void main(String args[]) {
 try {
 DocumentBuilderFactory factory =
 DocumentBuilderFactory.newInstance();

 DocumentBuilder builder =
 factory.newDocumentBuilder();

 Document document = builder.parse("hello.xml");

 Element root = document.getDocumentElement();

 Node textNode = root.getFirstChild();

 System.out.println(textNode.getNodeValue());

 } catch (Exception e) {
 e.printStackTrace(System.out);
 }
 }
}
```

# Illustration

```
import javax.xml.parsers.*;
import java.io.IOException;
import org.w3c.dom.*;

public class ExempleDOM {
 public static void main(String argc[]) throws IOException, DOMException {
 DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
 DocumentBuilder builder;
 try {
 builder = factory.newDocumentBuilder();

 Document xmlDoc = builder.newDocument();
 // creation des noeuds
 Element nom = (Element) xmlDoc.createElement("nom");
 Element prenom = (Element) xmlDoc.createElement("prenom");
 Element nomfam = (Element) xmlDoc.createElement("nomfam");
 // creation de l'arbre
 xmlDoc.appendChild(nom);
 nom.appendChild(prenom);
 prenom.appendChild(xmlDoc.createTextNode("Jean"));
 nom.appendChild(nomfam);
 nomfam.appendChild(xmlDoc.createTextNode("Dupont"));
 // positionnement d'un attribut
 nom.setAttribute("ville", "Paris");
 } catch (ParserConfigurationException e) {
 // TODO Auto-generated catch block
 e.printStackTrace();
 }
 // sortie
 System.exit(0);
 }
}
```

Document:

<nom ville ="Paris">

<prenom> Jean </prenom>

<nomfam> Dupont </nomfam>

</nom>

# Résumé (1)

- Une interface objet standard
