



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

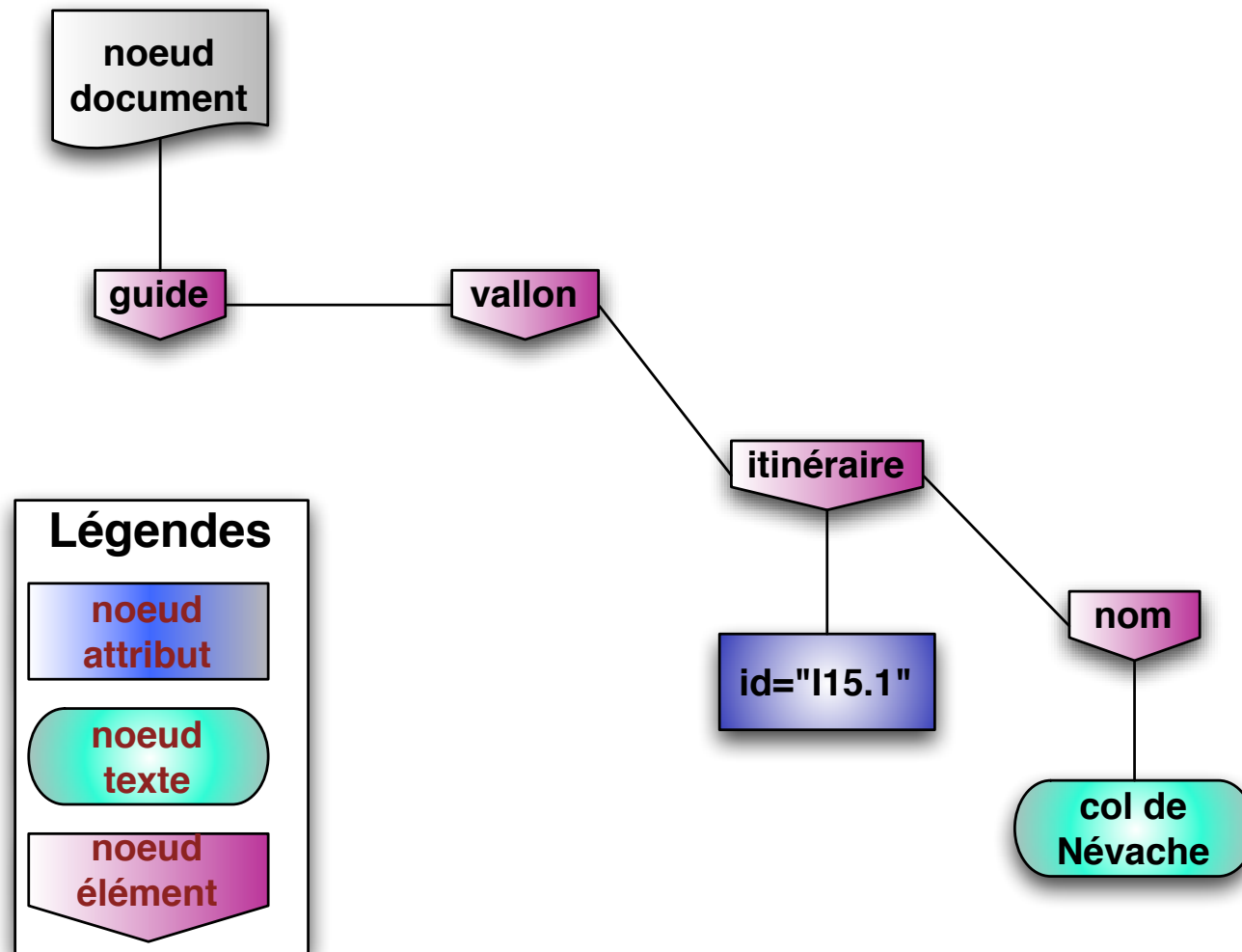
Évènements capturant du DOM

- 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 **fil**s 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

?





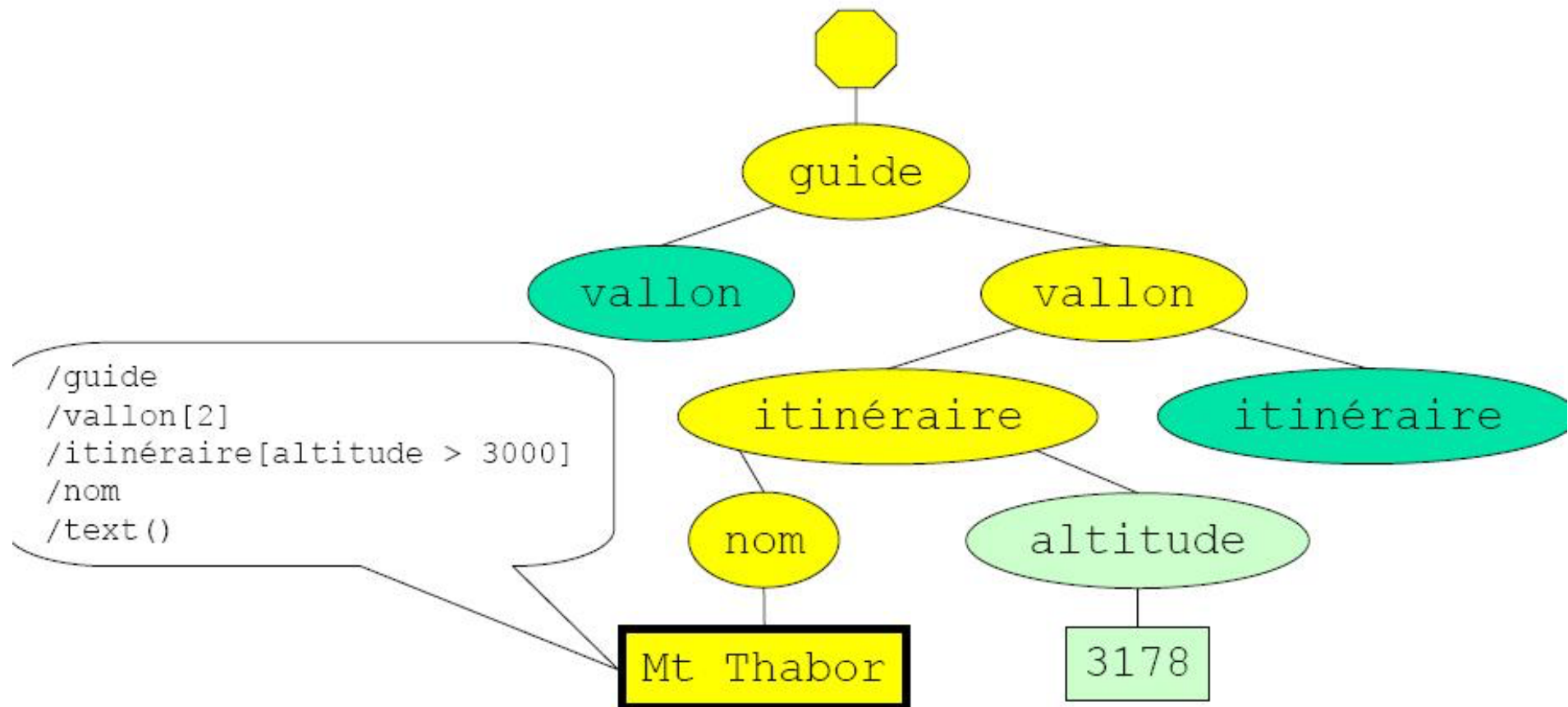
XML et ses applications : plan

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 :**
 - *$pas\ de\ localisation_1 / \dots / pas\ de\ localisation_n$*

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 **prédicats**.
- **Syntaxe :**
 - *axe::test de noeud prédicat₁... prédicat_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, ...)

