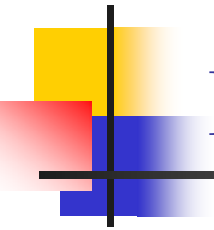
Description, parcours et transformation d'un document XML

Jacques Le Maitre
Université de Toulon et du Var



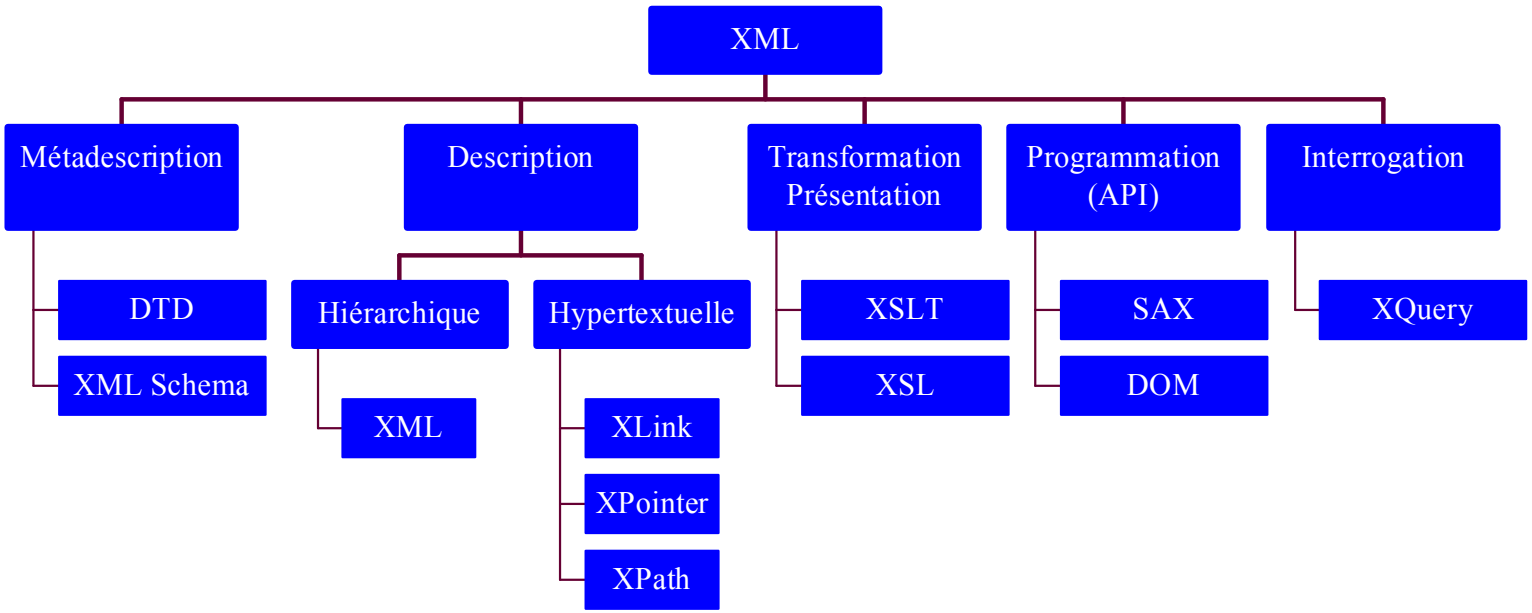
Différents aspects d'un document

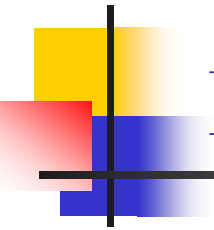
- Editorial
  - présentation du document.
- Signalétique
  - identification du document : ISBN, titre, auteurs, éditeur, année, ...
- Structurel
  - organisation logique du document : découpage en chapitres et en paragraphes, figures, annotations, ...
- Sémantique
  - sujet traité par le document.
- Multimédia
  - type des données véhiculées : textes, images, sons, animation, ...

De SGML à XML
en passant par HTML et HyTime

- SGML
  - Inventé par Charles Goldfarb, juriste chez IBM.
  - Du balisage typographique au balisage logique.
- HTML
  - Inventé par Tim Berners-Lee comme langage de description des pages du Web.
  - Concept d'ancre : hypertexte.
- HyTime
  - Langage très ambitieux pour la description des liens intra et inter documents.
- XML
  - Une synthèse de SGML, HTML et HyTime.

Le monde XML (ou plutôt un extrait !)





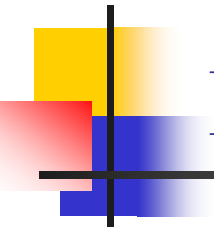
Plan

- Description : le langage XML,
- Parcours : le langage XPath,
- Structure hypertextuelle : le langage XLink,
- Transformation : le langage XSLT.

Description : le langage XML

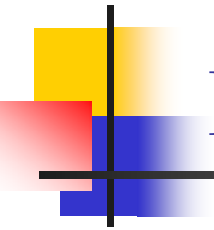
Structure logique d'un document XML

- 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**.
- 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.