  - Navigation dans l'arbre XML
  - Traitements spécifiques
- Performance limitée
  - Place mémoire importante
  - Traitement à la fin de l'analyse
- DOM 2.0
  - Accède dynamiquement au contenu et à la structure du document

# Résumé (2)

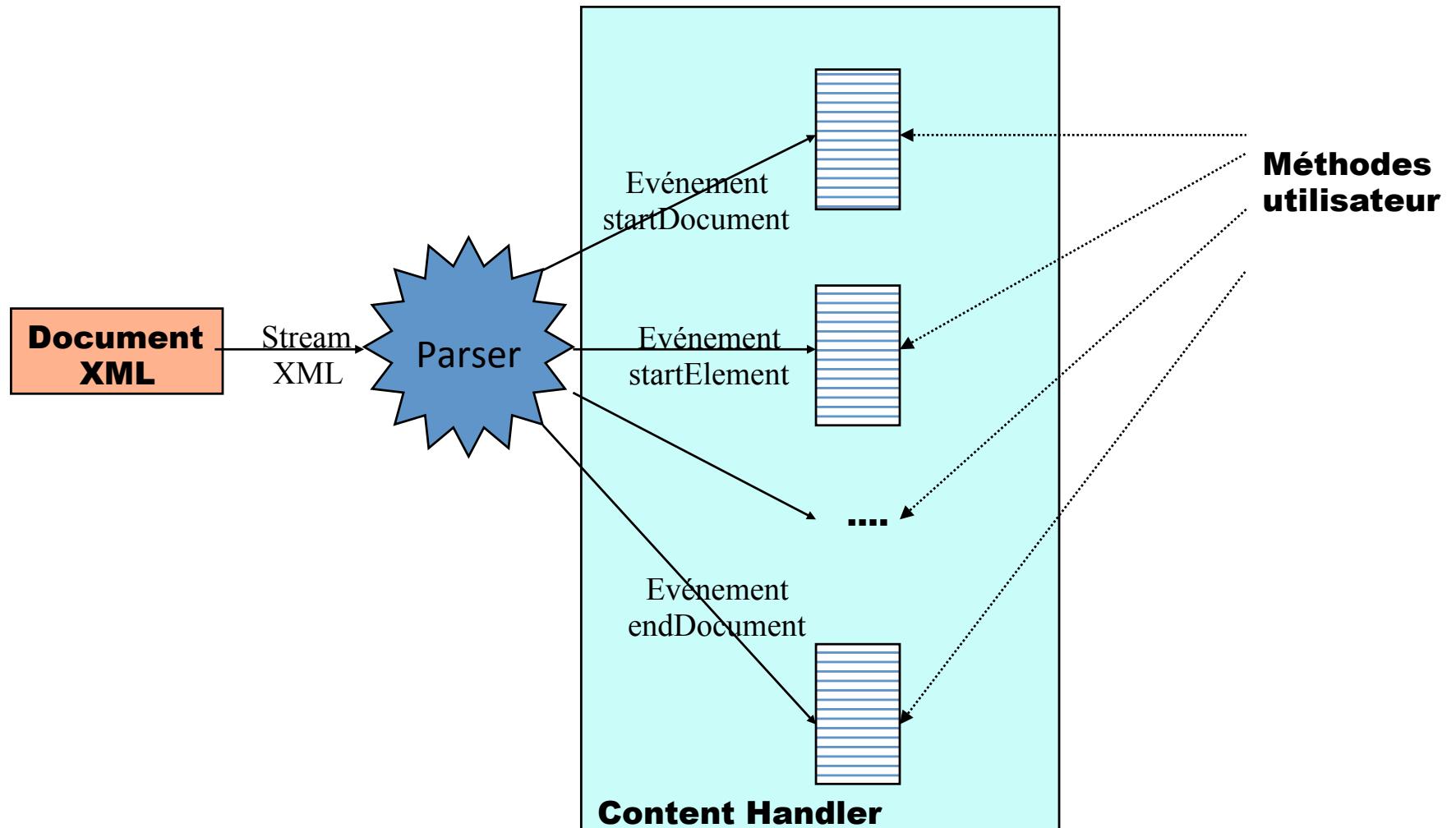
- Extensions en cours :
  - DOM Level 3 : **XPath**
    - Support direct de XPath
    - Définition d'un XPath Evaluator
    - Devrait être intégré aux parsers
  - DOM Level 3 : **Events**
    - Modèle d'événements
    - Associés à un nœud
    - Propagés dans l'arbre DOM
  - DOM Level 3 : **Style**
    - Accès aux styles
    - Mapping complet

# **INTERFACE SAX**

# L'interface SAX

- SAX (**S**imple **A**PI for **X**ML)
  - modèle simplifié d'événements
  - développé par un groupe indépendant.
- Types d'événement :
  - début et fin de document ;
  - début et fin d'éléments ;
  - attributs, texte, ... .
- Nombreux parseurs publics
  - XP de James Clark, Alfred, Saxon
  - MSXML3 de Microsoft
  - Xerces de Apache
  - JAXP de SUN

# Principe de fonctionnement

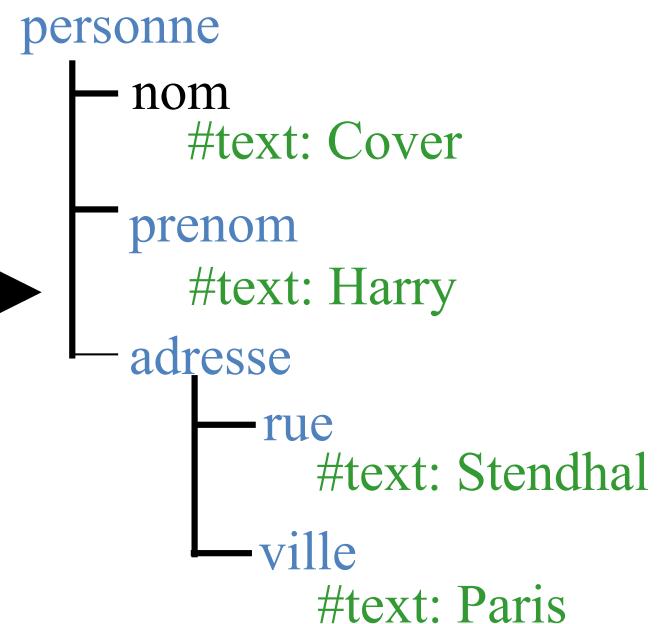


# Les méthodes essentielles

- **XMLReader**
  - setContentHandler
  - setErrorHandler
  - parse
- **ContentHandler**
  - startDocument
  - endDocument
  - startElement
  - endElement
  - characters
- **ErrorHandler**
  - fatalError
  - error
  - Warning

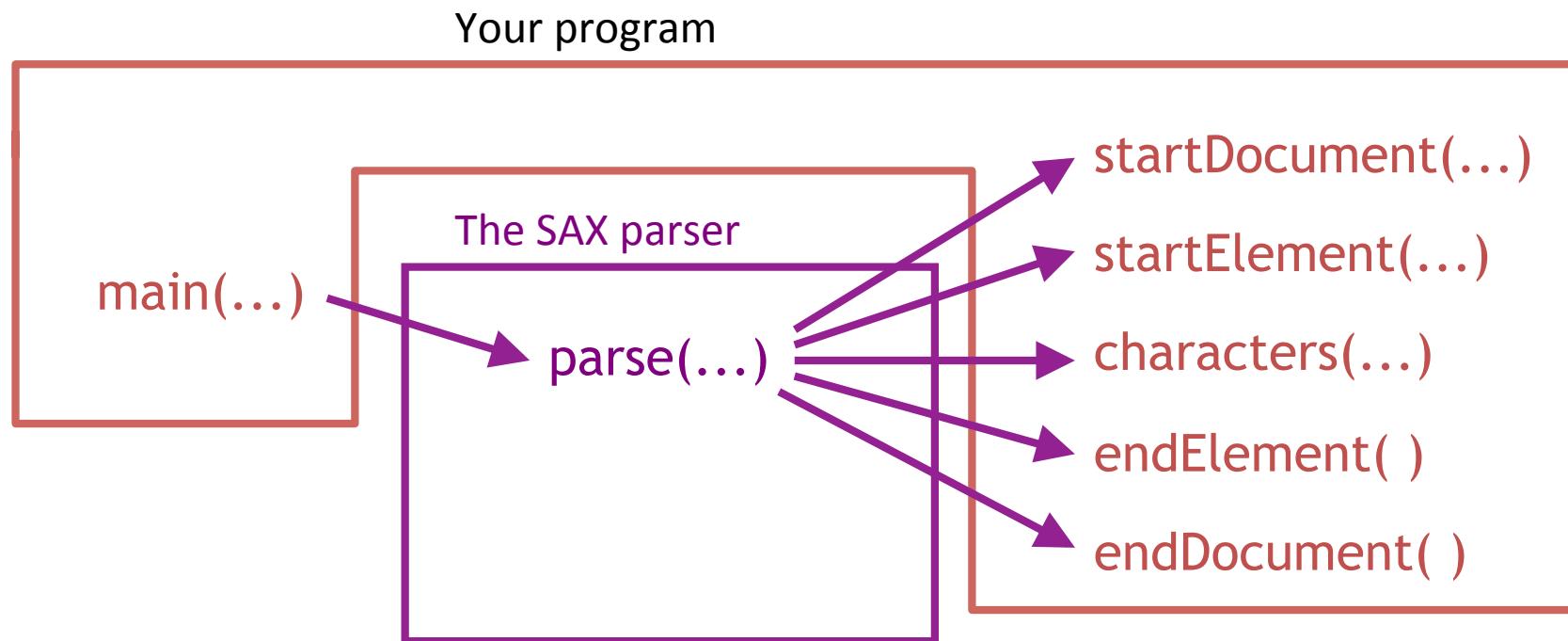
# Exemple SAX vs DOM

|             |   |                                 |
|-------------|---|---------------------------------|
| <personne>  | → | <i>startDocument ()</i>         |
