INU3011 Documents structurés

Cours 8

XPath

Plan

- Logistique
- Traitement de documents XML : XPath

Logistique

- Conférencière confirmée : Edith Cannet
  - Éditrice (CNRS - Institut des sciences humaines et sociales)
    - ingénierie éditoriale
    - humanités numériques
  - Pôle Document numérique – Maison de la Recherche en Sciences Humaines – Université de Caen (France)
- Conférence au cours 12, le 3 avril 2023

Traitement de documents XML

XPath

Documents exemples

- Trois documents dans 200-Ex-XPath :

  doc-jouet.xml

  doc-simple.xml (version "aérée")

    doc-simple-dense.xml (version "compacte")

  doc-realiste.xml

XPath, c’est quoi?

- « XPath is a language for addressing parts of an XML document […]. »
- C'est en fait un langage *d'extraction d'informations* à partir de documents XML
  - Principalement des *morceaux de document*
  - Aussi certaines informations *sur le document*, qui ne sont pas explicitement dans le document (ex.: nombre d'éléments)

Pourquoi apprendre XPath?

- C'est un langage normalisé
- Il est répandu (oXygen, IE, BD XML, etc.)
- Il est intégré comme sous-langage dans XSLT et XQuery
  - Notamment, toute expression XPath est aussi une requête XQuery valide !

Note: nous ne voyons ici qu'une partie de XPath 1.0, correspondant à peu près à la "syntaxe abrégée" présentée dans la spécification

Outils pour explorer XPath

- 1er choix: oXygen*
- 2e choix: oXygen*
- 3e choix: oXygen*

* Supporte aussi XPath 2.0, 3.0 et 3.1 (non utilisés dans ce cours)

Ressources externes sur XPath (et XSLT)

- Rappels de la bibliographie du cours :
  - Livre de Michael Kay sur XSLT 1.0 en réserve
    - Celui sur XSLT 2.0 est disponible en ligne pour umontreal.ca
  - Tutoriel XPath de W3Schools
  - Tutoriel XSLT de W3Schools

XPath : statut normatif

- XML Path Language (XPath) 1.0
  - W3C Recommendation 16 novembre 1999
- XML Path Language (XPath) 3.0
  - W3C Recommendation 8 avril 2014
- XML Path Language (XPath) 3.1
  - W3C Recommendation 21 mars 2017

Modèle de données XPath (1/3)

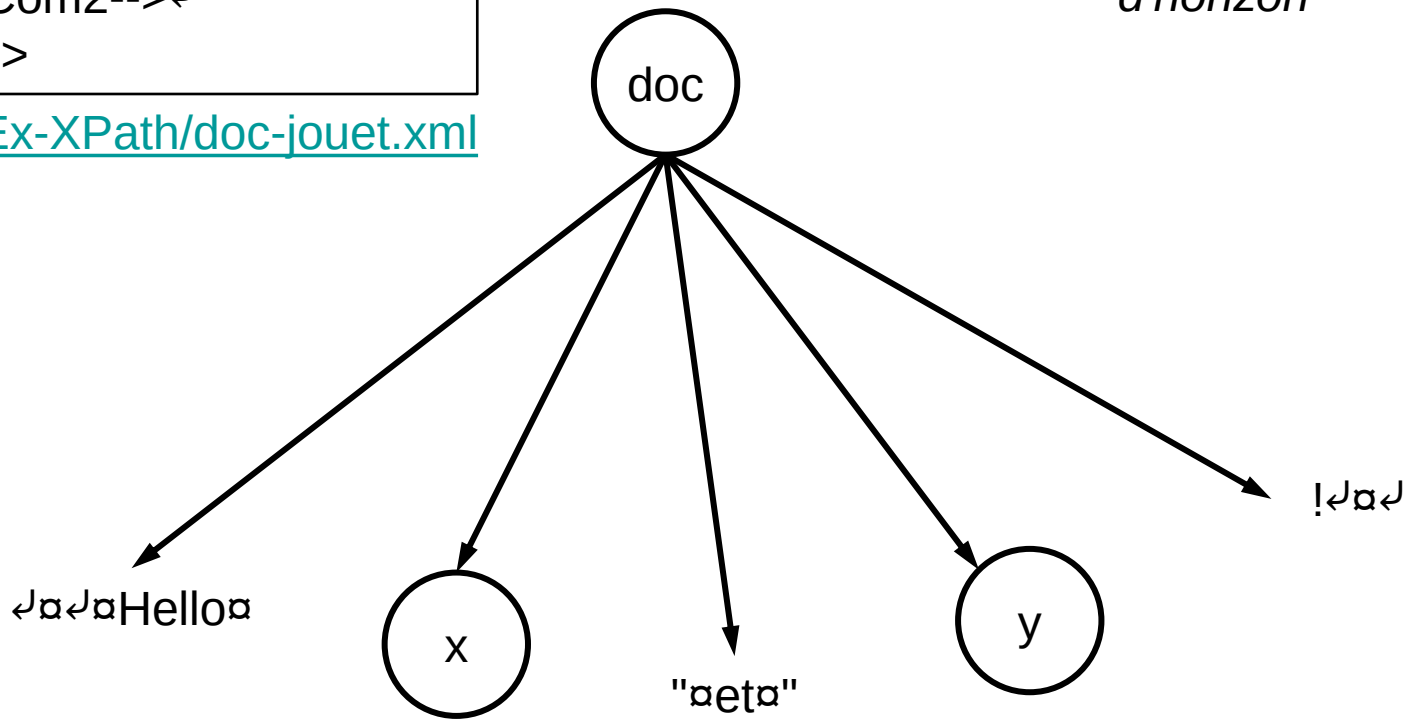
- Essentiellement :
  - Document XML = arbre inversé **non allégé** comme dans le *Premier tour d'horizon*
- Mais avec quelques ajouts:
  - Ajout d'un nœud "racine" au-*dessus* de l'élément de plus haut niveau
  - Attributs et commentaires sont des nœuds
    - Chaque bout de texte de part et d'autre d'un commentaire est un nœud textuel distinct

Modèle de données XPath (2/3)

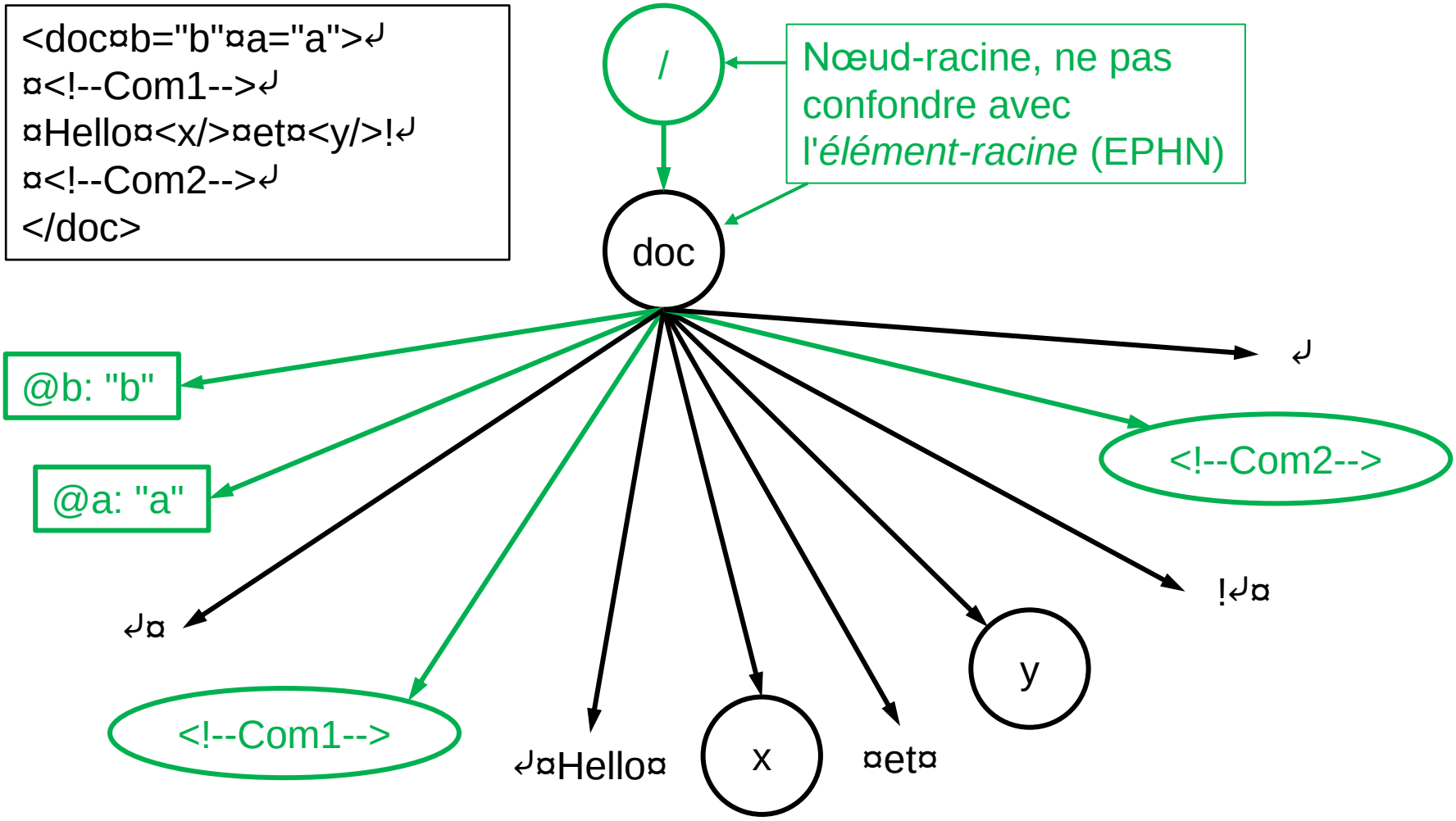
```
<doc¤b="b"¤a="a">↲
¤<!--Com1-->↲
¤Hello¤<x/>¤et¤<y/>!↲
¤<!--Com2-->↲
</doc>
```