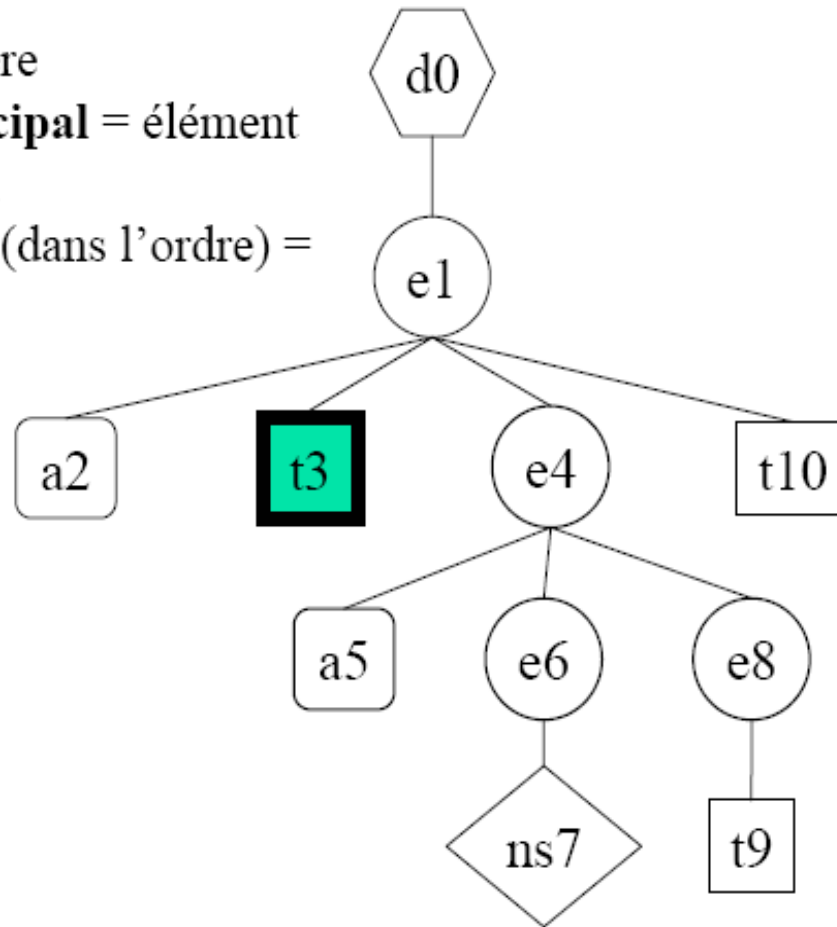
Axe : self

Sens = avant ou arrière

Sorte de nœud principal = élément

Noeud contexte = e1

Nœuds sélectionnés (dans l'ordre) =
e1



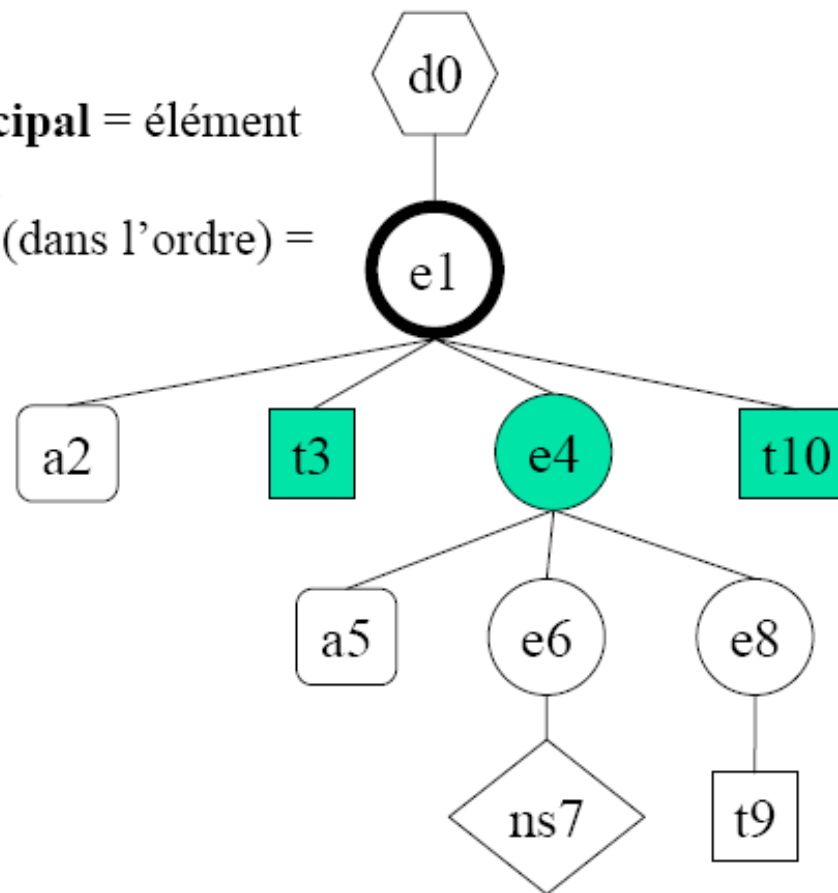
Axe : child

Sens = avant

Sorte de nœud principal = élément

Noeud contexte = e1

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



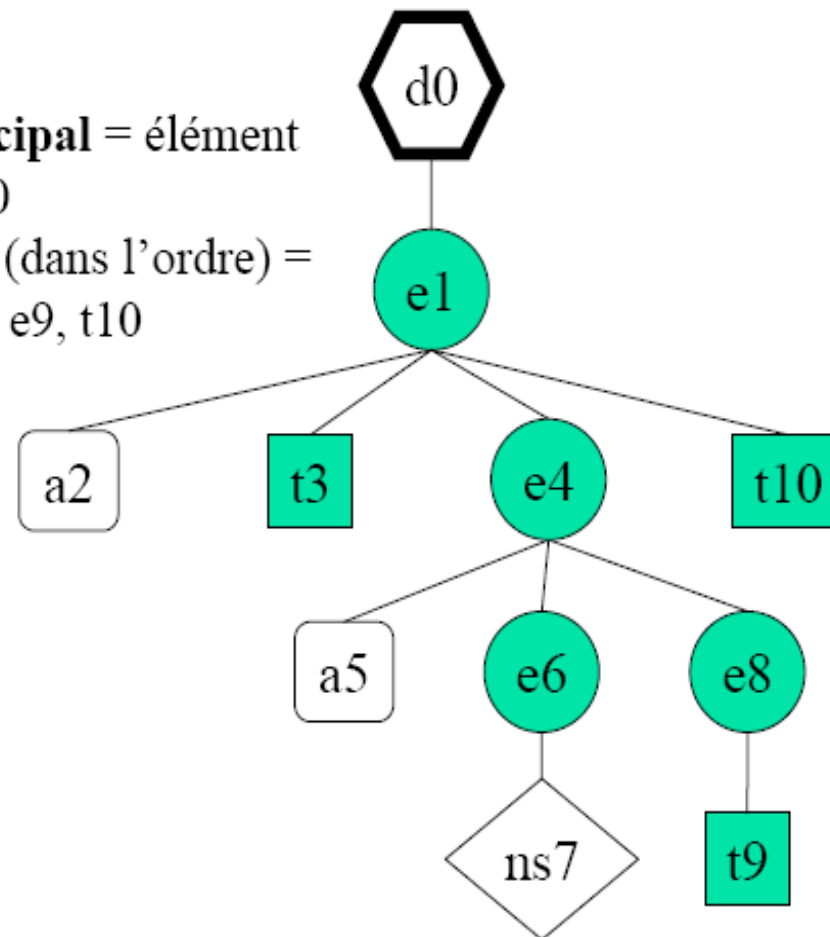
Axe : descendant

Sens = avant

Sorte de nœud principal = élément

Noeud contexte = d0

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



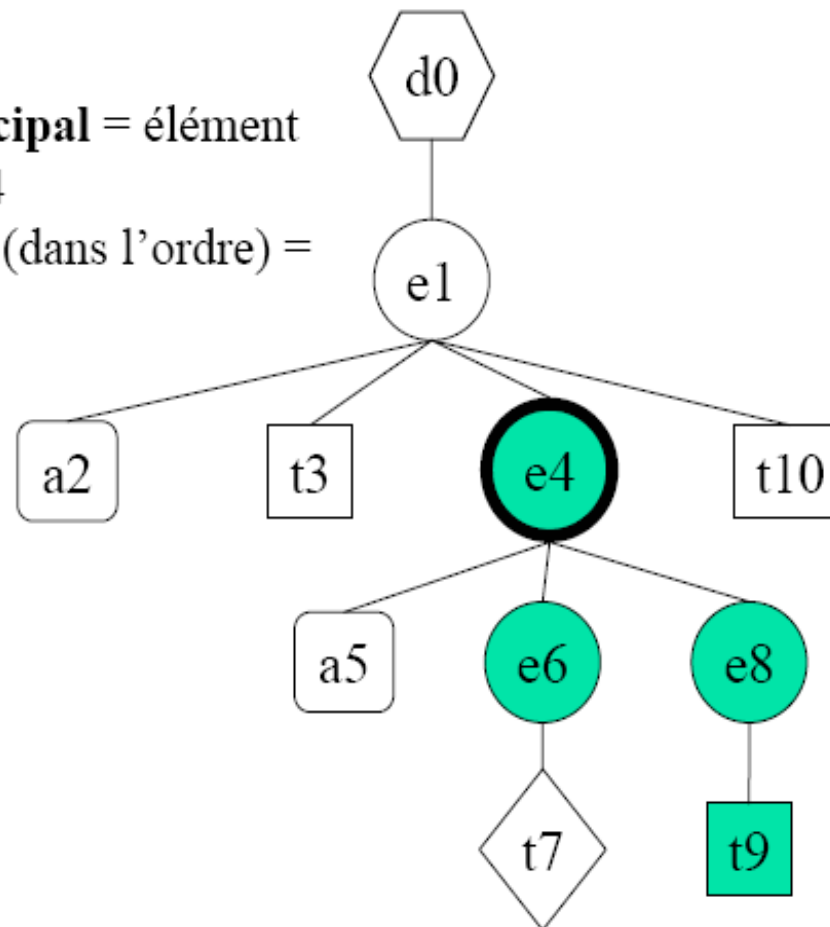
Axe : descendant-or-self

Sens = avant

Sorte de nœud principal = élément

Noeud contexte = e4

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



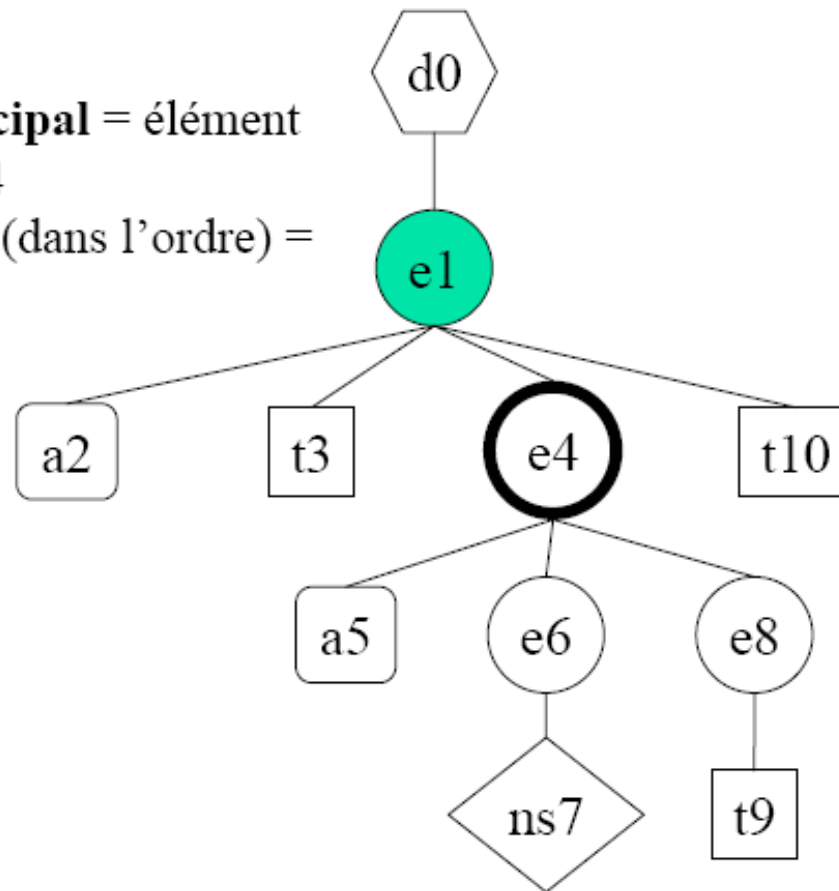
Axe : parent

Sens = arrière

Sorte de nœud principal = élément

Noeud contexte = e4

Nœuds sélectionnés (dans l'ordre) =
e1



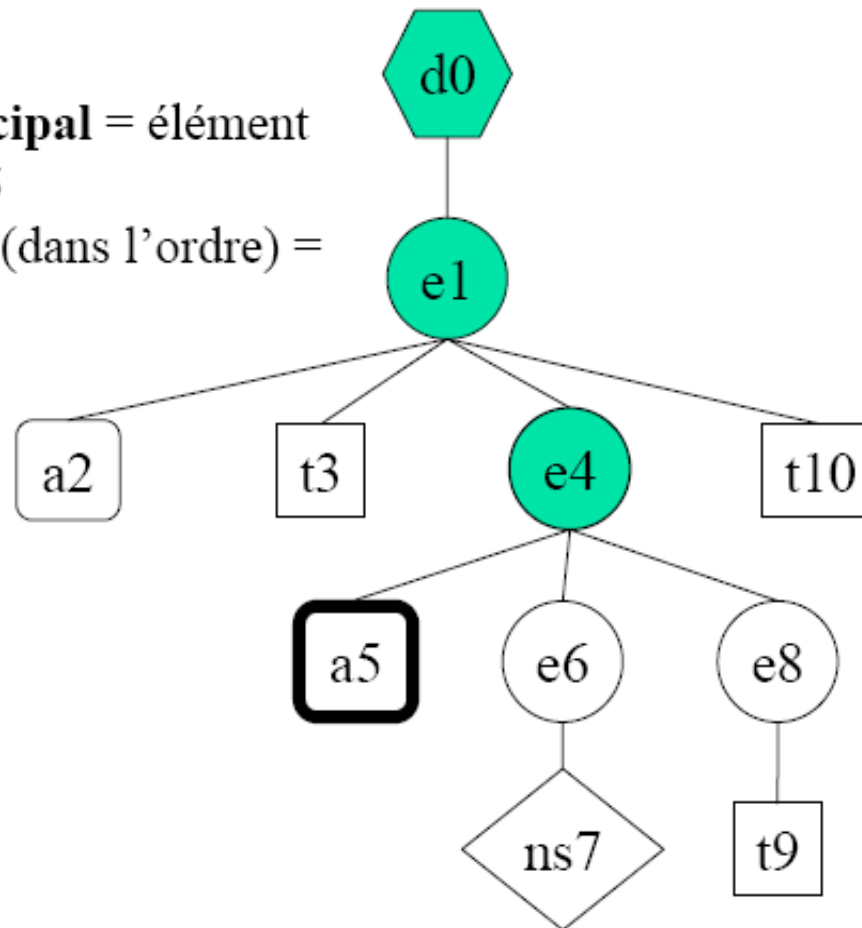
Axe : ancestor

Sens = arrière

Sorte de nœud principal = élément

Noeud contexte = a5

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



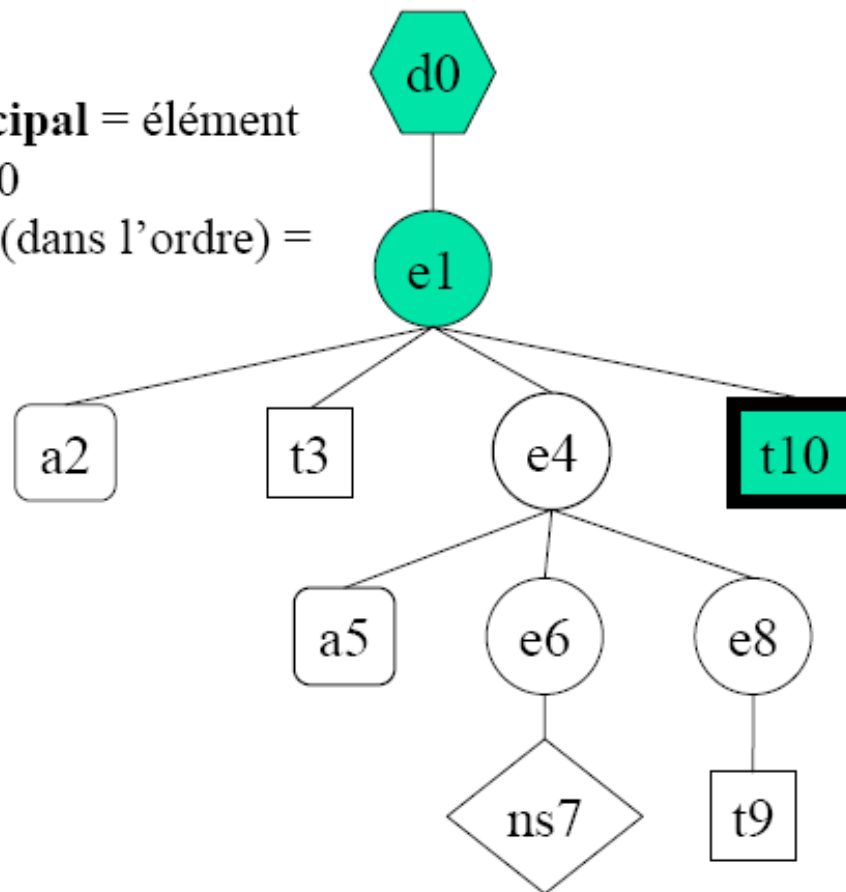
Axe : ancestor-or-self

Sens = arrière

Sorte de nœud principal = élément

Noeud contexte = t10

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



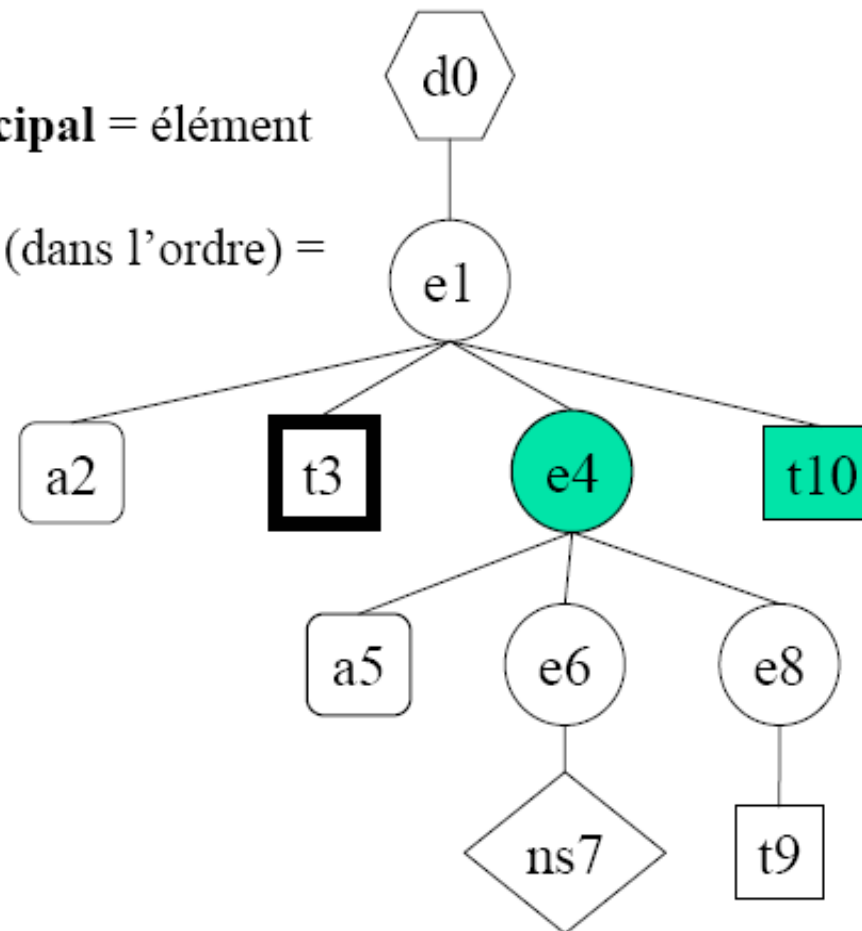
Axe : following-sibling

Sens = avant

Sorte de nœud principal = élément

Noeud contexte = t3

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



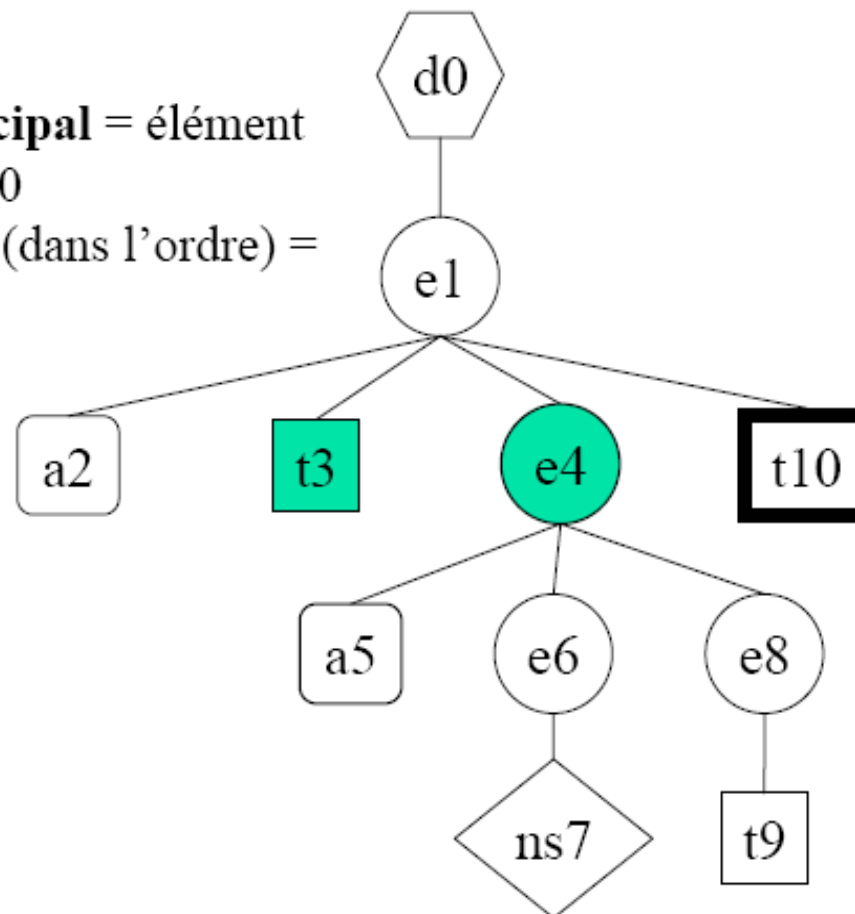
Axe : preceding-sibling

Sens = arrière

Sorte de nœud principal = élément

Noeud contexte = t10