| <nom>       | → | <i>startElement (personne)</i>  |
| Cover       | → | <i>startElement (nom)</i>       |
| </nom>      | → | <i>characters (Cover)</i>       |
| <prenom>    | → | <i>endElement (nom)</i>         |
| Harry       | → | <i>startElement (prenom)</i>    |
| </prenom>   | → | <i>characters (Harry)</i>       |
| <adresse>   | → | <i>endElement (prenom)</i>      |
| <rue>       | → | → <i>startElement (adresse)</i> |
| Stendhal    | → | <i>startElement (rue)</i>       |
| </rue>      | → | <i>characters (Stendhal)</i>    |
| <ville>     | → | <i>endElement (rue)</i>         |
| Paris       | → | <i>startElement (ville)</i>     |
| </ville>    | → | <i>characters (Paris)</i>       |
| </adresse>  | → | <i>endElement (ville)</i>       |
| </personne> | → | <i>endElement (adresse)</i>     |
|             | → | <i>endElement (personne)</i>    |
|             | → | <i>endDocument ()</i>           |



# Fonctionnement

- SAX fonctionne sur la base d'appels : appel au parseur qui appelle les méthodes nécessaires



# Simple programme SAX

Le programme comporte 2 classes:

- **SimpleSAXExemple** – contient la méthode **main**
  - Crée le **factory** des parseurs
  - Récupère un **parseur** avec ce dernier
  - Crée un objet **Handler** qui gère les appels du parseur
  - Associe le handler
  - Lie et parse le flux XML
- **Handler** – contient les handlers pour 3 évènements:
  - **startElement** début de document
  - **endElement** fin élément
  - **characters** texte rencontré

# SimpleSAXExemple