200-Ex-XPath/doc-jouet.xml

Rappel :
Arbre inversé *non allégé*, selon *Premier tour d'horizon*



Modèle de données XPath (3/3)

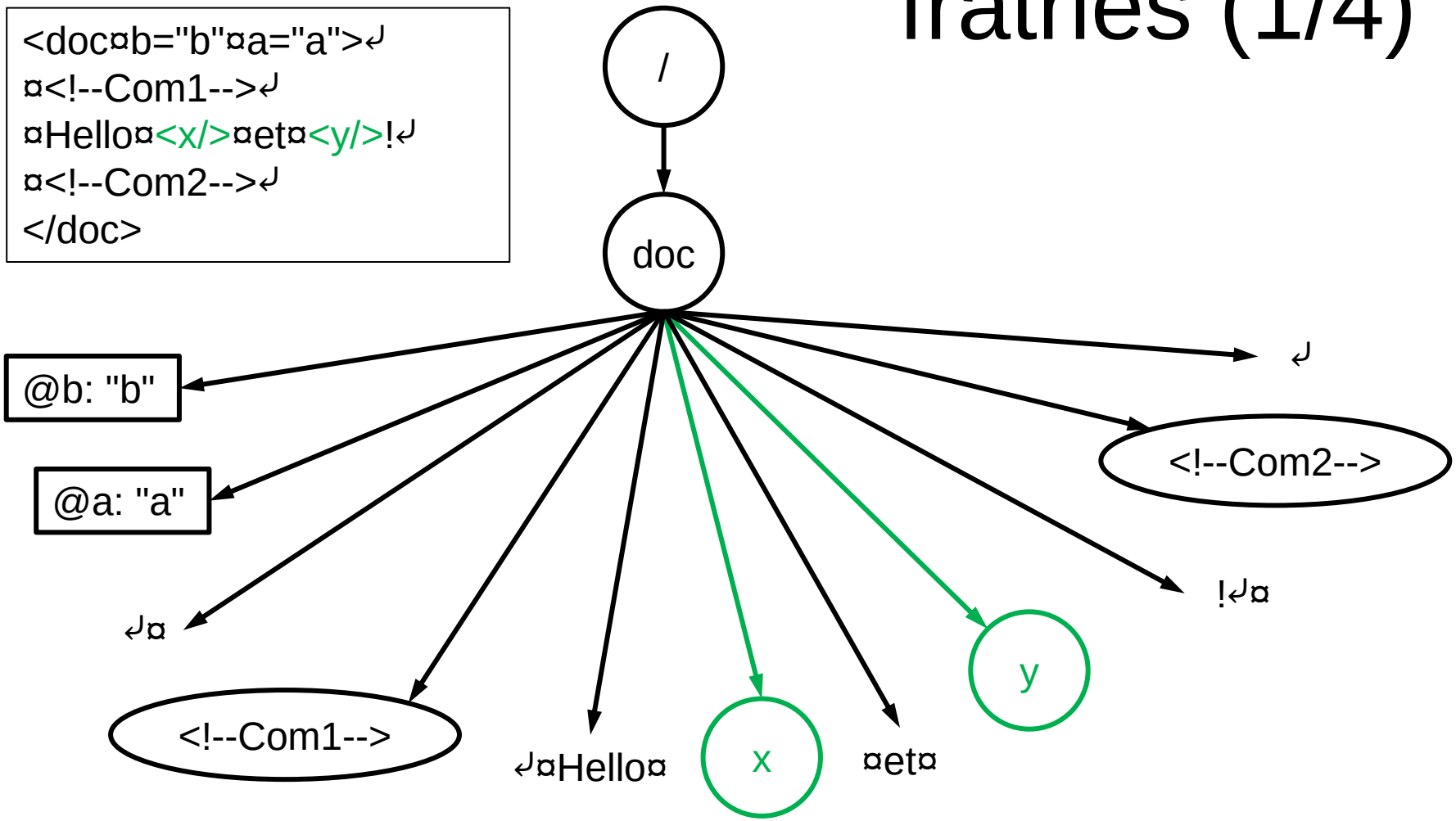


Un peu (plus) de généalogie…

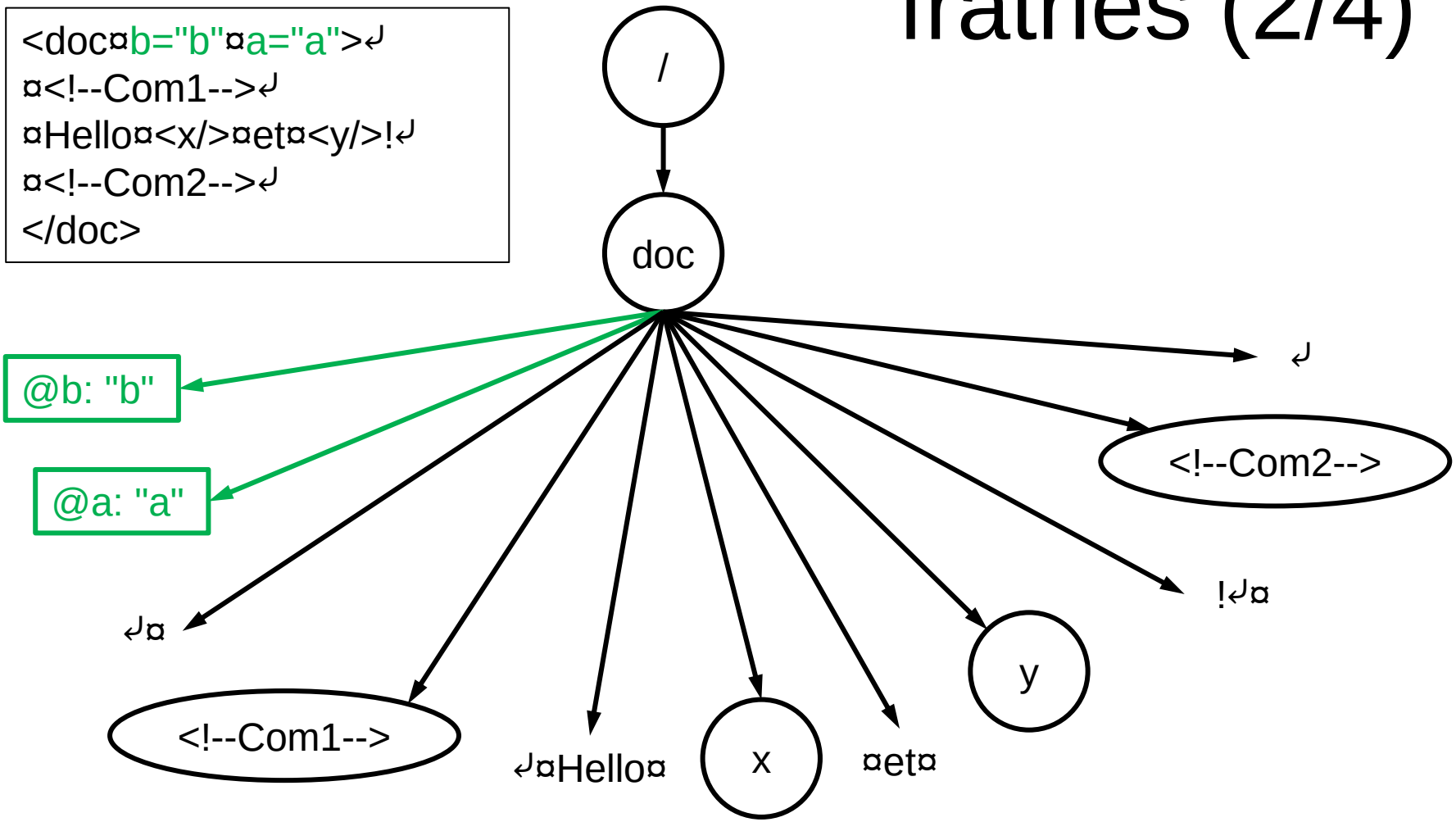
- Une *fratrie* est un ensemble de nœuds constitué de tous les enfants *du même type* (élément, texte, attribut, commentaire)* *et d'un même parent*
- Un nœud-élément peut donc être parent de jusqu'à *quatre** fratries différentes : une par type