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



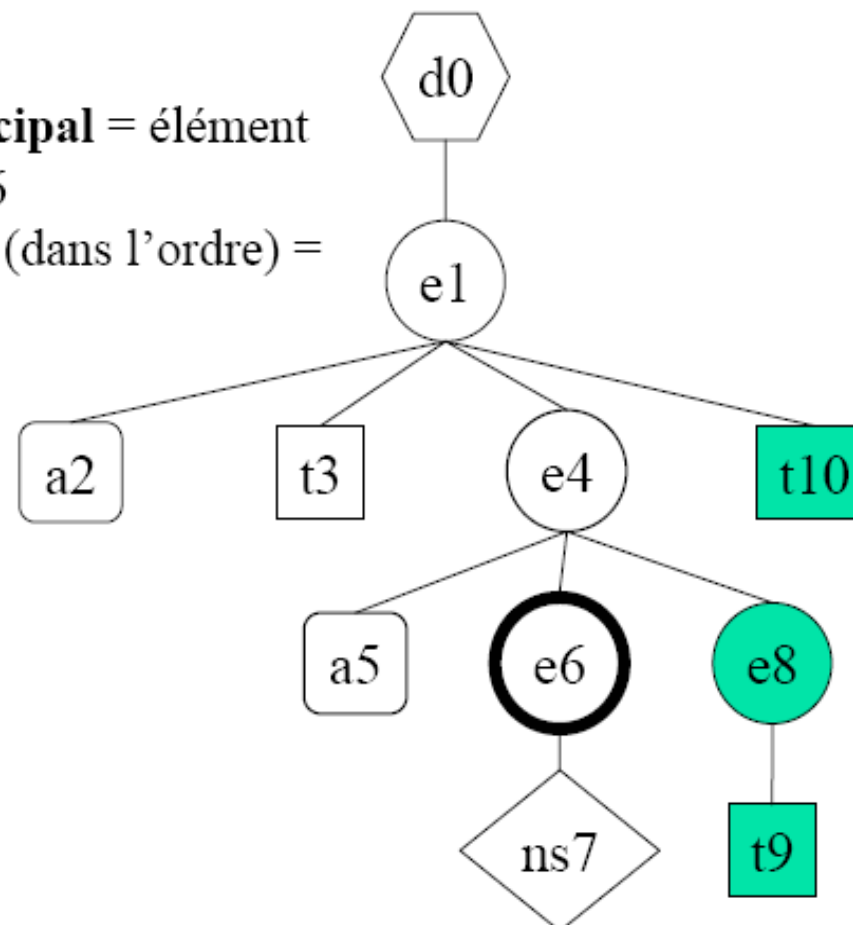
Axe : following

Sens = avant

Sorte de nœud principal = élément

Noeud contexte = e6

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



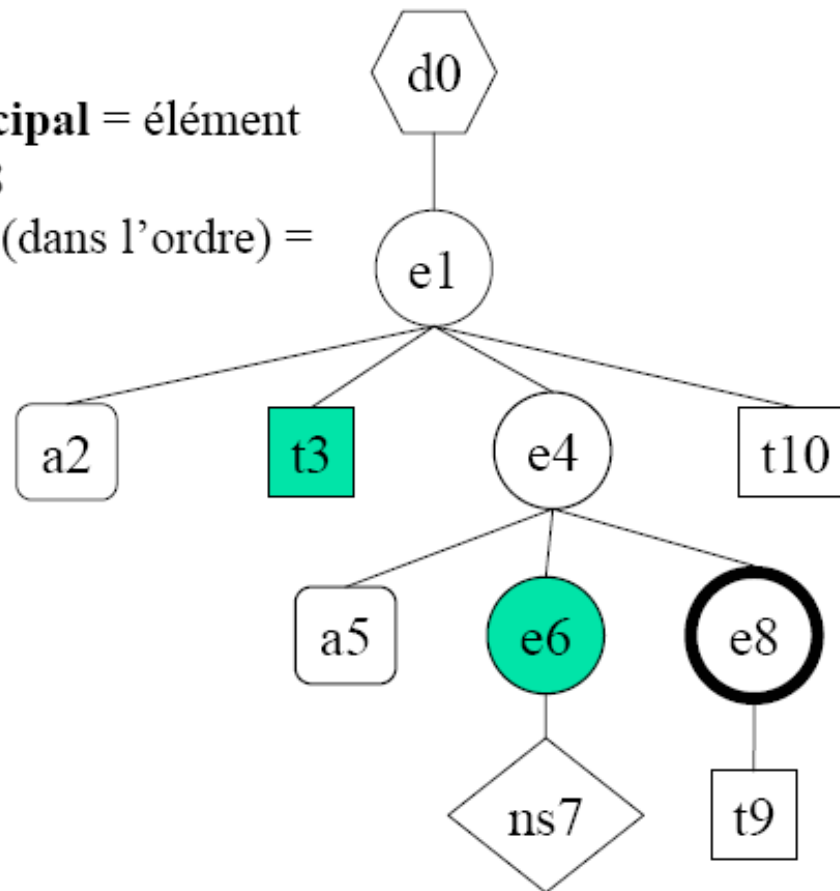
Axe : preceding

Sens = arrière

Sorte de nœud principal = élément

Nœud contexte = e8

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



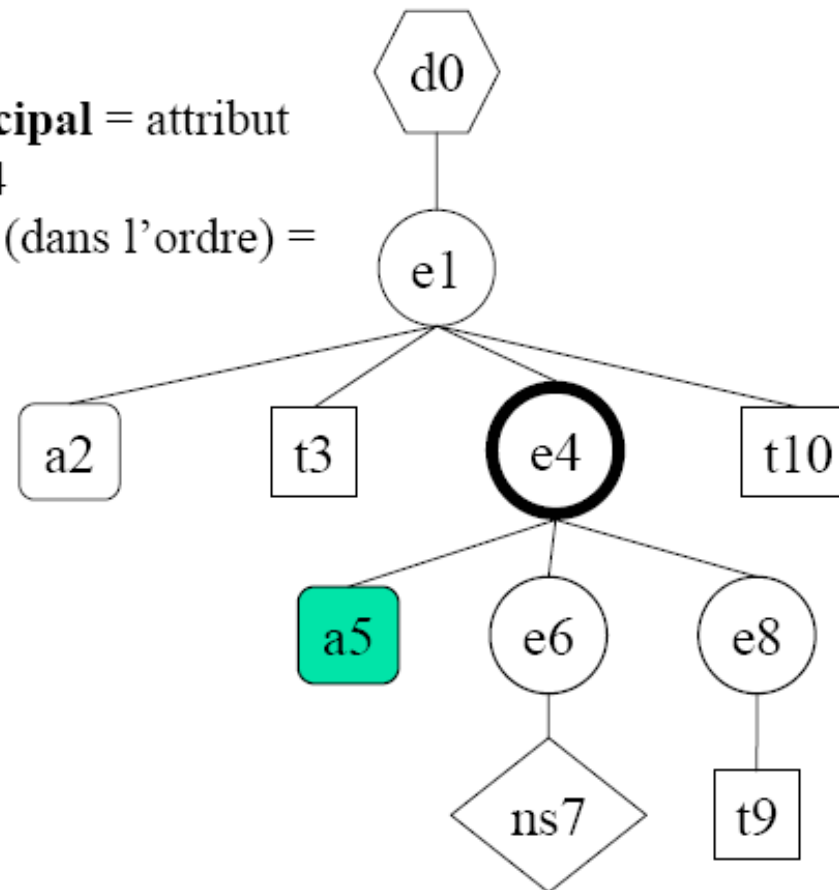
Axe : attribute

Sens = avant

Sorte de nœud principal = attribut

Noeud contexte = e4

Nœuds sélectionnés (dans l'ordre) =
a5



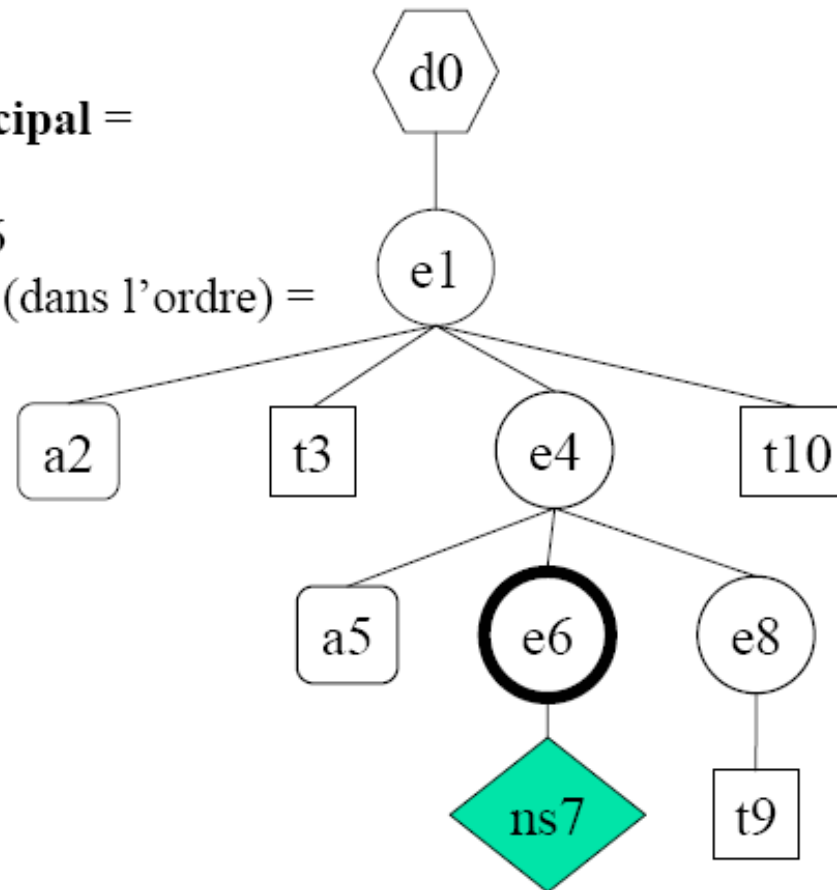
Axe : namespace

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 **prédicats**.
- **Syntaxe :**
 - *axe::test de noeud prédicat₁... prédicat_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 prédicats 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)

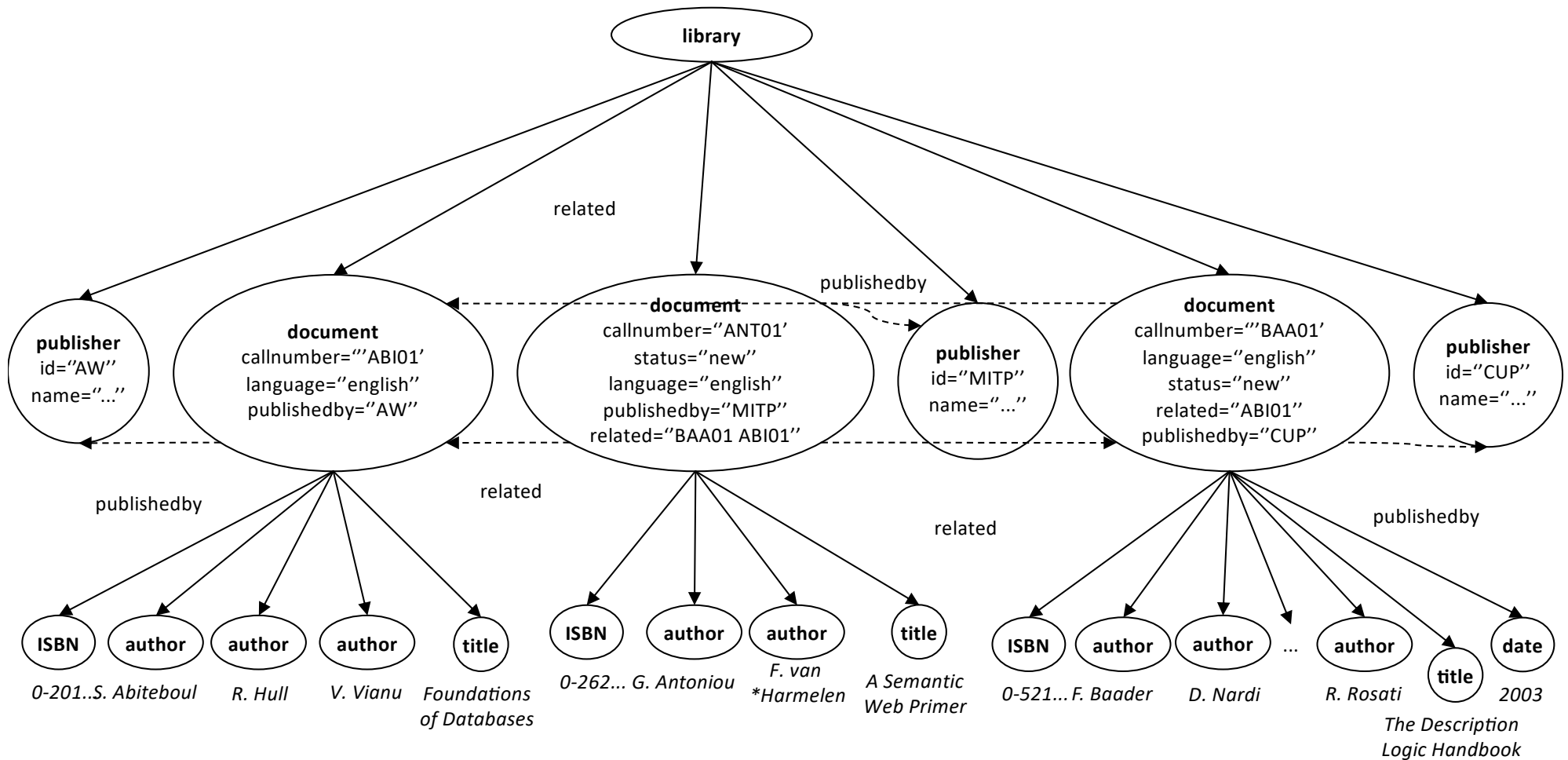
- Quel est le nom du dernier vallon du guide ?
`/guide/vallon[last()]/nom`
- Quels sont les textes de toutes les notes de type « prudence » ?
`//note[@type = "prudence"]/text()`
- Quels sont les noms des itinéraires cités dans la description de l'itinéraire de la Pointe de Névache ?
`//itinéraire[contains(text(), "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

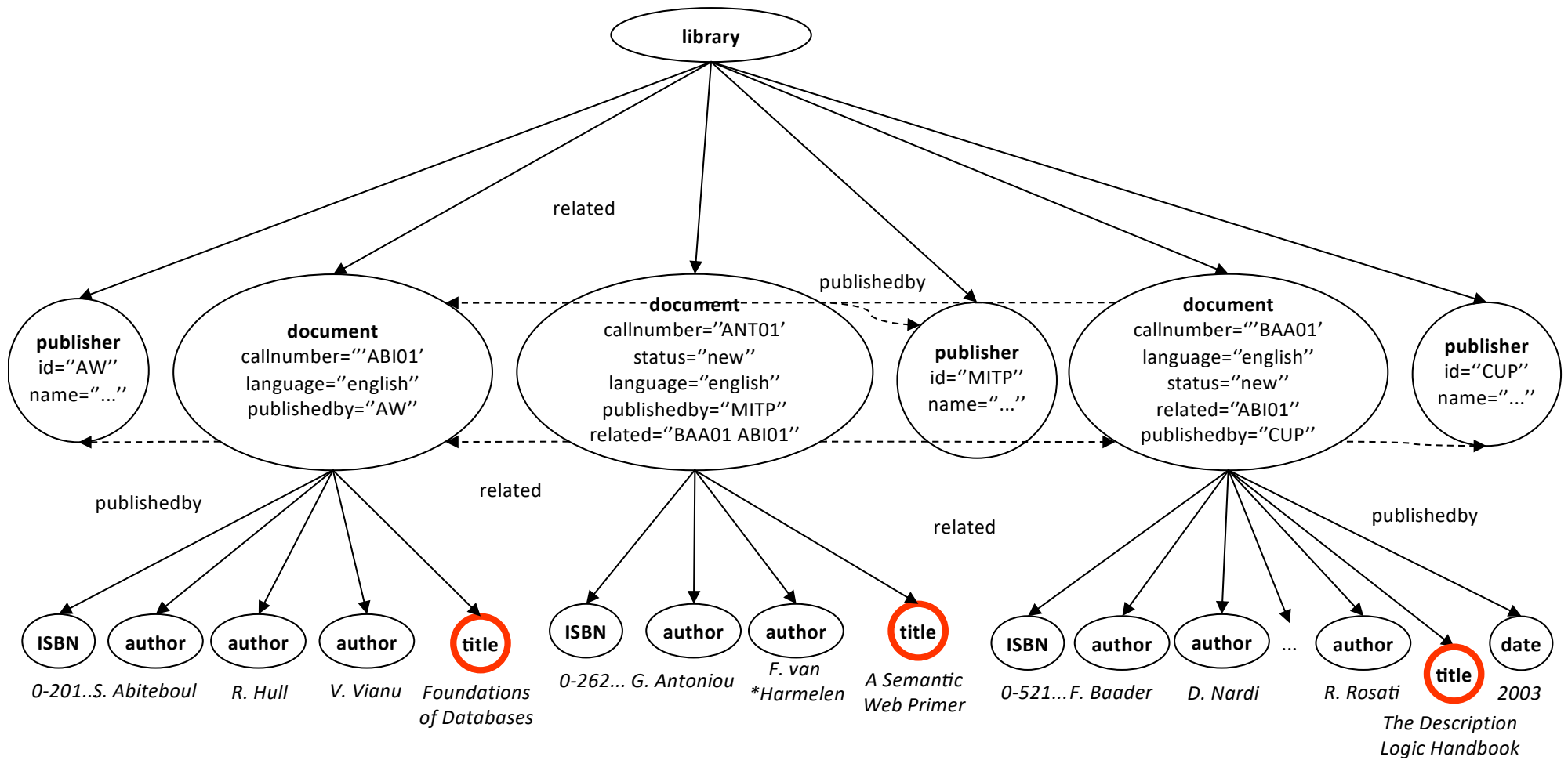
- **last** → nombre égal à la dimension contextuelle issue du contexte d'évaluation de l'expression
- **position** → nombre égal à la position contextuelle issue du contexte d'évaluation de l'expression
- **count** → nombre de nœuds de l'ensemble de nœuds passé en argument