Exemple

- Par exemple, un livre peut être représenté en XML par :
  - un élément *livre* composé :
    - d'un attribut *isbn* dont la valeur est l'ISBN de ce livre,
    - d'un élément *titre*,
    - de plusieurs éléments *chapitre*,
  - un élément *chapitre* est composé par :
    - un élément *titre*,
    - plusieurs éléments *paragraphe*,
  - un élément *titre* est composé du texte de ce titre,
  - un élément *paragraphe* est composé du texte de ce paragraphe.



DTD

- Les éléments qui décrivent un document peuvent être définis dans une **DTD** (Définition 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.

Un 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 (1)

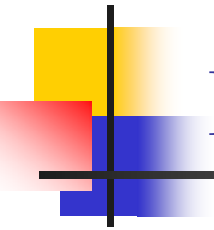
```
<?xml version="1.0"?>
<guide>
<titre>Itinéraires skieurs dans la vallée de la Clarée</titre>
<auteur>Jean-Gabriel Ravary</auteur>
<editeur>Le Polygraphe</editeur>
<annee>1991</annee>
…
<vallon id="V15">
<nom>Vallon des Muandes</nom>
<intro>
<para>Vallon situé à l'est du refuge des Drayères.</para>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

... Itinéraires ...

```
</vallon>
</guide>
```

Le guide « Itinéraires skieurs » en XML (2)

```
<itineraire 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>
</itineraire>
<itineraire 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>
</itineraire>
```



Production d'un document XML

- Pour produire un document XML, un utilisateur dispose des outils logiciels suivants :
  - éditeurs de texte classiques ;
  - éditeurs syntaxiques (Emacs ou XML Notepad, par exemple) qui permettent de produire des documents XML bien formés ;
  - analyseurs ou « parseurs », qui contrôlent la validité d'une DTD ou d'un document et le traduise sous une forme interne adaptée par son traitement par une application ;
  - éditeurs validants, qui combinent les fonctionalités des éditeurs syntaxiques et des analyseurs.

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.
- Dans le prologue et dans le contenu d'un élément, on peut insérer :
  - des **commentaires**,
  - des **instructions de traitement** (que nous n'étudierons pas dans ce cours) qui sont destinés aux applications traitant le document.
- Un document peut être découpé en **entités** enregistrées dans un ou plusieurs fichiers.

Noms 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.
- 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 (réservé à la séparation d'un nom et de son préfixe, voir ci-aprés « Espaces de noms »),
  - 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.

Elément

- Un élément est composé :
  - d'une **balise de début** qui contient le nom de l'élément et éventuellement ses attributs,
  - d'un **contenu**,
  - d'une **balise de fin**.
- 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

- **vide** :

```
<renvoi cible="I15.1"></renvoi> ou <renvoi cible="I15.1"/>
```

- **composé** d'éléments :

```
<intro>
   <para>Vallon situé à l'est du refuge …</para>
   <para>Le vallon le plus utilisé pour la traversée …</para>
</intro>
```

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

```
<nom>Col de Névache</nom>
<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="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 d'éléments : nous les appellerons **textes**,
  - les éléments.
- 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 annihiler ce rôle.
- Ceci peut être fait en insérant le texte contenant ces délimiteurs dans une section CDATA sous la forme suivante :

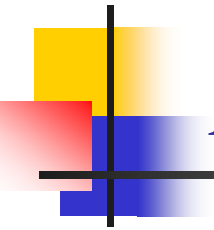
  `<![CDATA[`*texte contenant des délimiteurs*`]]>`

  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 :
  - 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 classés par vallon -->
```

Définition d'un élément

- Un élément est défini par la déclaration :
  `<!ELEMENT` *nom modèle de contenu*`>`
- Un élément mixte pouvant contenir des éléments T_1, …, T_n a pour modèle de contenu :
  `(#PCDATA |` T_1 `| … |` T_n`)*`
- Un élément composé d'une suite d'éléments T_1, …, T_n a pour modèle de contenu l'expression régulière construite sur le vocabulaire $\{T_1, \ldots, T_n\}$ 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`.

Définition d'attributs (1)

- A chaque type d'élément est attaché un ensemble d'attributs.
- Une définition d'attributs a la forme suivante :

  `<!ATTLIST` *nom-élément*

  nom-attribut type déclaration-de-défaut

  ...

  nom-attribut type déclaration-de-défaut>

  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.
- Les noms d'attributs sont locaux à chaque type d'élément : deux éléments de type différent peuvent avoir des attributs de même nom.

Définition d'attributs (2)

- 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, CR, LF, tabulation),
  - `NMTOKEN` ou `NMTOKENS` : nom ou suite de tokens de nom,
  - $(nom_1 \mid \ldots \mid nom_n)$: un des tokens de nom énumérés.
  - `ENTITY`, `ENTITIES` et `NOTATION` que nous n'étudierons pas.

Définition d'attributs (3)

- 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.

Règles de composition 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.
- 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, annee, vallon+)>
<!ELEMENT titre (#PCDATA)>
<!ELEMENT auteur (#PCDATA)>
<!ELEMENT editeur (#PCDATA)>
<!ELEMENT annee (#PCDATA)>
<!ELEMENT vallon (nom, intro, itineraire+)>
<!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 itineraire (nom, alt, cotation, num, para+)>
<!ATTLIST itineraire id ID #REQUIRED>
<!ELEMENT alt (#PCDATA)>
<!ELEMENT cotation (#PCDATA)>
<!ELEMENT num (#PCDATA)>
```