*Trois types additionnels de nœuds sont définis dans XPath, mais ne sont pas vus dans le cours: *processing-instruction*, *namespace* et *root*

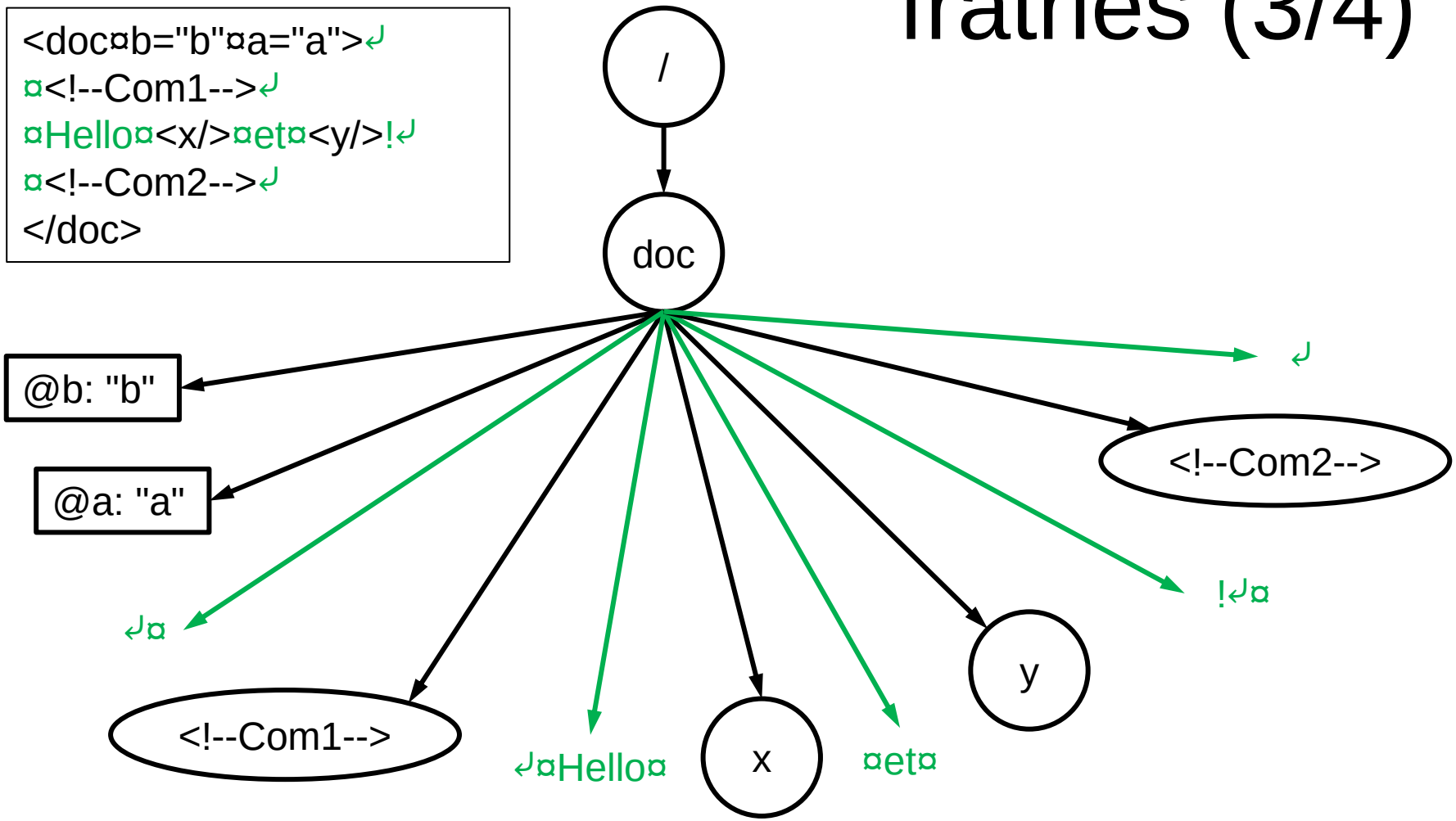
Les différentes fratries (1/4)



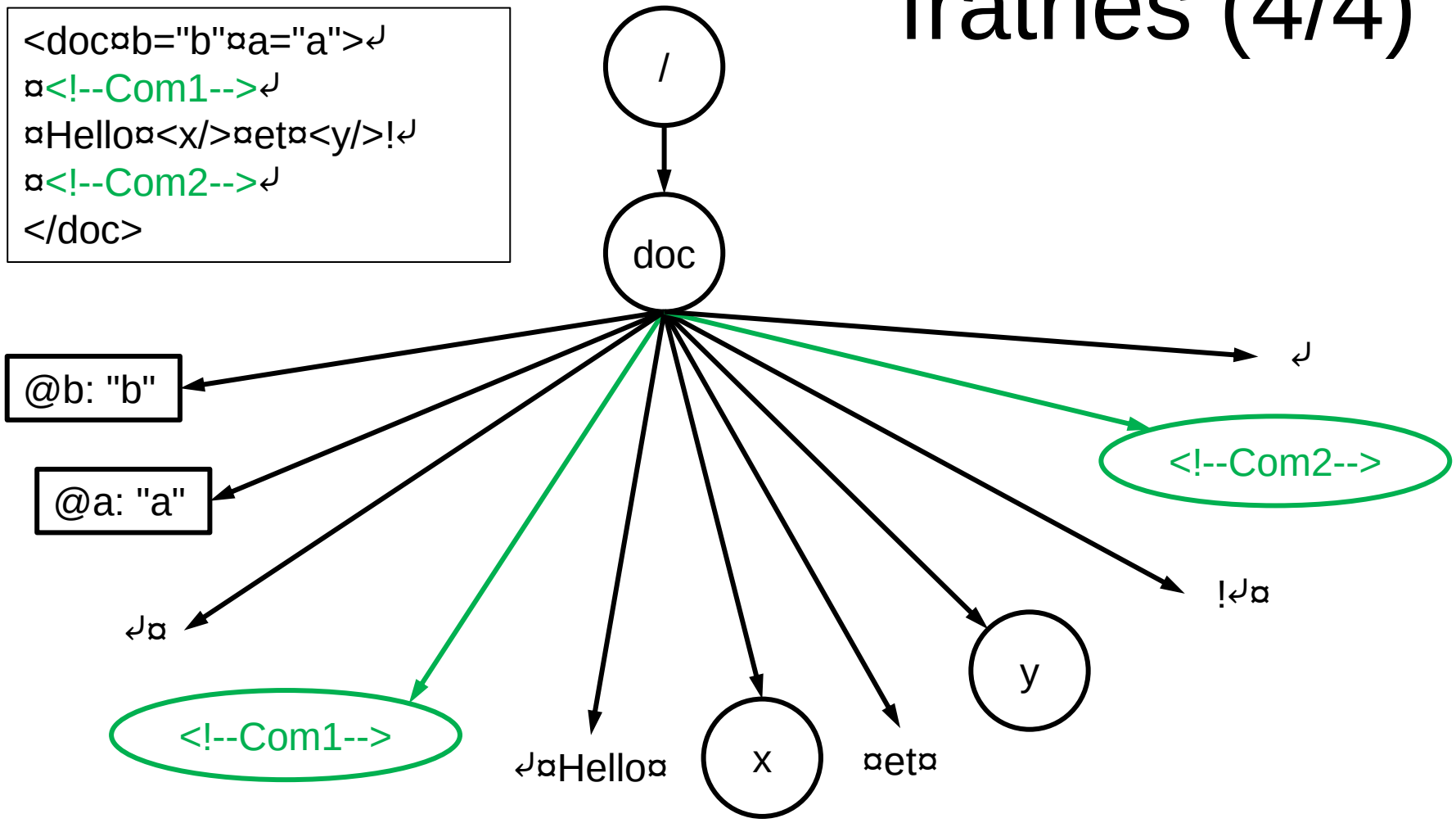
Les différentes fratries (2/4)



Les différentes fratries (3/4)



Les différentes fratries (4/4)



Noms de nœud

- Les nœuds de types *élément* et *attribut* ont un *nom*
  - Pour un nœud élément, c'est le *nom de l'élément* (ou *identificateur générique*)
  - Pour un attribut, c'est le *nom d'attribut*
- Les nœuds des autres types n'ont *pas de nom*
  - nœud textuel, commentaire