EXEMPLES EN IMAGES

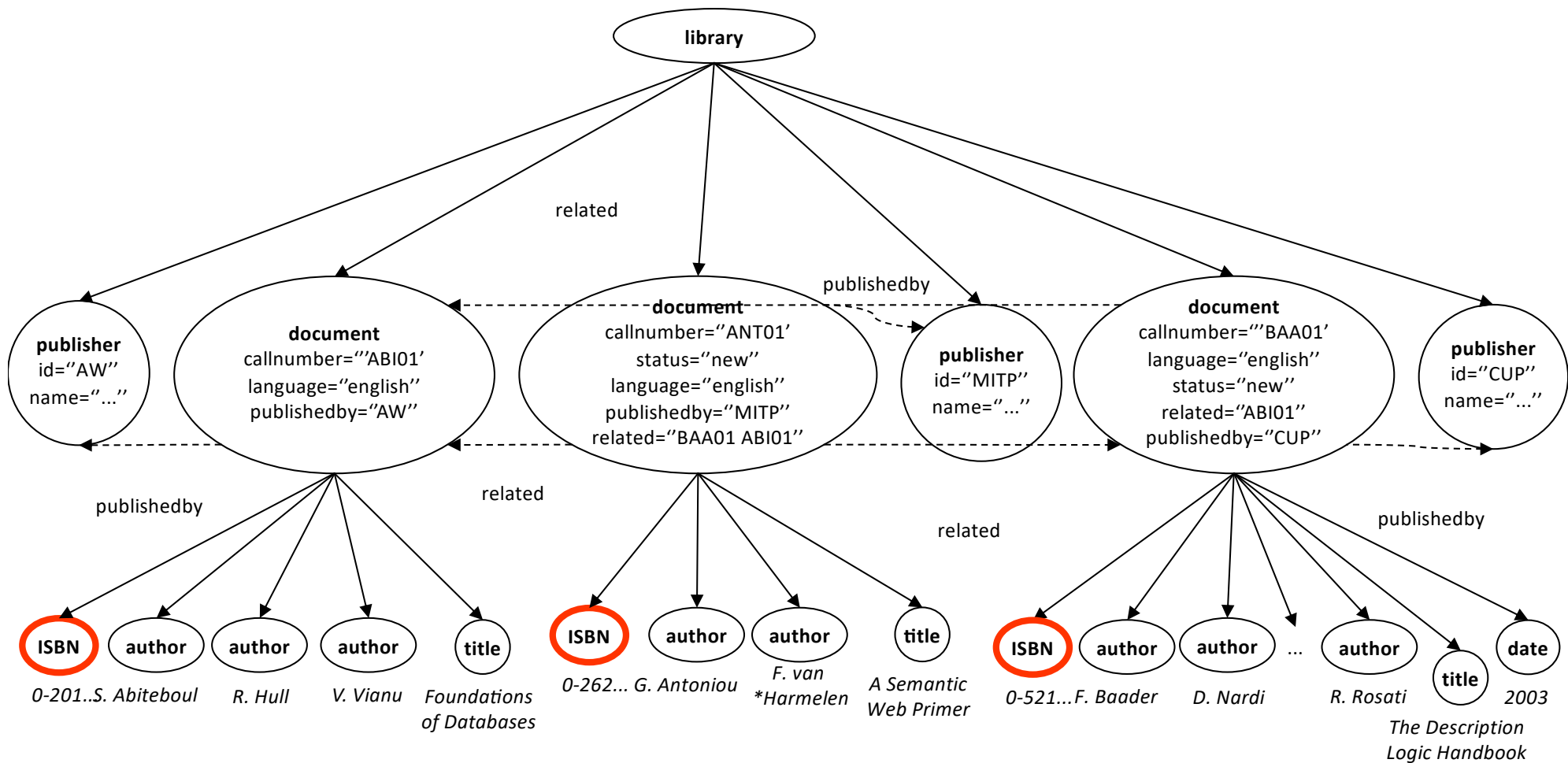
Arbre XML



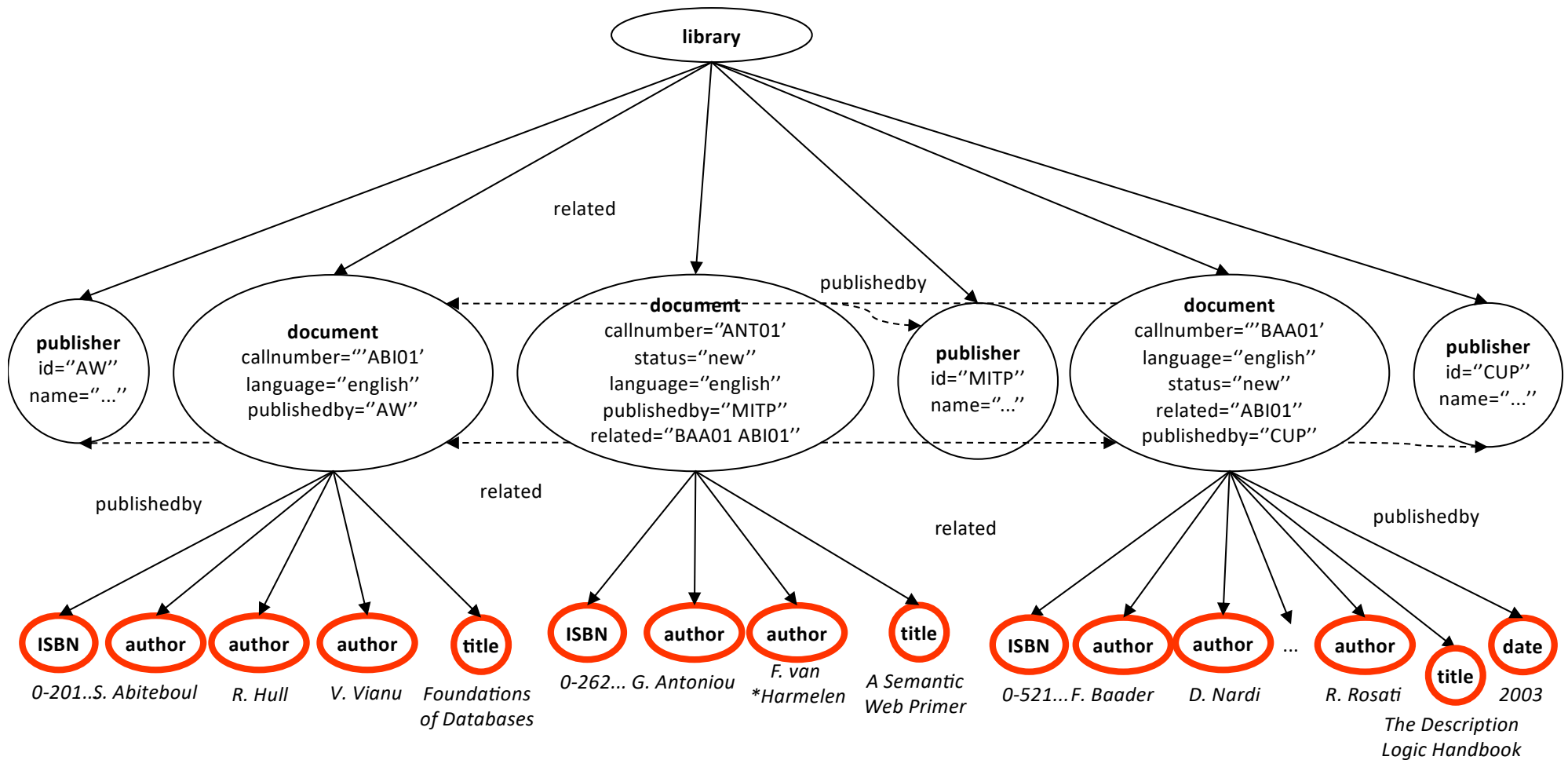
/library/document/title



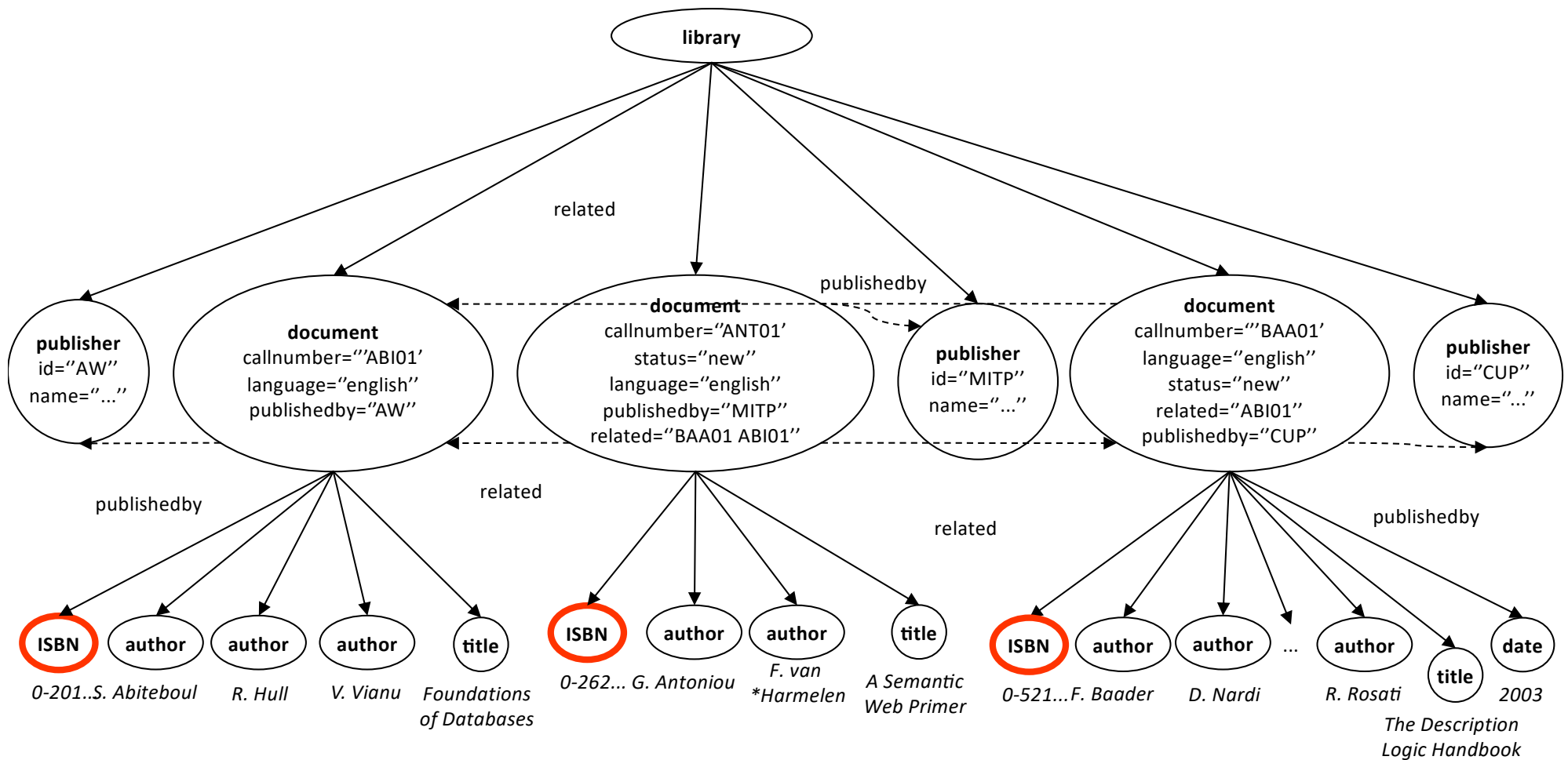
//ISBN



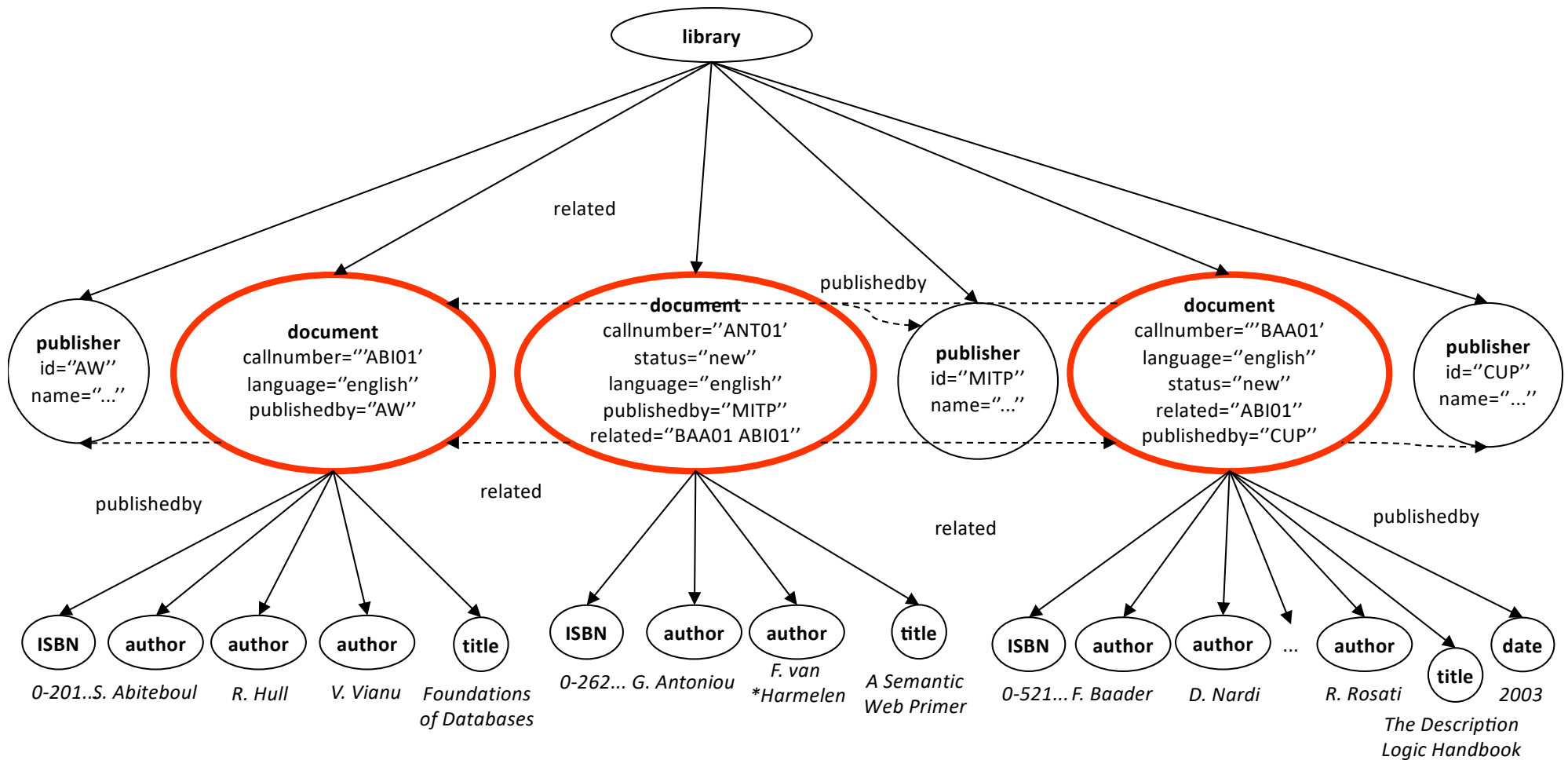
/library/document/*



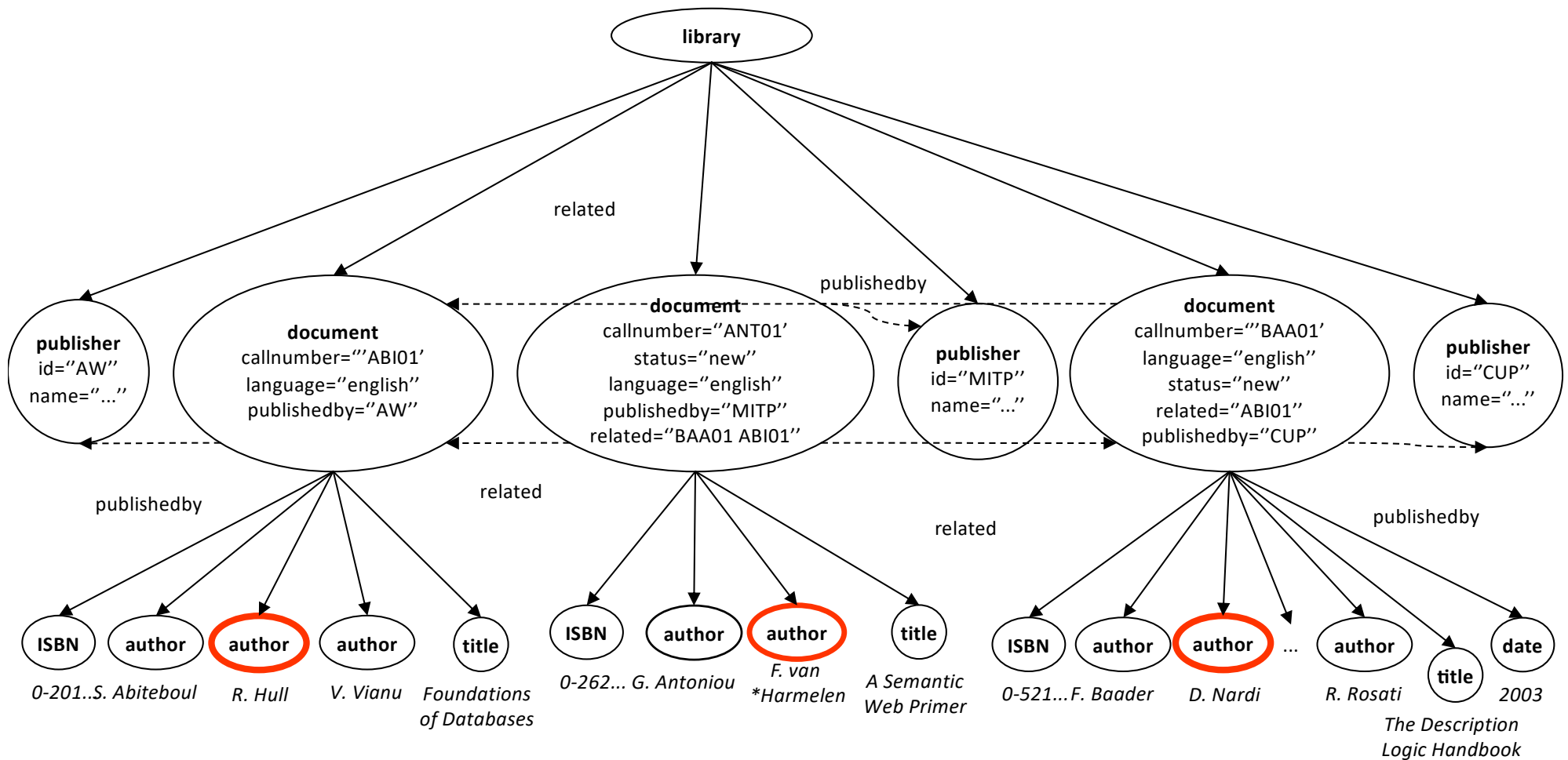
/*/*ISBN



/**/**/**/..