- ```
import javax.xml.parsers.*; // for both SAX and DOM
import org.xml.sax.*;
import org.xml.sax.helpers.*;
```
- ```
// For simplicity, we let the operating system handle exceptions
// In "real life" this is poor programming practice
public class SimpleSAXExemple {
 public static void main(String args[]) throws Exception {
```
- ```
        // Create a parser factory
        SAXParserFactory factory = SAXParserFactory.newInstance();
```
- ```
 // Tell factory that the parser must understand namespaces
 factory.setNamespaceAware(true);
```
- ```
        // Make the parser
        SAXParser saxParser = factory.newSAXParser();
        XMLReader parser = saxParser.getXMLReader();
```

SimpleSAXExemple (suite)

- Crédit slide précédent du parser de type **XMLReader**
- ```
// Create a handler
Handler handler = new Handler();
```
- ```
// Tell the parser to use this handler  
parser.setContentHandler(handler);
```
- ```
// Finally, read and parse the document
parser.parse("hello.xml");
```
- ```
} // end of Sample class
```

Remarques

- Vous aurez besoin de `hello.xml` :
 - En ligne de commande le fichier XML dans le même répertoire
 - Si `java.io.FileNotFoundException`, c'est lié à sa visibilité

La classe Handler

- `public class Handler extends DefaultHandler {`
 - `DefaultHandler` définit les méthodes de gestion
 - 3 méthodes pour gérer
 - Chaque méthodes peut provoquer une `SAXException`
- `// SAX calls this method when it encounters a start tag`
`public void startElement(String namespaceURI,`
`String localName,`
`String qualifiedName,`
`Attributes attributes)`
`throws SAXException {`
`System.out.println("startElement: " + qualifiedName);`
`}`

La classe Handler (suite)

- // SAX calls this method to pass in character data