Expression XPath

- Une *expression XPath* est une expression du langage XPath qui "interroge" un document XML, et retourne :
  - Soit: un *ensemble de nœuds* extraits du modèle XPath du document (cet ensemble peut être vide)
  - Soit: une *valeur calculée* à partir du document
    - nombre, chaîne de caractères, valeur booléenne

Expressions qui retournent un ensemble de nœuds

Elles s'appellent
"chemins XPath"
ou simplement "chemins"

Tests de nœud (1/3)

- Une des composantes les plus importantes des chemins sont les *tests de nœud*
- Un test de nœud est une expression qui sélectionne certains nœuds parmi un ensemble de nœuds *candidats* (qui dépend du contexte)

Tests de nœud (2/3)

- Chaque test de nœud exprime un critère de sélection basé sur :
  - le type de nœud
  - et (parfois) le *nom* du nœud

Tests de nœud (3/3)

Test	**Sélectionne**
*	• Les nœuds élément
idGen (ex.: autr)	• Les nœuds élément dont le nom = *idGen*
text()	• Les nœuds texte
@*	• Les nœuds attribut
@*nomAttr* (ex.: @courriel)	• Les nœuds attribut dont le nom = *nomAttr*
comment()	• Les nœuds commentaire

Un contexte spécifique

- Le contexte :

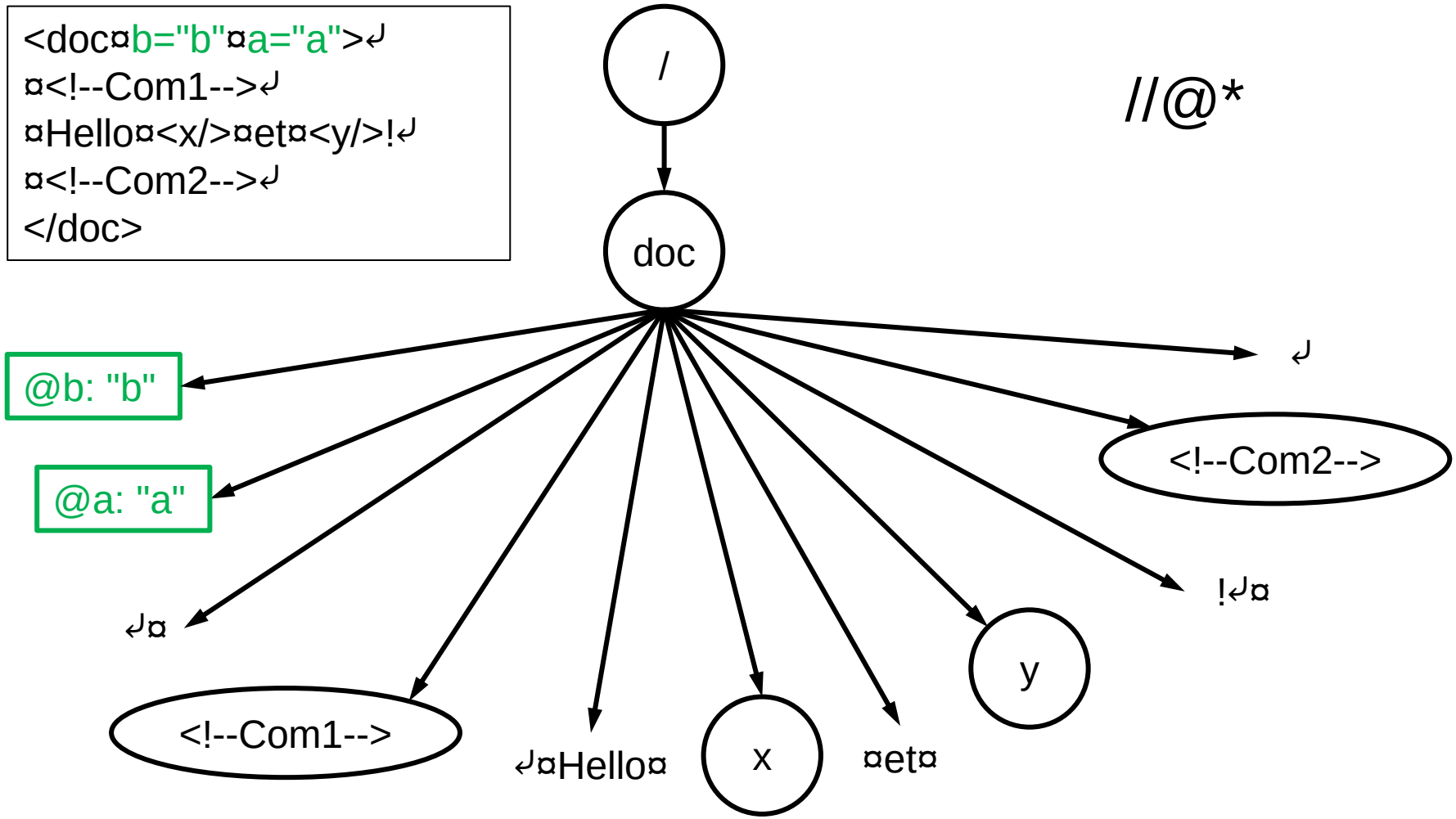
  // *précédant* un test-de-nœud

- applique le test de nœud à *tous* les nœuds du document
  - i.e. tous les nœuds du document sont *a priori* candidats pour être sélectionnés
- Permet d'expérimenter avec les tests de nœud pour bien les comprendre

Exemple 1

```
<doc¤b="b"¤a="a">↲
¤<!--Com1-->↲
¤Hello¤<x/>¤et¤<y/>!↲
¤<!--Com2-->↲
</doc>
```

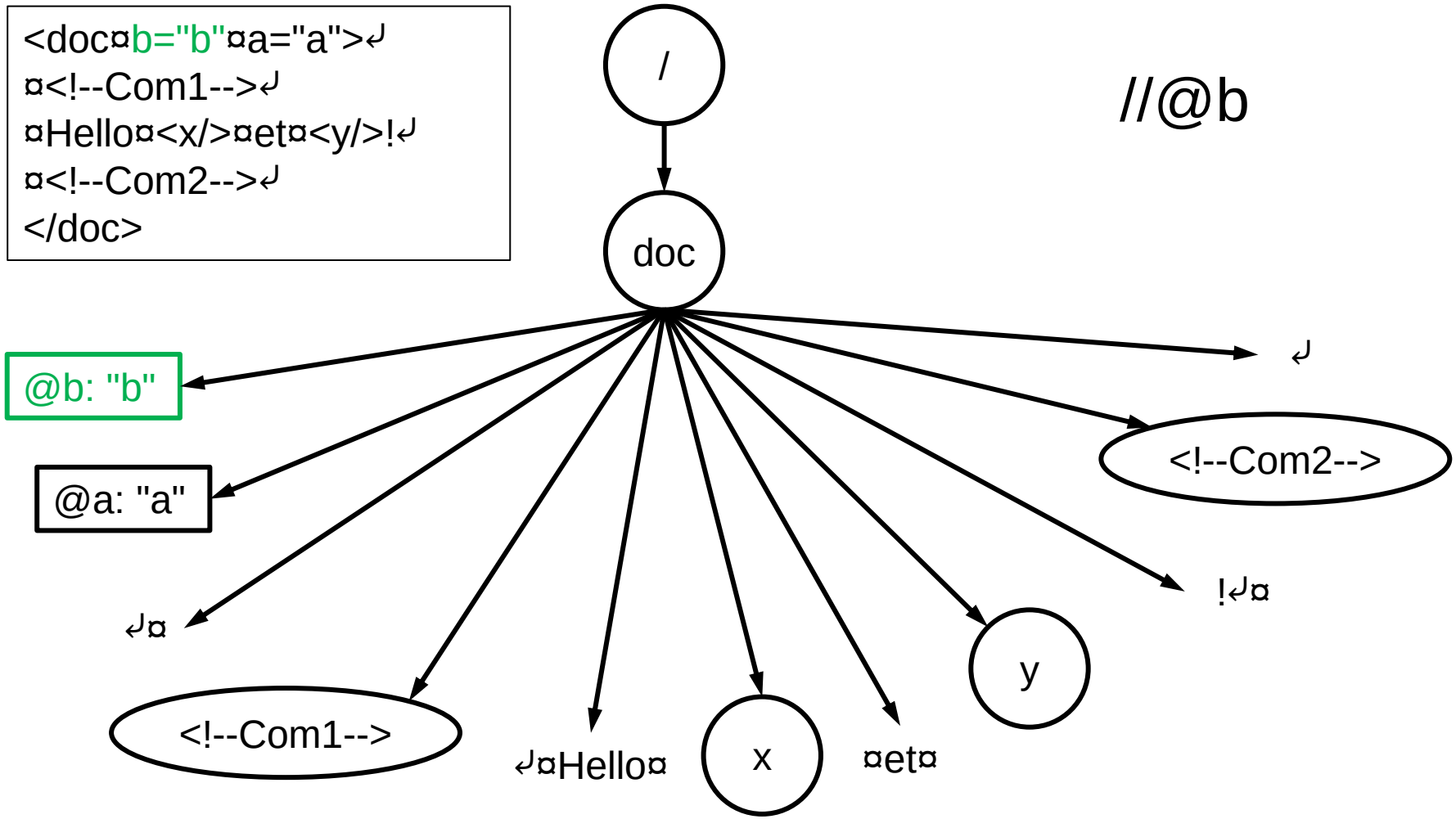
//@*



Exemple 2

```
<doc¤b="b"¤a="a">↲
¤<!--Com1-->↲
¤Hello¤<x/>¤et¤<y/>!↲
¤<!--Com2-->↲
</doc>
```