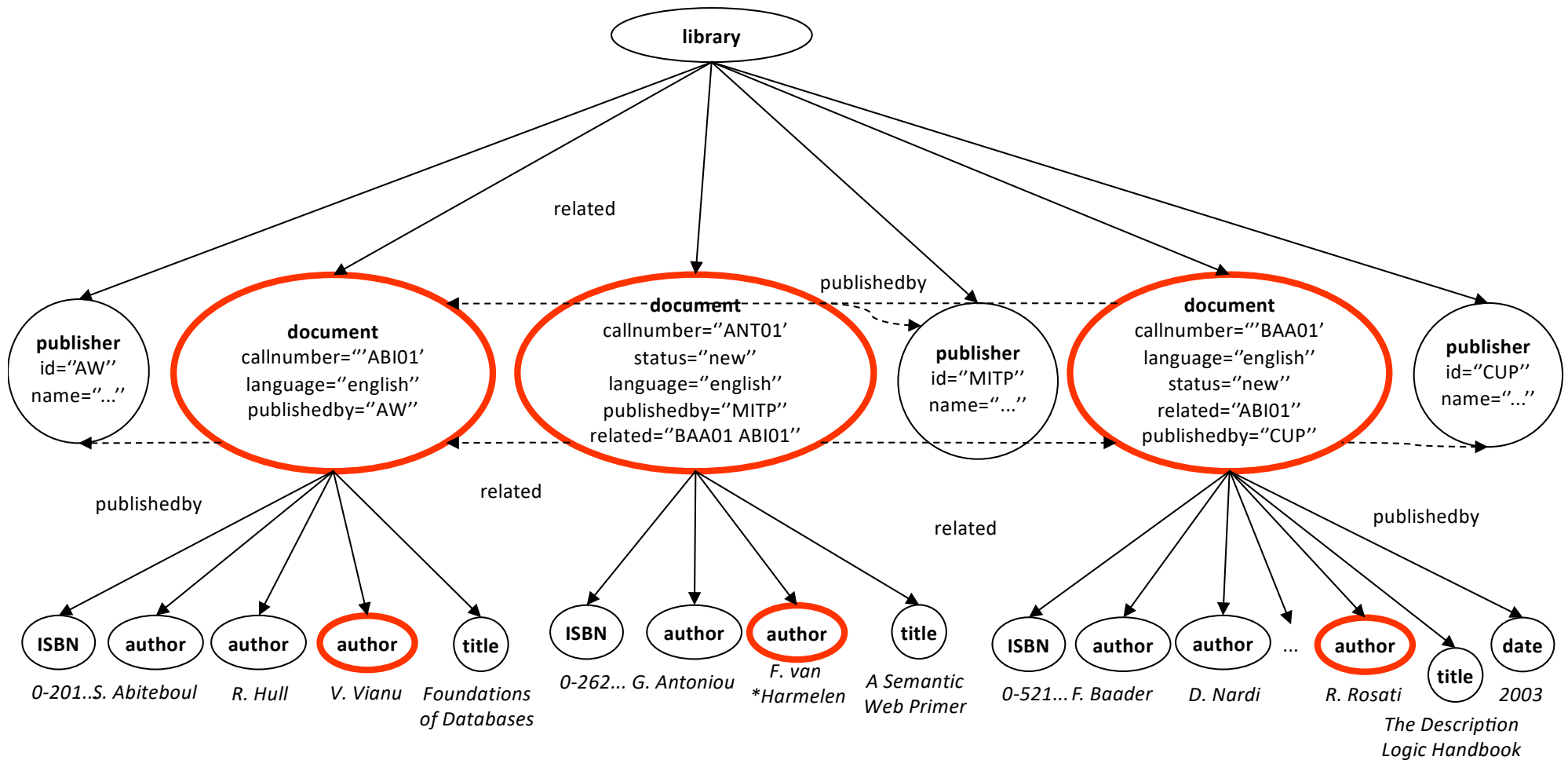


/library/document/author[2]

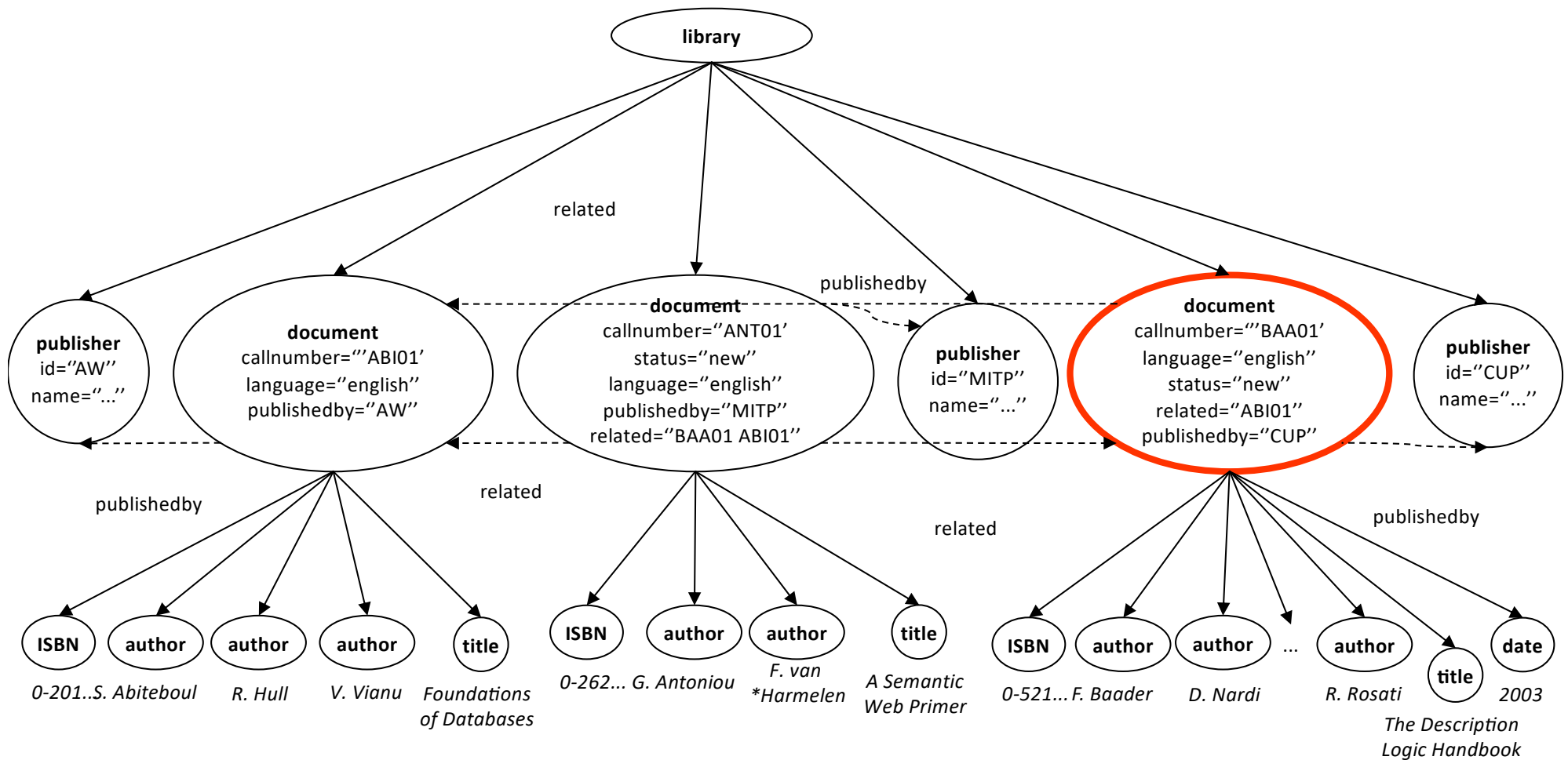


/**/**/.. |

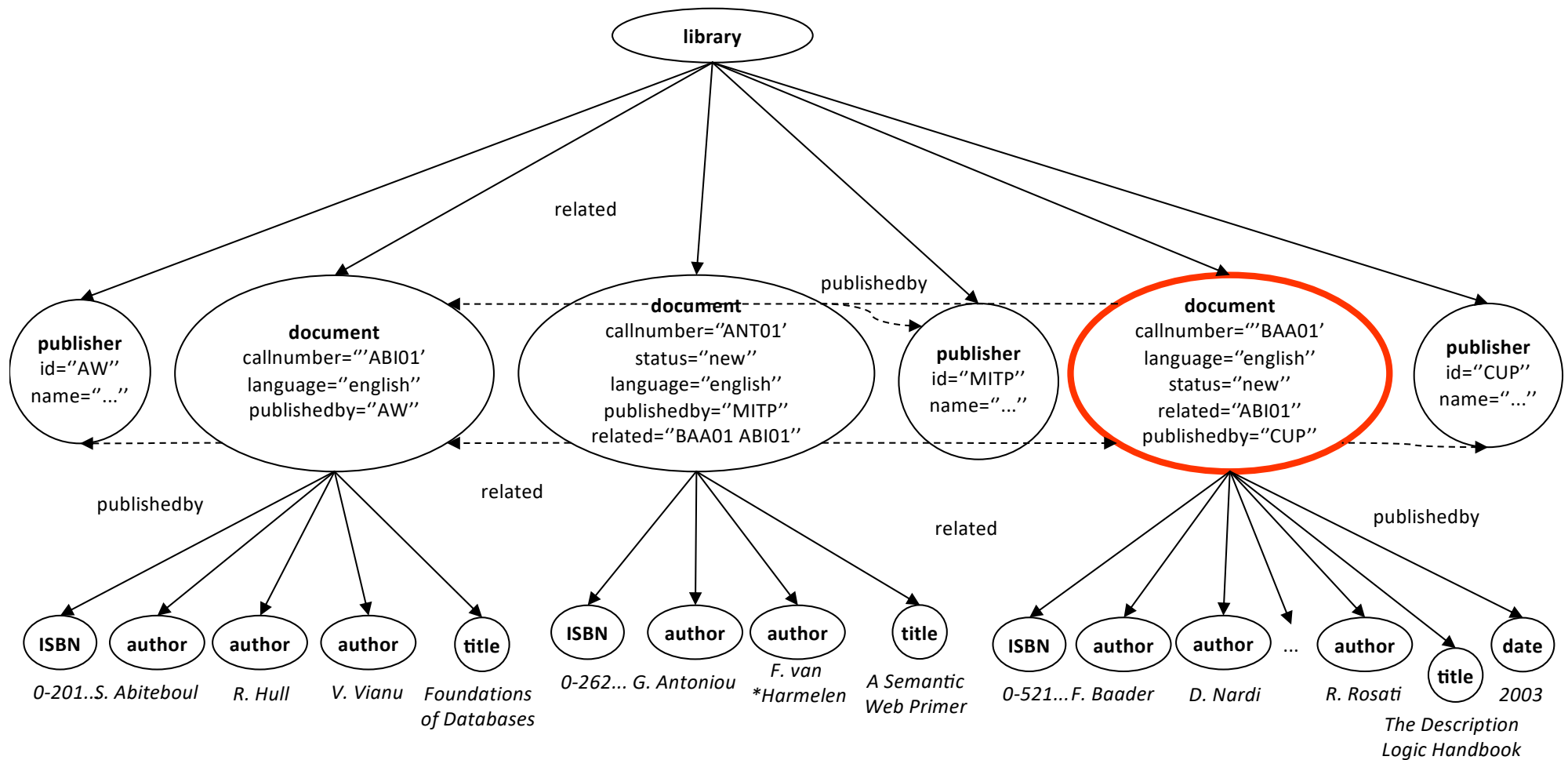
/library/document/author[last()]



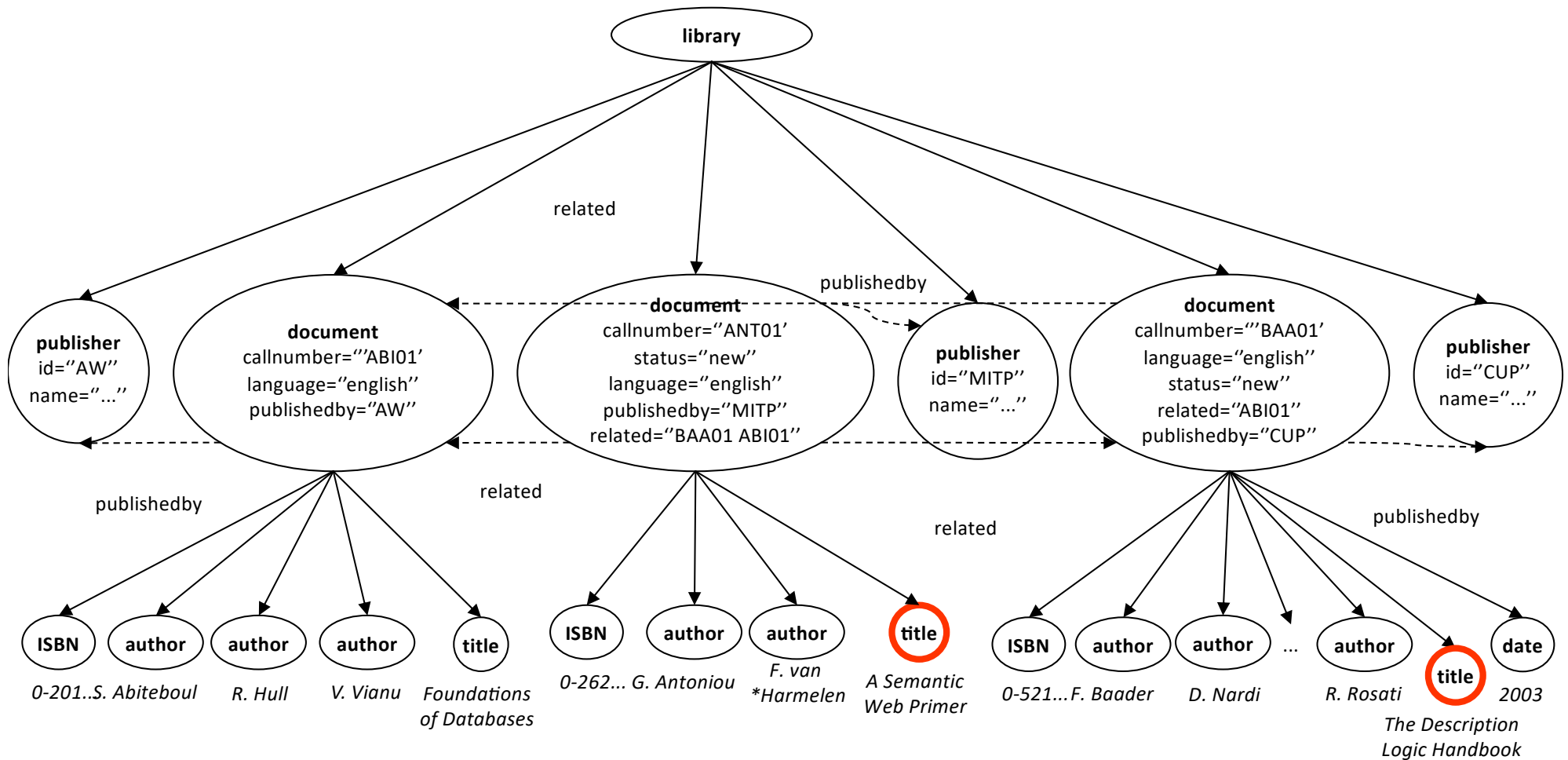
/library/document[date]