```
public void characters(char ch[], int start, int length)
    throws SAXException {
    System.out.println("characters: \\" + 
        new String(ch, start, length) + "\\\");
```
 - // SAX call this method when it encounters an end tag

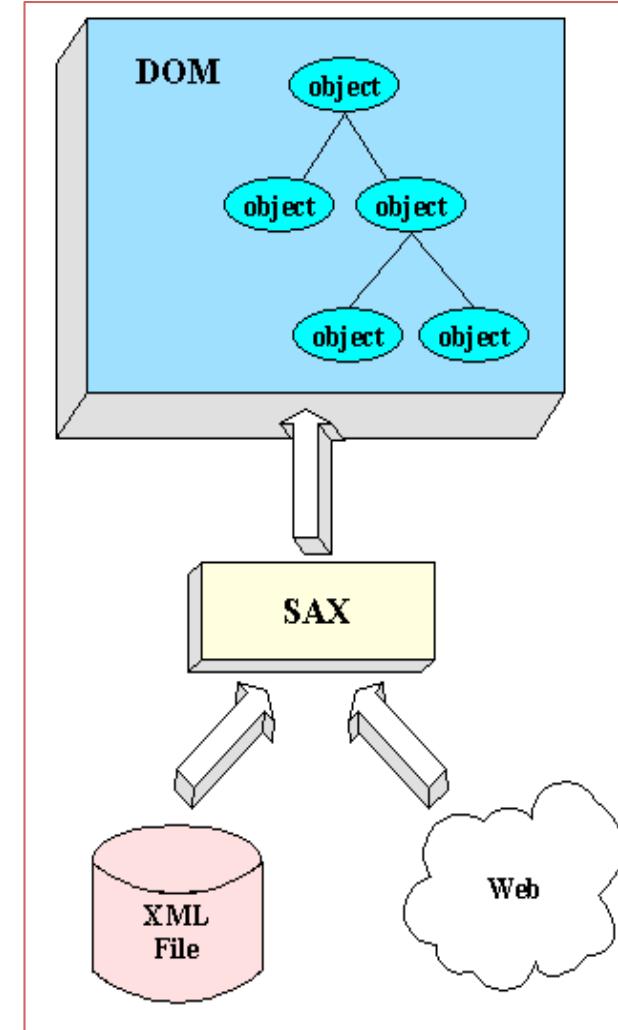
```
public void endElement(String namespaceURI,
    String localName,
    String qualifiedName)
    throws SAXException {
    System.out.println("Element: /" + qualifiedName);
}
```
- } // End of Handler class

Resultats

- Si hello.xml contient:
`<?xml version="1.0"?>`
`<display>Hello World!</display>`
- Le résultat est :
`startElement: display`
`characters: "Hello World!"`
`Element: /display`

DOM versus SAX

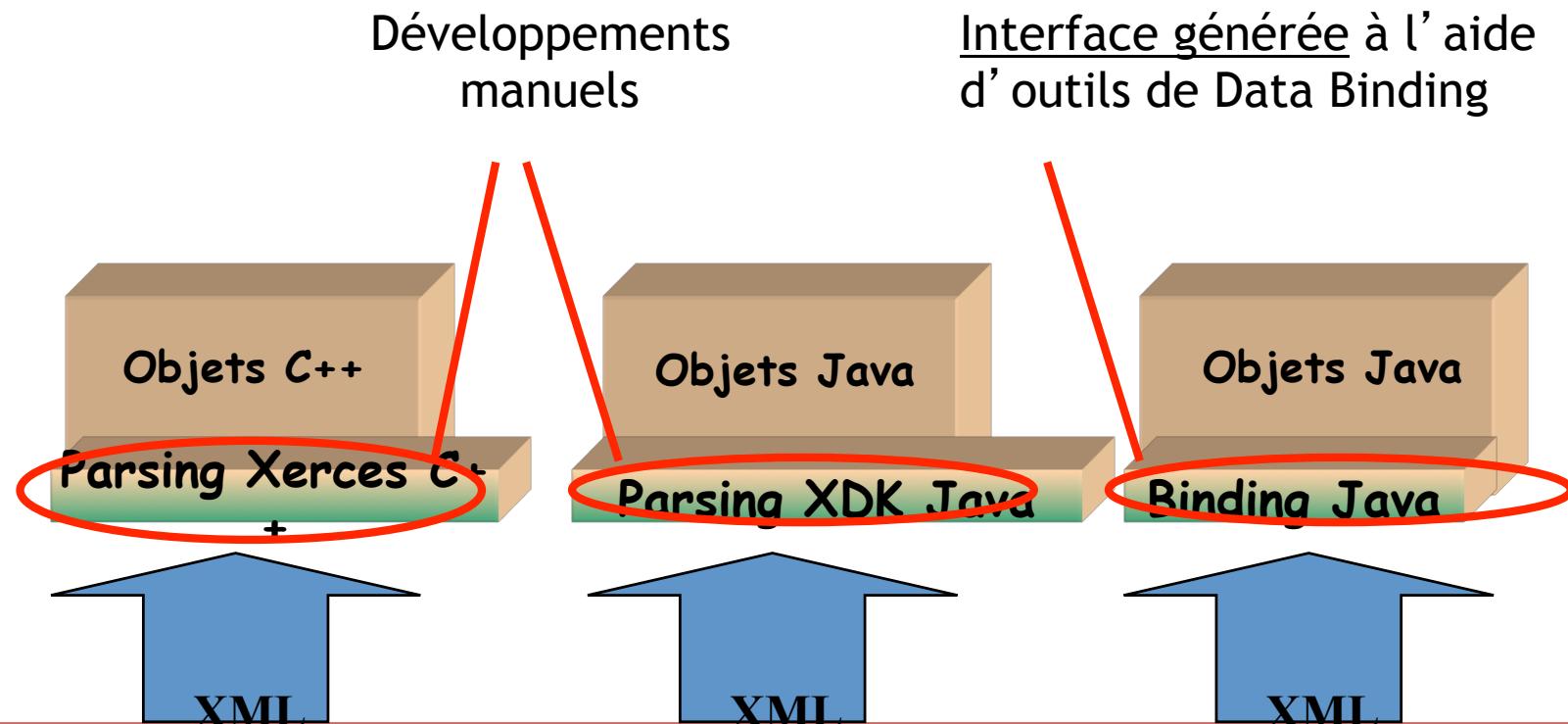
- DOM utilise SAX pour la construction de l'arbre d'un document XML
- SAX est plus léger que DOM
- Au-dessus de DOM, il est possible d'implémenter un « method caller » ...



MAPPING OBJET - XML

Le mapping objet (Data Binding)

- Définition
 - Mapping entre un document XML et un graphe d'objets métier C++ ou Java



Avantages

- Activité de développement "**parsing**" réduite
 - Gain en coût de développement des interfaces
 - Applications peuvent se concentrer sur le fonctionnel
 - Développeurs n'ont pas à se soucier du mapping XML
- Systèmes Techniques simplifiés
 - Capitalisation sur l' interface (utilisation, évolution, ...)
 - Gain en terme de fiabilité (fin de l' utilisation de parseurs variés, parfois écrits à la main)
 - Performances de l' interface sont optimisées
- Outils de mapping intégrés aux serveurs
 - J2EE = JAXB
 - .NET = mapper

Comment traiter les tags XML ?

- Tags connus
 - déclenchement d'un traitement prédéfini
 - appel de méthode associée avec paramètres
 - transformation et stockage en BD
 - génération de requête base de données
- Tags inconnus
 - les ignorer : pas toujours satisfaisant
 - les interdire (DTD, Schéma) : peu flexible
 - leur appliquer un traitement standard : affichage, erreur, etc.
 - rechercher le traitement approprié : bibliothèque Java

XML: synthèse des outils