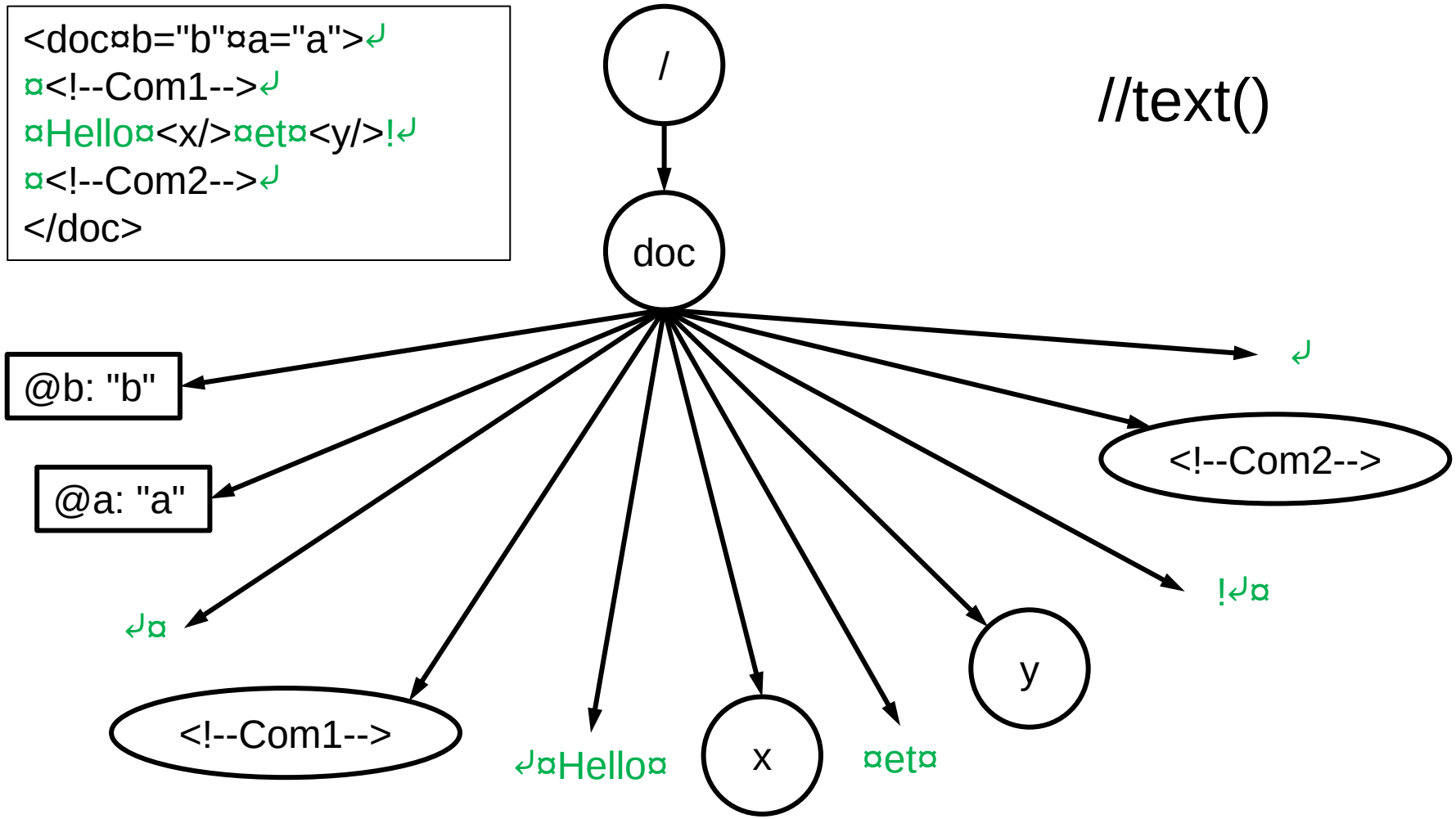
//@b



Exemple 3

```
<doc¤b="b"¤a="a">↲
¤<!--Com1-->↲
¤Hello¤<x/>¤et¤<y/>!↲
¤<!--Com2-->↲
</doc>
```

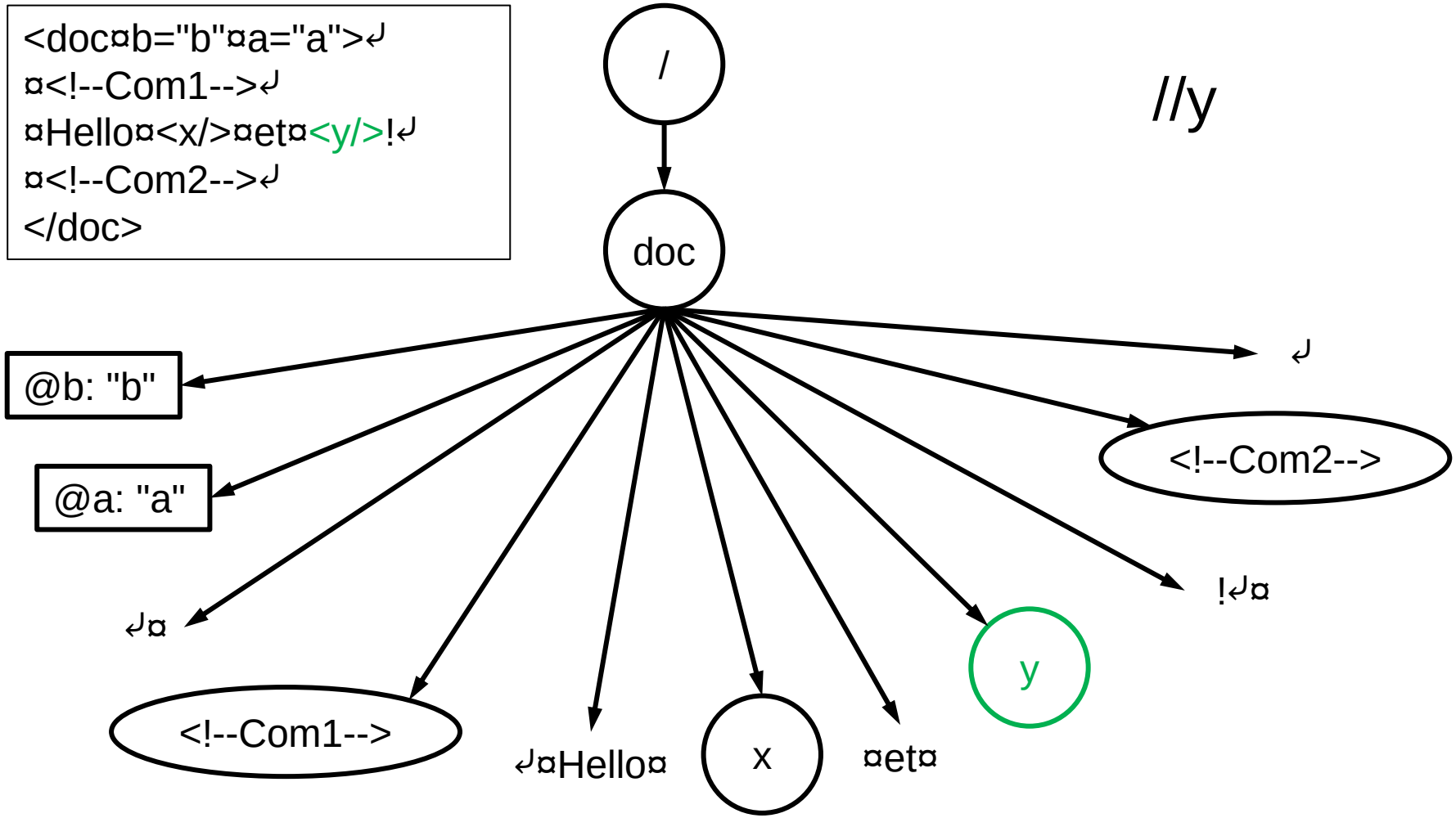
//text()