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



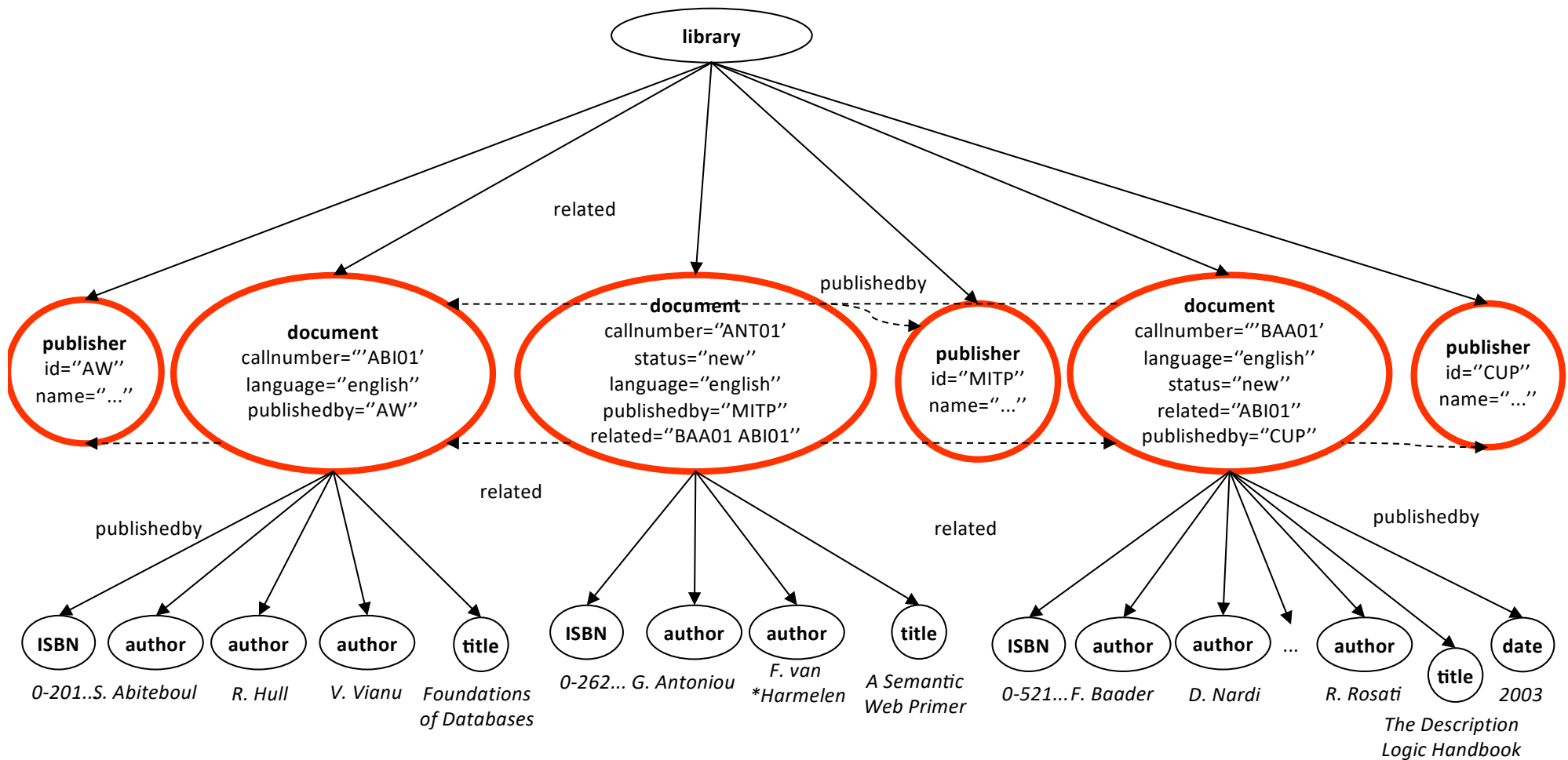
**/library/document[ISBN='0-262-01210-3' or
date]/title**



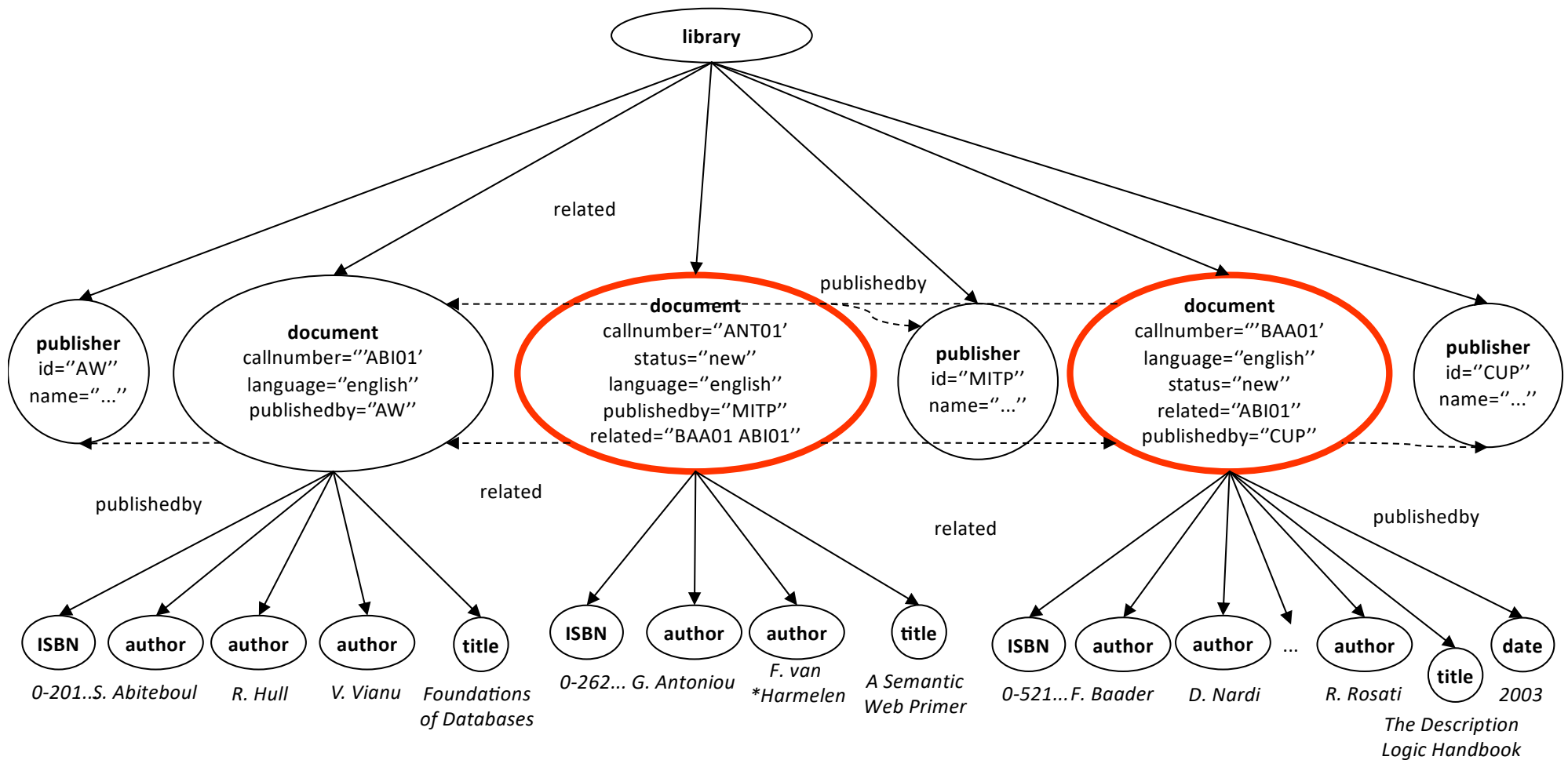
Interrogation XML : XPath

- Éléments qui possèdent des attributs
 - **`//*[@*]`**
- Éléments ayant un attribut donné **related**
 - **`//document[@related]`**
- Éléments ayant des attributs donnés **related et language**
 - **`//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']`**

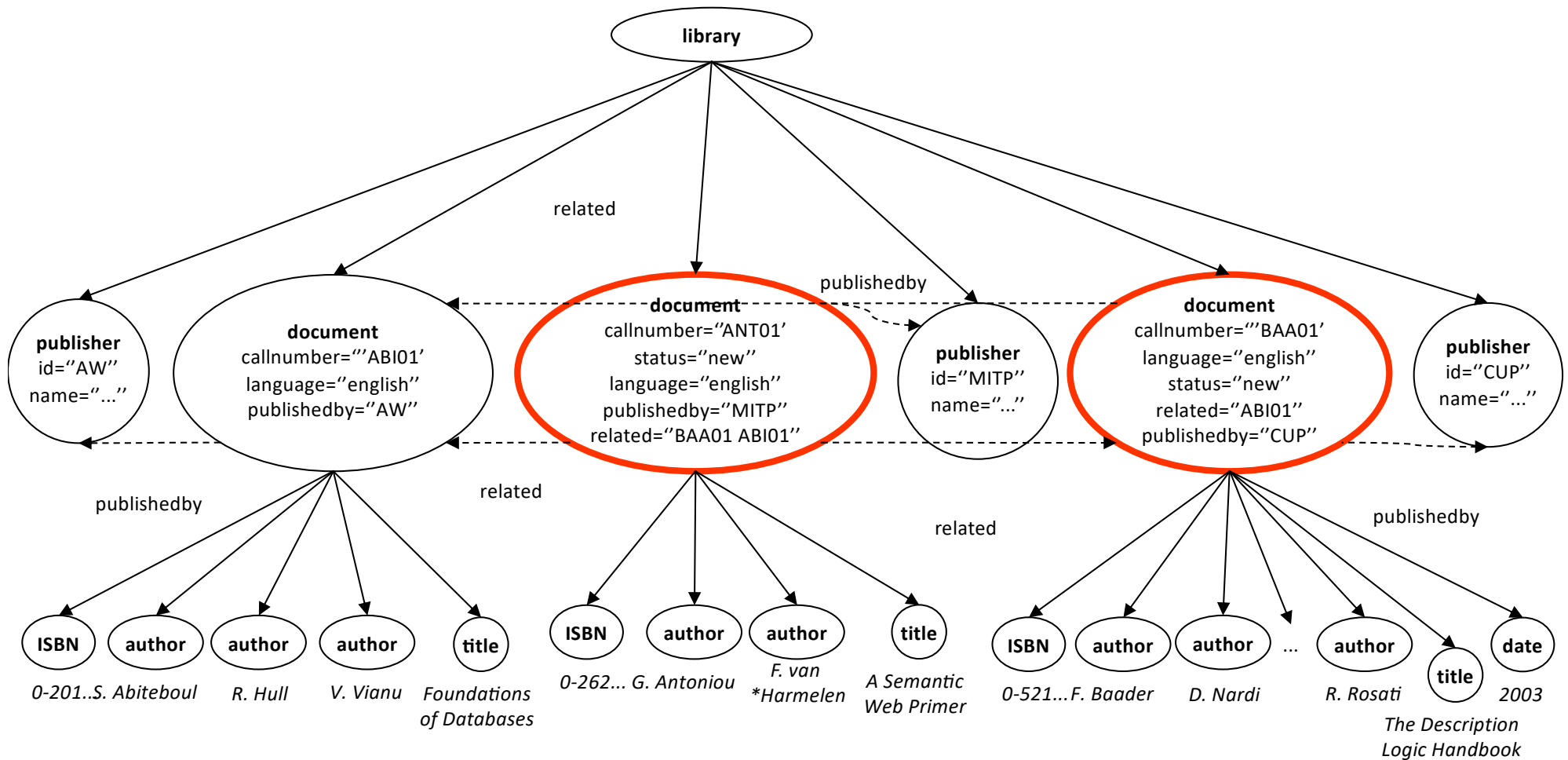
//*[@*]



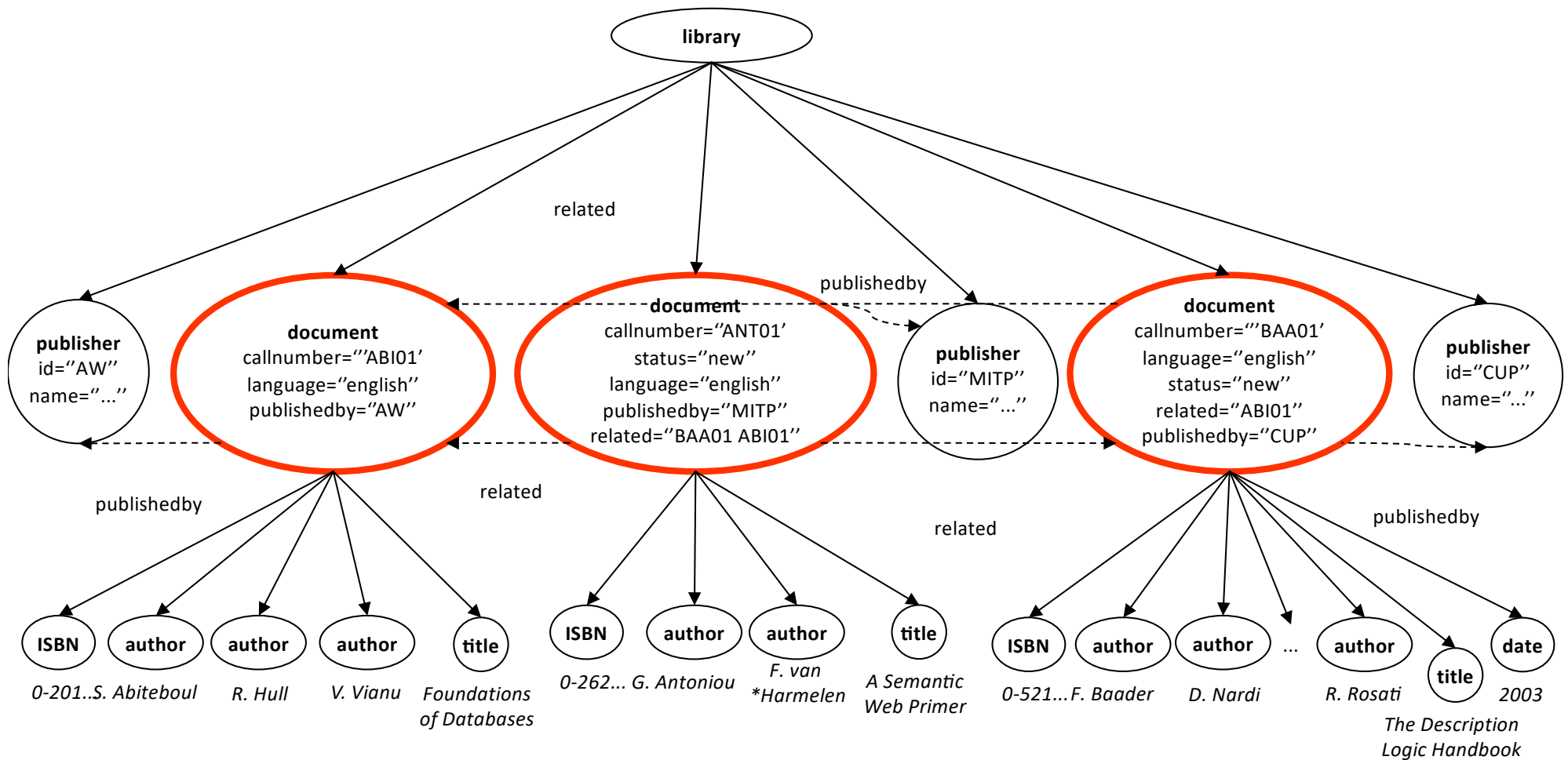
//document[@related]