Exemple 4

```
<doc¤b="b"¤a="a">↲
¤<!--Com1-->↲
¤Hello¤<x/>¤et¤<y/>!↲
¤<!--Com2-->↲
</doc>
```

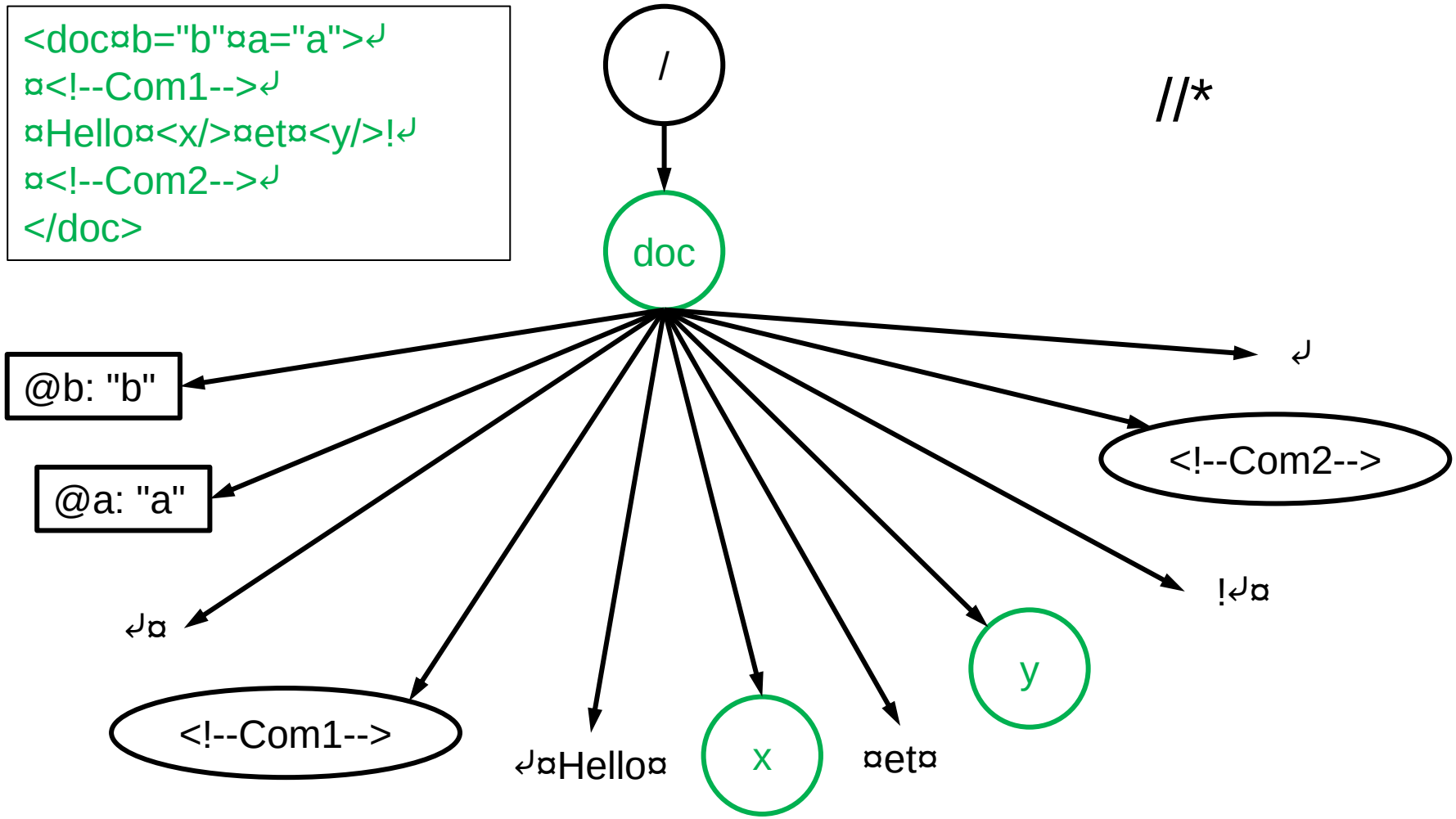
//y



Exemple 5

```
<doc¤b="b"¤a="a">↲
¤<!--Com1-->↲
¤Hello¤<x/>¤et¤<y/>!↲
¤<!--Com2-->↲
</doc>
```

//*



Il est possible qu'un test de nœud ne sélectionne aucun nœud…

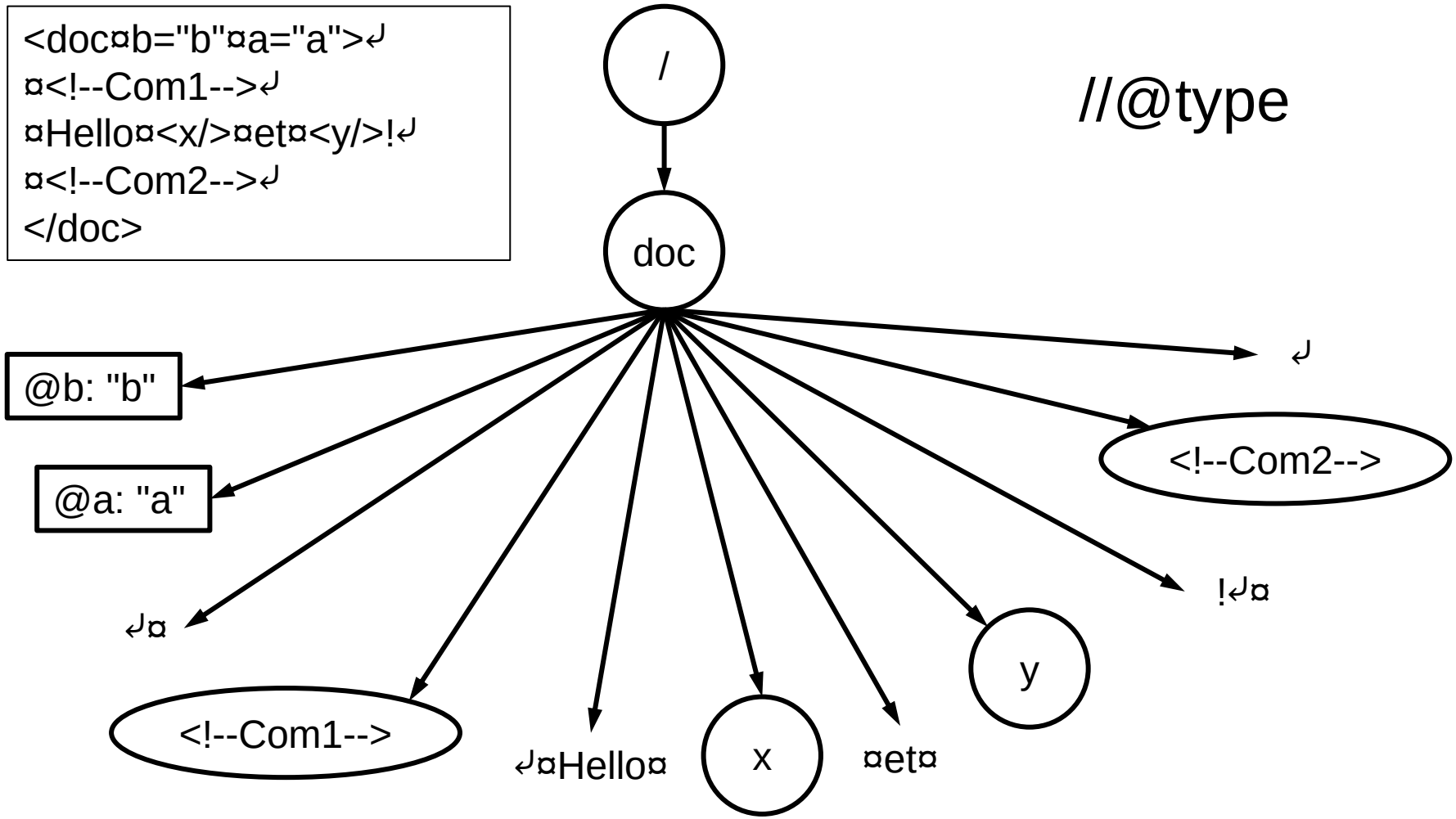
Le résultat est l'*ensemble vide*

Ce n'est pas une erreur

Exemple 6

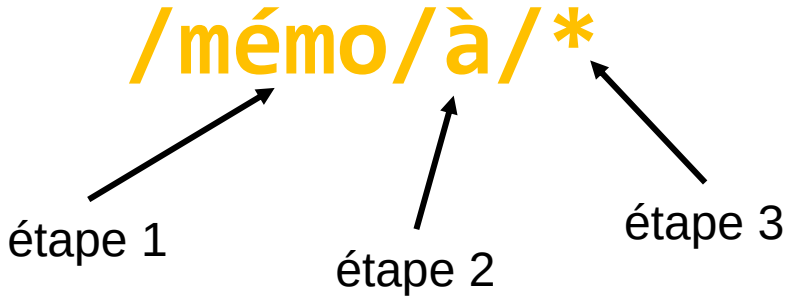
```
<doc¤b="b"¤a="a">↲
¤<!--Com1-->↲
¤Hello¤<x/>¤et¤<y/>!↲
¤<!--Com2-->↲
</doc>
```

//@type



Chemins absolus simples

- Un chemin *absolu* commence par "/"
- Est constitué d'*étapes* séparées par un "/"
- Chaque étape est un *test de nœud*
- Ex.:



Évaluation d'un chemin absolu simple

1. #étap = 1
2. étap-préc = { nœud-racine }
3. étap-cour = { }
4. Pour chaque nœud *n* dans étap-préc :
   Ajouter à étap-cour les enfants de *n*
5. étap-préc = nœuds de étap-cour qui satisfont le test de nœud de l'étape #étap
6. #étap = #étap + 1
7. Si l'étape #étap existe, retourner à 3
8. Le résultat du chemin est étap-préc

Ex. chemins absolus simples

```
<mémo>
  <auteur adr="lr@lr.org">Léa Roy</auteur>
  <!-- Vérifier adresse -->
  <date>2042-07-07</date>
  <prio><normal /></prio>
  <à>
    <nom adr="sp@sp.net">Syd Pine</nom>
    <nom adr="md@md.com">Mike Day</nom>
  </à>
  <cc>
    <nom>Joe Doll</nom>
  </cc>
  <corps>
    <par>Réunion <em>27 <em>mai</em> 2043</em>.</par>
    <par rôle="sig">Léa, PDG</par>
  </corps>
</mémo>
```

Chemins:

1- **/mémo/à/nom**

2- **/mémo/corps/*/em**

200-Ex-XPath/doc-simple.xml

Autres exemples

- /mémo/auteur
- /*/auteur
- /*/*/nom
- /*/à/*/@adr
- /*/*/*/@*

Étapes "descendants" (ou //)

- Si une étape commence par « // » au lieu de « / », au point 4. de l'évaluation, tous les *descendants* de *n* sont ajoutés à étap-cour…

Évaluation d'un chemin absolu avec étapes "//"

1. #étap = 1
2. étap-préc = { nœud-racine }
3. étap-cour = { }
4. Pour chaque nœud *n* dans étap-préc :
   Ajouter à étap-cour : les *enfants* de *n*
   OU les *descendants* de *n*, si l'étape #étap est "//"
5. étap-préc = nœuds de étap-cour qui satisfont le test de nœud de l'étape #étap
6. #étap = #étap + 1
7. Si l'étape #étap existe, retourner à 3
8. Le résultat du chemin est étap-préc

Exemples avec étapes "descendants" (//)

- //nom
- //à//nom
- //@adr
- //@*
- //*
- //em
- //em/em

Test de nœud "parent" (ou ..)

- ".." est un test de nœud spécial : "parent"
- Il sert à indiquer que l'étape est une étape "parent", dans quel cas, c'est le *parent* de *n* qui est ajouté au point 4. de l'évaluation
- Au point 5. lors d'une étape parent, ".." se comporte comme "*" (sélectionne tous les nœuds-éléments)…

Évaluation d'un chemin absolu

1. #étap = 1
2. étap-préc = { nœud-racine }
3. étap-cour = { }
4. Pour chaque nœud *n* dans étap-préc :
   Ajouter à étap-cour : les *enfants* de *n*
   OU les *descendants* de *n*, si l'étape #étap est "//"
   OU le *parent* de *n*, si l'étape #étap est ".."
5. étap-préc = nœuds de étap-cour qui satisfont le test de nœud de l'étape #étap
6. #étap = #étap + 1
7. Si l'étape #étap existe, retourner à 3
8. Le résultat du chemin est étap-préc

Exemples avec étapes "parent"

- //date/../*
- //@*/..
- //nom/..

200-Ex-XPath/ex-detailles.pdf

Exemples 1 à 5

Chemins relatifs (1/4)

- Un chemin qui ne commence pas par "/" est un chemin *relatif*
- Un chemin relatif commence donc par un test de nœud, qui constitue l'étape 1 du chemin
- Un chemin relatif est toujours évalué relativement à un "nœud courant", déterminé par le contexte

Chemins relatifs (2/4)

- Dans oXygen, le nœud courant est celui *où se trouve le curseur* au moment de l'évaluation du chemin relatif
  - Exception : si le curseur se trouve dans un nœud texte, c'est *son parent* (donc, un nœud élément) qui est le nœud courant

Chemins relatifs (3/4)

- La seule différence dans l'évaluation d'un chemin relatif est qu'au point 2., c'est le *nœud courant* qui est placé dans étap-préc, plutôt que le nœud-racine

Évaluation d'un chemin relatif

1. #étap = 1
2. étap-préc = { nœud courant }
3. étap-cour = { }
4. Pour chaque nœud *n* dans étap-préc :
   Ajouter à étap-cour : les *enfants* de *n*
   OU les *descendants* de *n*, si l'étape #étap est "//"
   OU le parent de *n*, si l'étape #étap est ".."
5. étap-préc = nœuds de étap-cour qui satisfont le test de nœud de l'étape #étap
6. #étap = #étap + 1
7. Si l'étape #étap existe, retourner à 3
8. Le résultat du chemin est étap-préc

Chemins relatifs (4/4)

- Pour faire de la première étape d'un chemin relatif une étape "descendants" (//), on inscrit : **`.//`** *avant* le premier test de nœud, p.ex. :

  `.//*`

  Cet exemple retourne tous les nœuds-éléments descendants du nœud courant

  Le faire avec `corps` comme nœud courant dans `doc-realiste.xml`

Deux chemins particuliers

- Le chemin absolu "/" → le nœud-racine
- Le chemin relatif "." → le nœud courant

- Faire des exemples de chemins relatifs avec oXygen

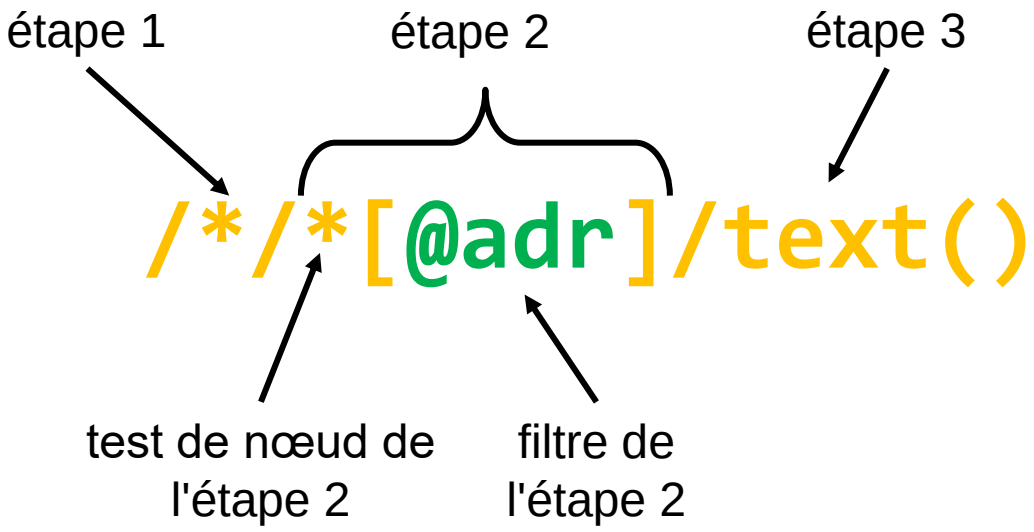
Filtres (1/4)

- Un *filtre* est un critère de sélection de nœuds appliqué à la fin d'une étape
- Il *laisse passer* vers l'étape suivante certains des nœuds de l'étape à laquelle il s'applique et en *élimine* d'autres
- Forme générale:

  [*filtre*] après le test de nœud d'une étape
  (non permis dans une étape parent "..");

Filtres (2/4)

- Exemple : `/*/*[@adr]/text()`



Filtres (3/4)

- Il peut y avoir plusieurs filtres à la fin d'une même étape; ils sont exécutés *de gauche à droite*, chacun travaillant sur le résultat du précédent; ex.:

  [@courriel][last()]

  – Le premier filtre travaille sur l’ensemble de nœuds sélectionnés par le test de nœud de l'étape

Évaluation d'un chemin absolu

1. #étap = 1
2. étap-préc = { nœud-racine }
3. étap-cour = { }
4. Pour chaque nœud *n* dans étap-préc :
   Ajouter à étap-cour : les *enfants* de *n*
   OU les *descendants* de *n*, si l'étape #étap est "//"
   OU le parent de *n*, si l'étape #étap est ".."
5. étap-préc = nœuds de étap-cour qui satisfont le test de nœud et les filtres de l'étape #étap
6. #étap = #étap + 1
7. Si l'étape #étap existe, retourner à 3
8. Le résultat du chemin est étap-préc