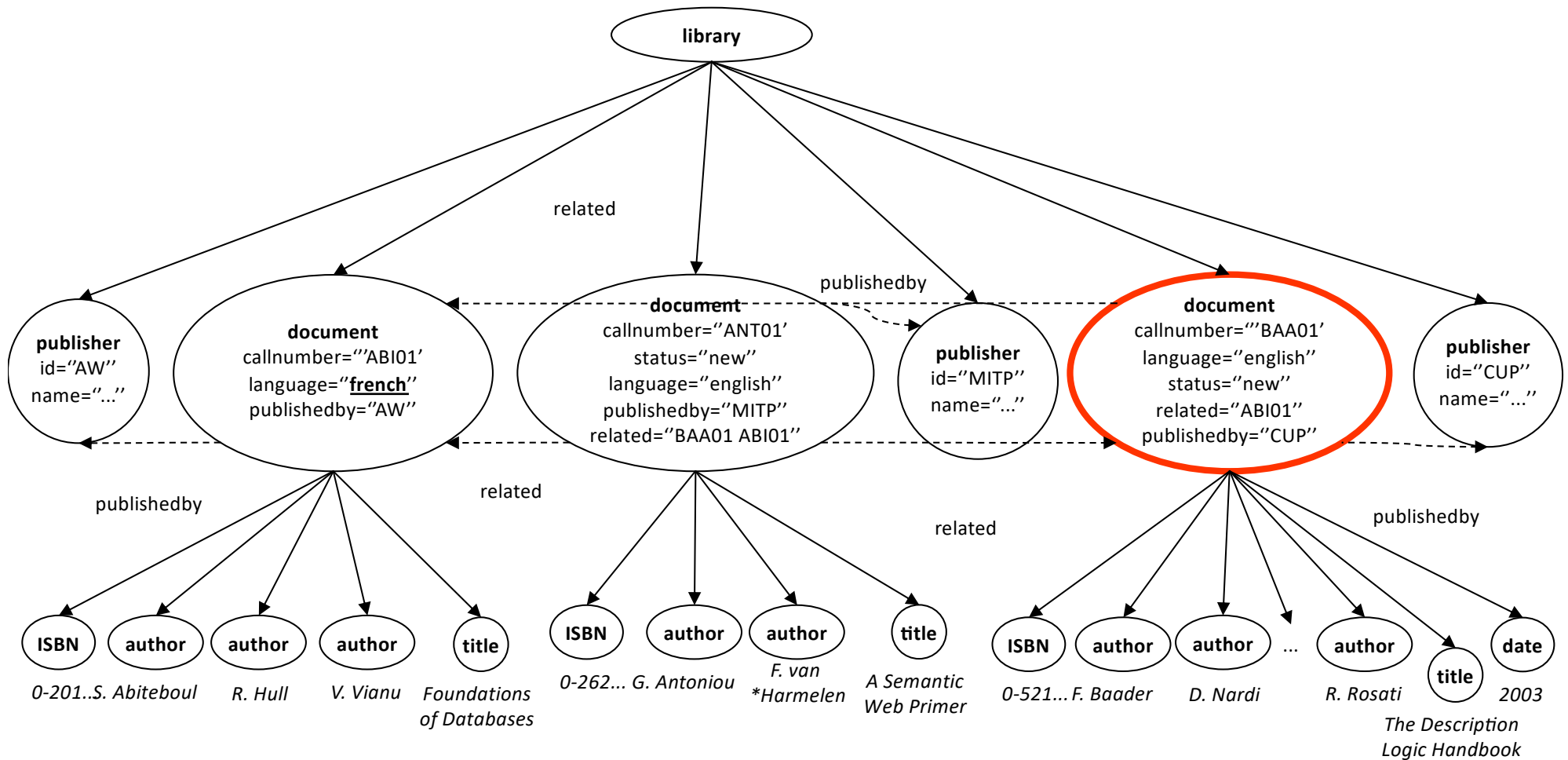
//document[@related and @language]



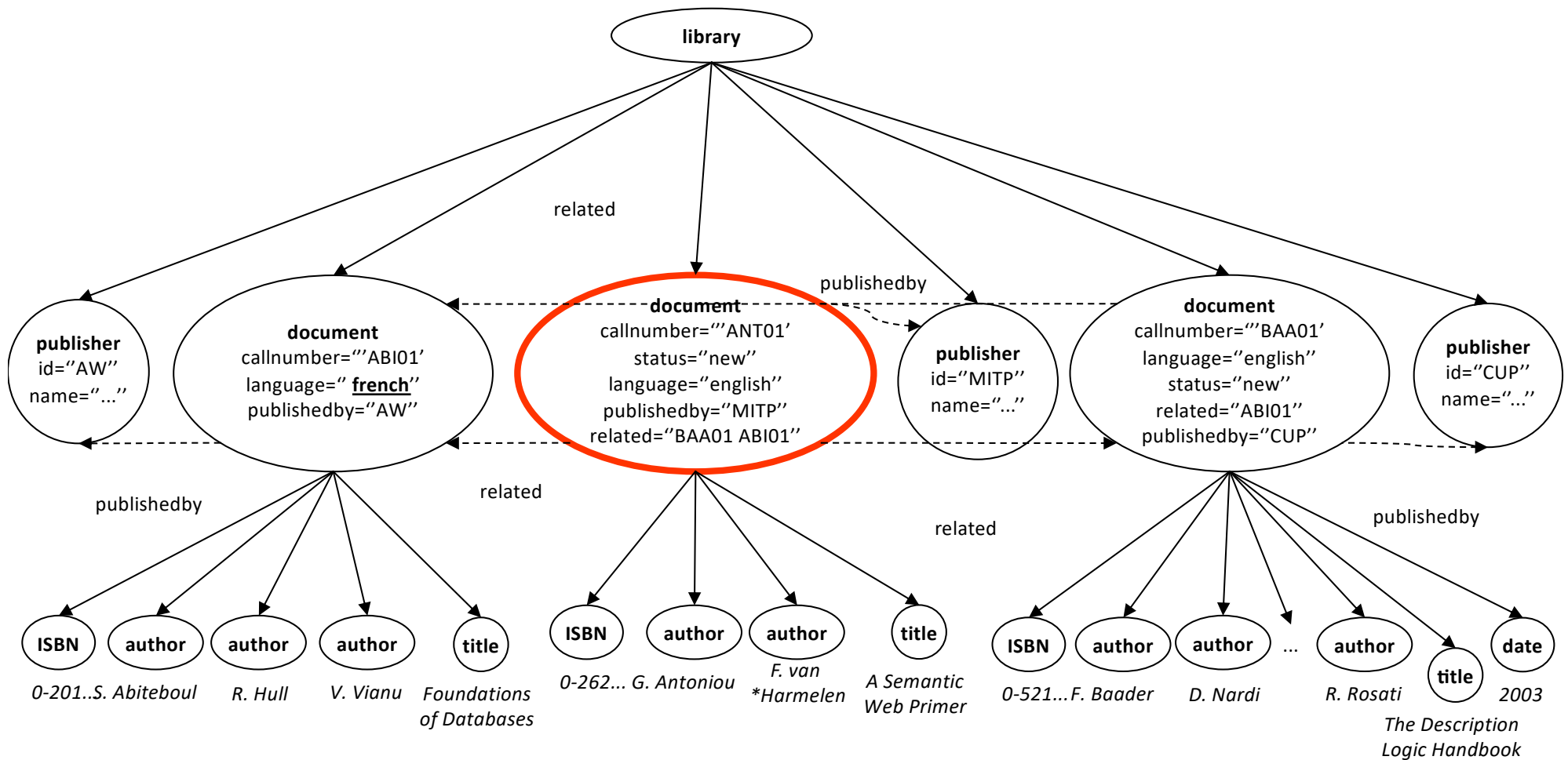
//document[@publishedby='AW' or
@status='new']



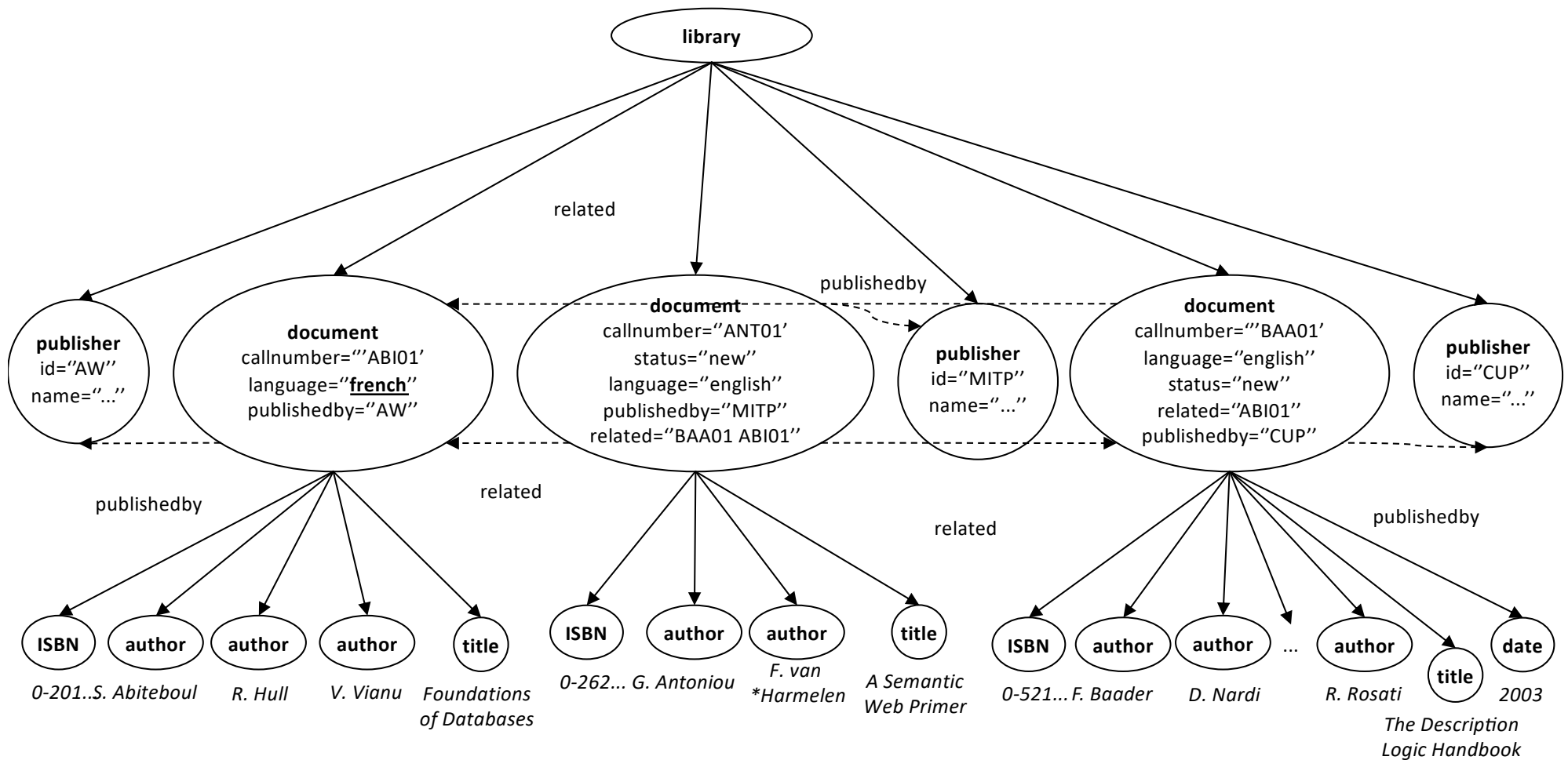
/library/document[@language='english'][2]



/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 (@nom 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 retrouverez l'ensemble des fonctions sur le site du W3C à l'adresse :
- <http://www.w3.org/TR/xpath#corelib>

END

?

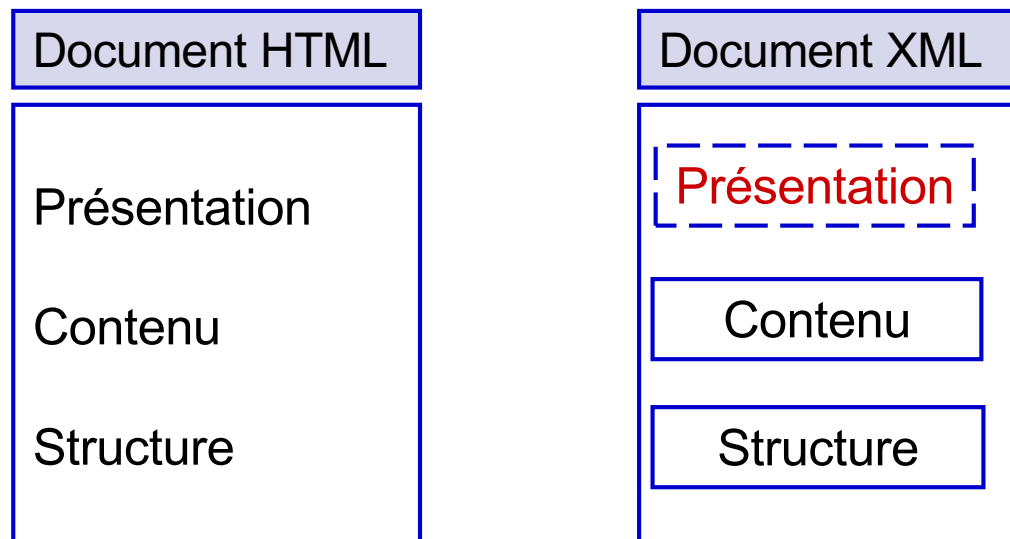




XML et ses applications : plan

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

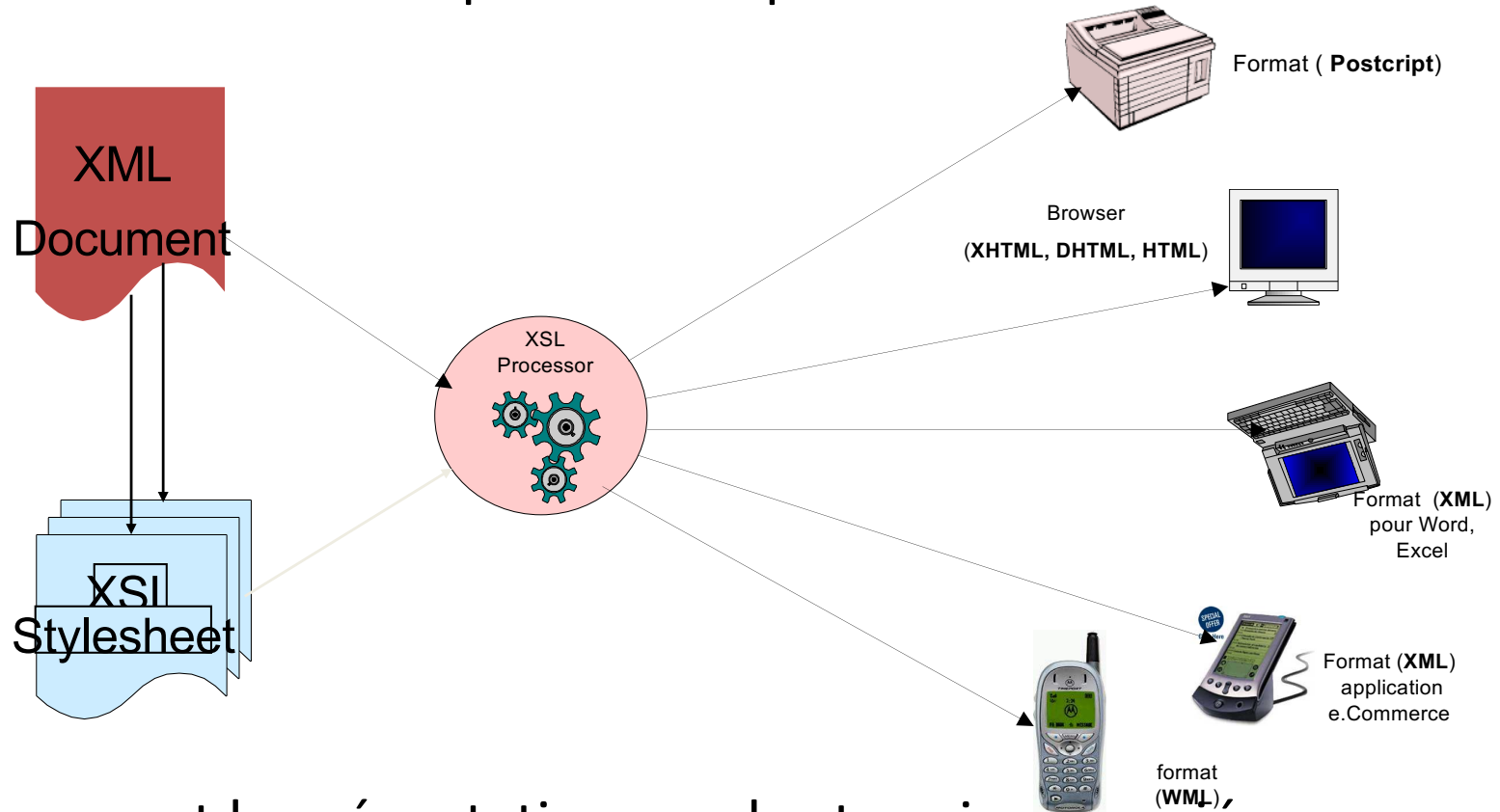
- XSLT permet de produire un nouveau document (XML, HTML, ...) par **transformation** d'un document XML existant.
- Une **transformation** s'exprime sous la forme d'une **feuille de style** qui est un document XML composé d'un ensemble de **règles** de transformation.
 - **Une règle comporte deux parties :**
 - 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**.