Évaluation d'un chemin relatif

1. #étap = 1
2. étap-préc = { nœud courant }
3. étap-cour = { }
4. Pour chaque nœud *n* dans étap-préc :
   Ajouter à étap-cour : les *enfants* de *n*
   OU les *descendants* de *n*, si l'étape #étap est "//"
   OU le parent de *n*, si l'étape #étap est ".."
5. étap-préc = nœuds de étap-cour qui satisfont le test de nœud et les filtres de l'étape #étap
6. #étap = #étap + 1
7. Si l'étape #étap existe, retourner à 3
8. Le résultat du chemin est étap-préc

Filtres (4/4)

- Un filtre est *lui-même une expression XPath*
- Cette expression est évaluée avec comme « nœud courant » chacun des nœuds soumis au filtre, tour à tour
- Pour chaque nœud, le résultat de l'évaluation détermine si ce nœud est *conservé* (ou *gardé*) par le filtre ou *éliminé*

Cas 1 : Le filtre est un chemin

- Lorsque le filtre est lui-même un chemin (il sera habituellement relatif), alors si l'évaluation retourne un ensemble non vide, le verdict est *conservé*
- Si le chemin qu'est le filtre ne retourne rien (l'ensemble vide), le verdict est *éliminé*

Le filtre est un chemin - exemples

- Exemples (à essayer après "//*") :

  [nom] : le nœud a-t-il un enfant élém. "nom"?

  [@adr] : le nœud a-t-il un enfant attrib. "adr"?

  [.//à] : le nœud a-t-il un descendant élém. "à"?

  [../cc] : le nœud a-t-il un frère élém. "cc" (ou est-il lui-même un élément "cc")?

  [*/em] : le nœud a-t-il un petit-fils élément "em"?

Cas 2 : Le filtre ≠ chemin

- Un *filtre* peut aussi être un type d'expression *autre qu'un chemin*, donc qui retourne *une valeur* (nombre, booléen, chaîne de caractères)
- Dans ce cas, le verdict est décidé comme suit :
  - si le résultat = la valeur booléenne « faux » ou la chaîne vide : *éliminé*
  - si le résultat = la valeur booléenne « vrai » ou une chaîne non vide : *conservé*
  - si le résultat est numérique : le filtre est *positionnel;* le verdict dépend du *rang* du nœud *dans sa fratrie*
    - voir Filtres positionnels plus loin

Expressions retournant une valeur

- Les expressions qui retournent une valeur utilisent, en plus des chemins :
  - Des constantes :
    - nombres : 0, 10, -5.5, etc.
    - chaînes : 'café', "thé", etc.
  - Des opérateurs et fonctions, par exemple :
    - comparaisons : =, >, <, !=, … (<)
    - arithmétiques : +, -, *, div, …
    - opérateurs booléens : and, or, not()
    - autres : contains(), …

Filtres de contenu textuel (1/2)

[. = 'mai'] : le *contenu textuel* du nœud est-il exactement égal à "mai"?

[@adr = 'lr@lr.org'] : le nœud a-t-il un enfant attribut "adr" dont le *contenu textuel* est "lr@lr.org"?

N.B.: Le *contenu textuel* pour un nœud attribut est la *valeur d'attribut.* Pour un nœud commentaire, c'est le *texte du commentaire.*

La casse est *significative.*

Filtres de contenu textuel (2/2)

[contains(@adr, 'md.com')] : le nœud a-t-il un enfant attribut adr dont le contenu textuel *contient* la chaîne "md.com"?

[contains(., 'Léa')] : le contenu textuel du nœud contient-il la chaîne "Léa"?

N.B.: La casse est toujours significative dans les comparaisons de chaînes avec =, contains, <, >, !=

Filtres positionnels (1/6)

- Si l'expression du filtre retourne une *valeur numérique*, le nœud est *conservé* si son *rang* parmi tous les nœuds de sa fratrie entrant dans le filtre est *égal à cette valeur*
- Il est *éliminé* si son rang parmi tous les nœuds de sa fratrie entrant dans le filtre est n'importe quelle autre valeur

  N.B.: La numérotation commence à 1

Filtres positionnels (2/6)

- Exemples

  [1] : Le nœud est-il le *premier* de sa fratrie à entrer dans le filtre ?

  [2] : … est-il le *deuxième* …

  [last()] : … est-il le *dernier* …

  last() est une fonction qui retourne le *nombre de nœuds de la fratrie du nœud courant qui entrent dans le filtre*

Filtres positionnels (3/6)

- Le rang (position) des nœuds dans une fratrie correspond à l'ordre d'apparition des nœuds dans le **texte** du document.
- Par exemple :

//*[1]	éléments premiers de fratrie
//*[2]	éléments seconds de fratrie
//*[last()]	éléments derniers de fratrie

Filtres positionnels (4/6)

- Une autre fonction utile pour les filtres positionnels :

  position()

  - Retourne la position elle-même du nœud courant parmi ceux de sa fratrie qui entrent dans le filtre
- Permet des conditions positionnelles plus génériques, par exemple :

  [position() < last()] : tous les nœuds de la fratrie, *sauf le dernier*

Filtres positionnels (5/6)

- Les filtres positionnels ne sont pas fiables avec les nœuds attributs, car l'ordre d'apparition de ces nœuds dans le modèle XPath est *imprévisible* (et c'est voulu)

  //@*[1] retourne un nœud pour chaque fratrie d'attributs, mais ce pourrait être n'importe lequel

- Bref, il est préférable de sélectionner les attributs par leur nom et non leur position

Filtres positionnels (6/6)

- Exemple:

  Même si on sait que les attributs sont toujours inscrits dans l'ordre "src" puis "alt", il faut les extraire par leur nom et non par leur position :

  //img/@src ~~//img/@*[1]~~

  //img/@alt ~~//img/@*[2]~~

200-Ex-XPath/ex-detailles.pdf

Exemples 6 et suiv. (avec filtre)

Chemins dans les filtres

- Comme noté précédemment, pratiquement tous les chemins qu'on retrouve dans un filtre sont *relatifs*, car on veut vérifier un critère additionnel sur des nœuds déjà sélectionnés; il est naturel que ce critère s'exprime relativement à ces nœuds

Opérateurs booléens (1/3)

- Les opérateurs booléens peuvent s'utiliser non seulement avec des valeurs booléennes, mais aussi avec *n'importe quelle expression XPath* (chemin ou non)
- La valeur de l'expression est interprétée comme "vrai" ou "faux" par l'opérateur booléen…

Opérateurs booléens (2/3)

- Sont interprétés comme "vrai" :
  - Un ensemble non vide de nœuds
  - Une valeur numérique différente de 0
  - Une chaîne de caractères non vide
- Sont interprétés comme "faux" :
  - Un ensemble vide de nœuds
  - Une valeur numérique 0
  - Une chaîne de caractères vide

Opérateurs booléens (3/3)

- Par exemple :

  not(*chemin*)

  - Donne vrai si *chemin* ne retourne rien
  - Donne faux si *chemin* retourne au moins 1 nœud

  chemin1 or *chemin2*

  - Donne vrai si au moins un des 2 chemins retourne quelque chose
  - Donne faux si les 2 chemins retournent l'ensemble vide

Autres opérateurs et fonctions

- En plus de ceux de la diapo 59
- count(*chemin*) nombre de nœuds retournés par *chemin*
- name(*chemin*) **nom** du premier nœud retourné par *chemin* (si c'est un nœud élément ou attribut, chaîne vide sinon)
- Tous les opérateurs et fonctions (liens vers le site du W3C)

Exemples

À exécuter sur : 200-Ex-XPath/doc-realiste.xml

- //*[count(nom) > 2]

  Éléments contenant directement aux moins 2 sous-éléments « nom »
- //*[contains(name(..), 'ité')]

  Éléments dont le nom du parent contient « ité »
- //*[count(@*) > 1]

  Éléments avec plus d’une spécification d’attribut

Analogie avec les URL (1/2)

- Peuvent être relatifs ou absolus
- Le "chemin d'accès" dans une URL absolue indique comment trouver un fichier sur un serveur à partir de sa racine

`http://myserv.ca/dossier/ssdos/fichier.pdf`

- Un chemin XPath absolu indique comment trouver des nœuds dans un document XML à partir de sa racine

Analogie avec les URL (2/2)

- Ressemblances:
  - "/" pour "descendre" dans la hiérarchie
  - ".." pour "remonter" vers le haut
- Mais :
  - Une URL vise à identifier un seul fichier
  - Un chemin XPath peut retourner > 1 nœud
  - Sans équivalent dans les URL:
    - Jokers (*, @*), filtres, étapes "descendants" (//)