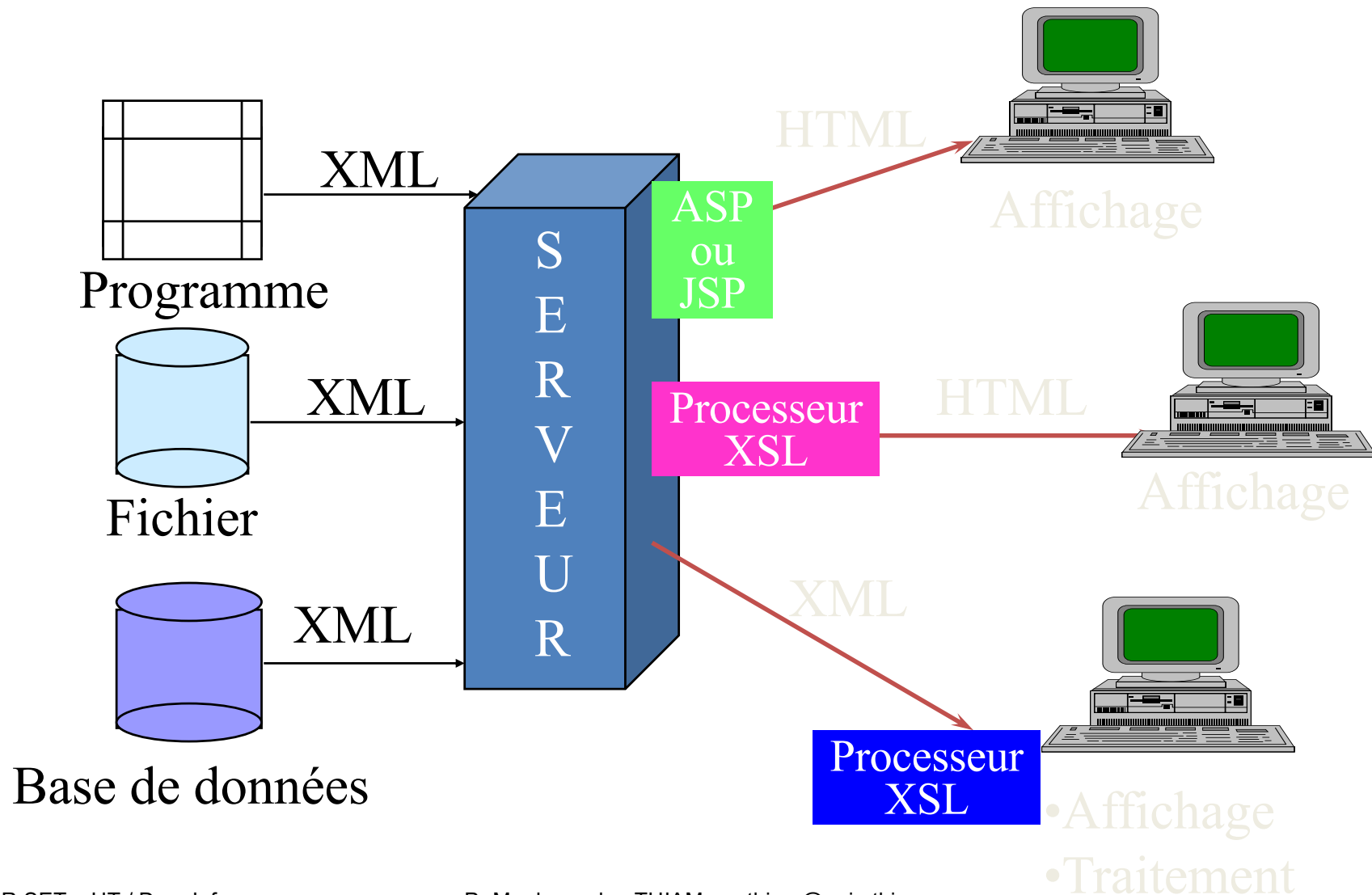
Publications avec XSL

- Plusieurs formats de publication pour un contenu

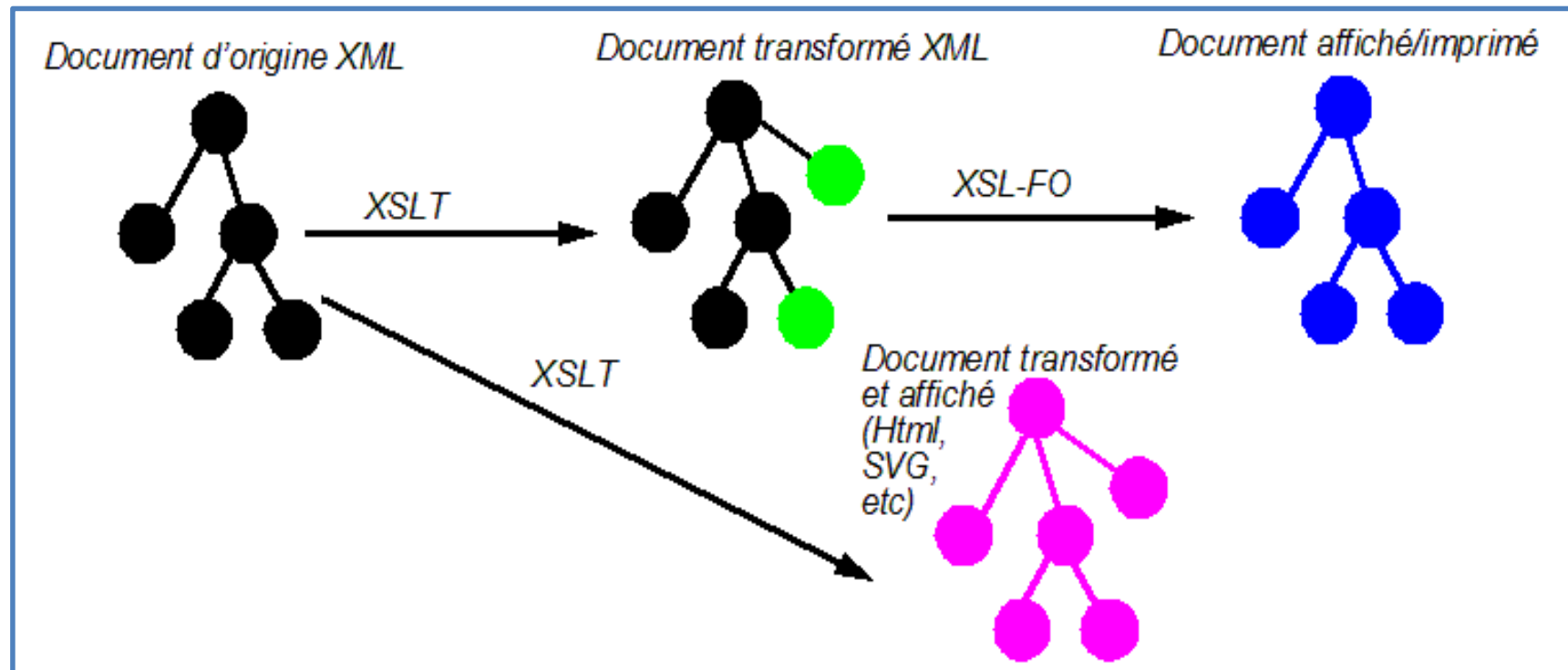


- XSL permet la présentation sur des terminaux variés

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 :

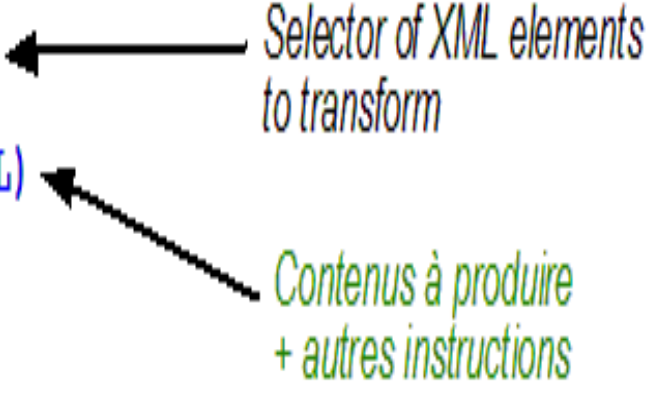
```
<xsl:template match="p">  
    Transformation  
</xsl:template>
```

- 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">
.... contents to produce (i.e. HTML)
.... further instructions
</xsl:template>
```



*Selector of XML elements
to transform*

*Contenus à produire
+ 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 **flot 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>`

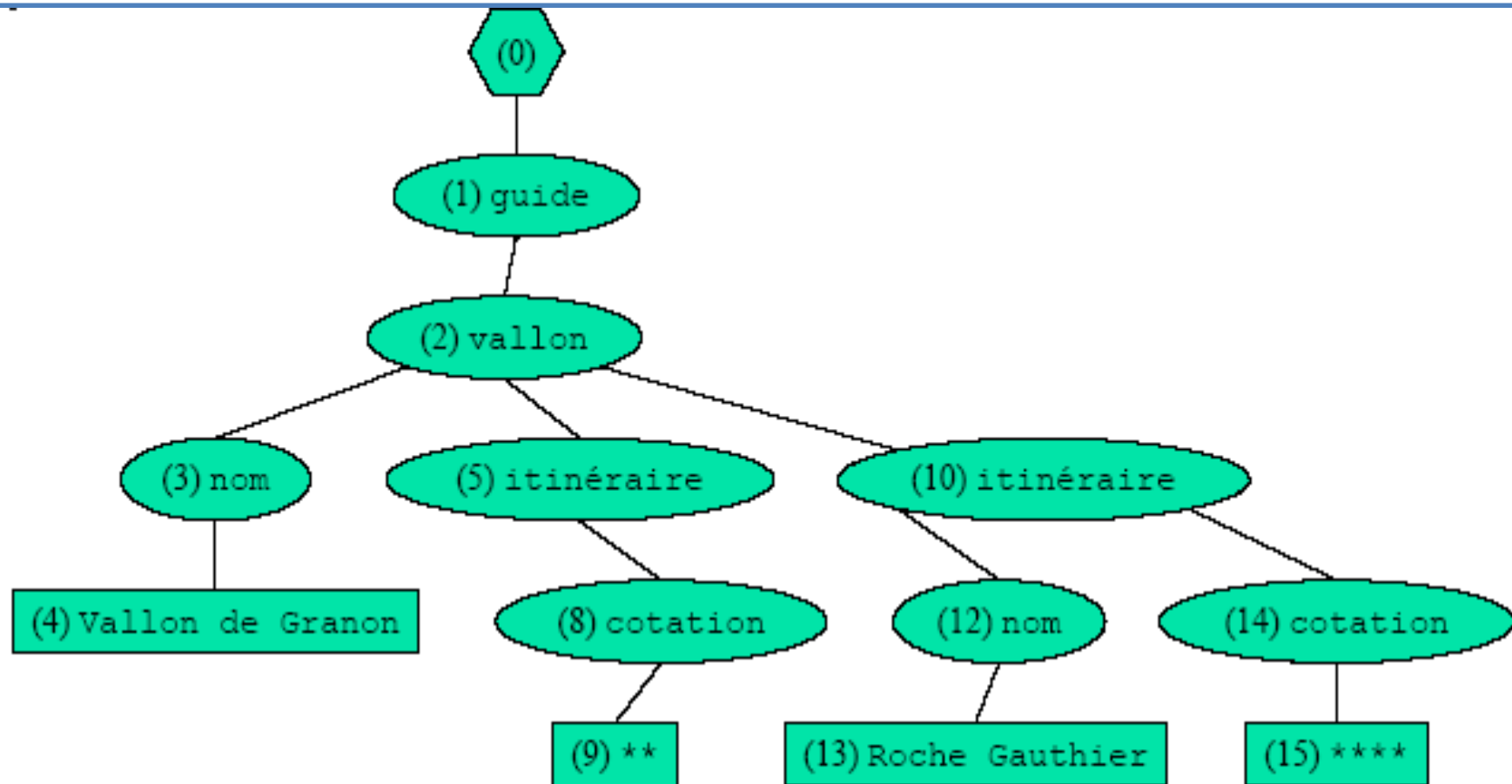
`<nom>Pic du Lauzin (Vallon des Acles)</nom>`

`<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> ...
```

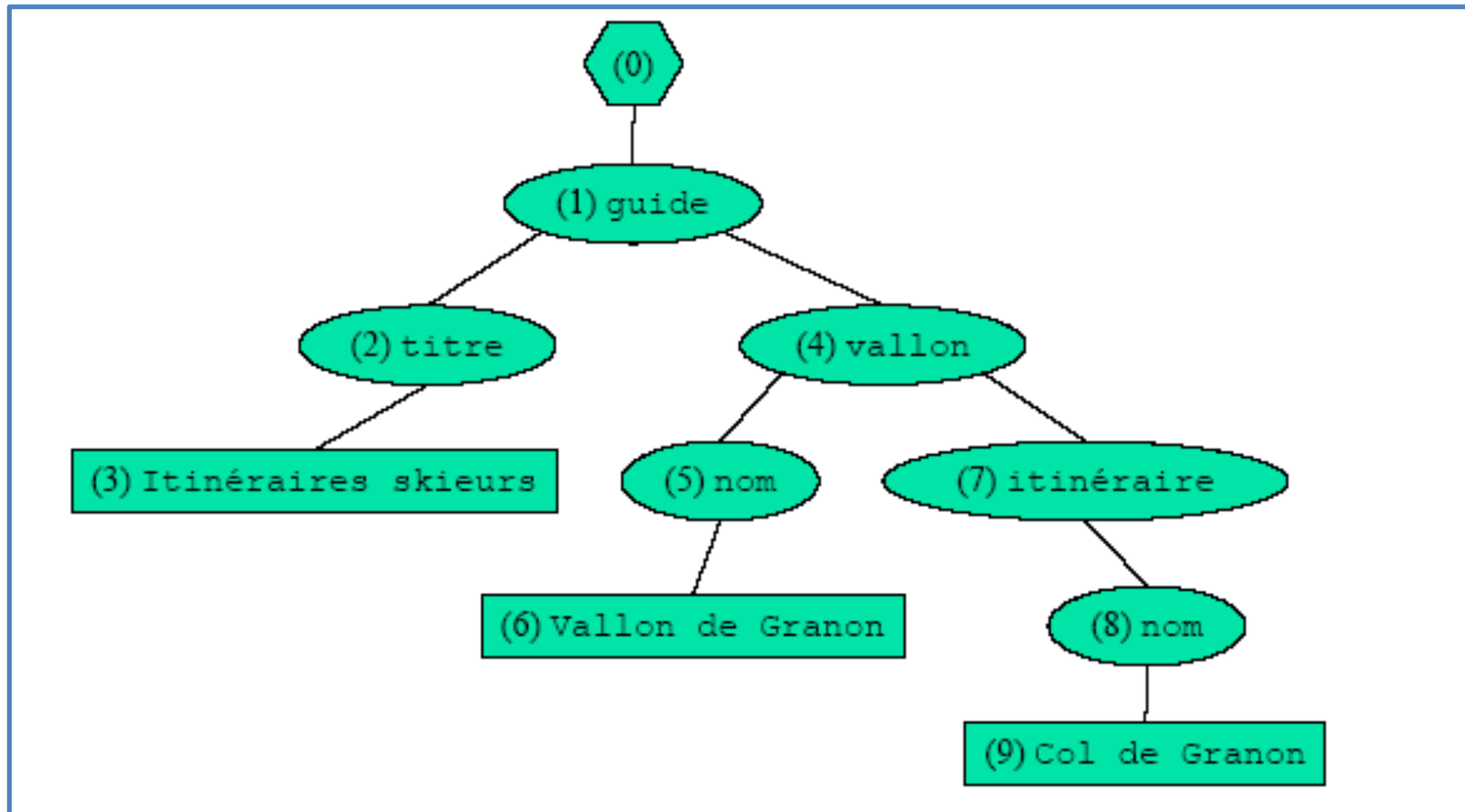
```
</vallon>
```

```
...
```

```
</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="itineraire/nom">
16.    <xsl:copy-of select="."/>
17. </xsl:template>
18. <xsl:template match="text()|@"/>
19. </xsl:stylesheet>
```

Cette règle cache la règle prédéfinie 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 prédéfinie 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 prédéfinie 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 prédéfinie 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)