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 prédéfinies**,
  - les **entités caractères**,
  - les **entités générales** qui sont des fragments de l'élément du document,
  - les **entités paramètres** qui sont des fragments de DTD.

Déclaration d'une entité

- Déclaration d'une entité paramètre :
  - interne

    `<!ENTITY %` *nom* `"`*entité*`">`
  - externe

    `<!ENTITY %` *nom* `SYSTEM` *nom du fichier contenant l'entité*`>`
- Déclaration d'une entité générale interne :
  - interne

    `<!ENTITY` *nom* `"`*entité*`">`
  - externe

    `<!ENTITY` *nom* `SYSTEM "`*nom du fichier contenant l'entité*`">`
- Par exemple :

```
<!ENTITY % identificateur ID #REQUIRED>
<!ENTITY ref "refuge">
<!ENTITY vallon-muandes SYSTEM "mon_site/muandes.xml">
```

Référence à une entité

- Une référence à une entité paramètre a la forme suivante :

  ```
  %nom;
  ```

  Par exemple :

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

  au lieu de :

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

- Une référence à une entité générale a la forme suivante :

  ```
  &nom;
  ```

  Par exemple :

  ```
  <para>S'élever au-dessus du &ref; des Drayères en
    direction est. Suivre…</para>
  ```

  au lieu de :

  ```
  <para>S'élever au-dessus du refuge des Drayères en
    direction est. Suivre…</para>
  ```

Entités caractères

- Un caractère non disponible sur la station de travail peut être représenté par son code Unicode (voir ci-après « Codage des caractères ») en décimal ou en hexadécimal, sous la forme d'une référence a une entité :

  `&#`*code décimal*`;`

  `&#x`*code hexadécimal*`;`

- Par exemple :

  `&#38;` caractère &

  `&#x03A6;` lettre grecque Φ

Entités 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 :

  `<` référe le caractère <

  `>` référe le caractère >

  `&` référe le caractère &

  `&apos;` référe le caractère '

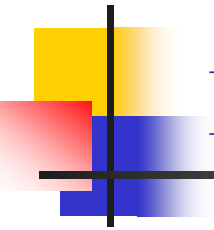
  `&quot;` référe le caractère "

- Par exemple, la phrase :

  « L'expression <ALT>2794</ALT> est un élément XML. »

  peut être représentée par l'élément suivant :

```
<phrase>L'expression &lt;ALT>2794&lt;/ALT> est un
élément XML.</phrase>
```

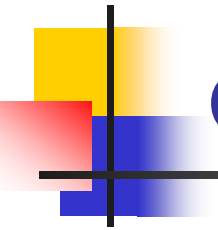


Remplacement des entités

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

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.
- 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.



Organisation d'un document XML valide

- 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 multifichier

```
<?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`.

Réalisation physique du document « Itinéraires skieurs »

```
<?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>
<editeur>Le Polygraphe</editeur>
<annee>1991</annee>
...
&vallon-muandes;
...
</guide>
```

```
<?xml version="1.0" ?>
<!ELEMENT guide (titre,
auteur+, editeur, annee,
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>
```

Espace de noms

- L'importation d'éléments ou d'attributs contenus dans des entités externes peut entraîner des conflits de noms.
- Ces conflits peuvent être évités en définissant des **espaces de noms**.
- Un **espace de noms** est identifié de façon unique par une URI (Uniform Resource Locator).
- Pour obtenir des noms uniques, il suffit de qualifier chaque nom par l'URI de l'espace de noms dont il provient.

  $\Rightarrow$ le nom obtenu est appelé **nom étendu**.
- Pour simplifier l'écriture des noms étendus, on associe un **préfixe** (un nom XML) à chaque espace de noms.

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*`"`

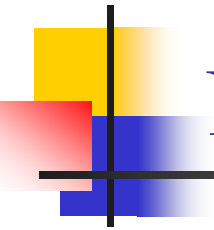
- On peut déclarer un espace de noms par défaut par l'attribut :

  `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.

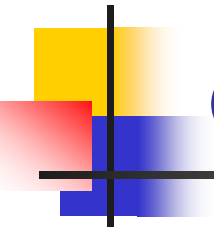
Exemple d'espace de noms

- Supposons que l'URI « monSite/dtdLivre.xml » contienne une DTD pour la description de livres et que les éléments `auteur`, `editeur` et `annee` du guide « Itinéraires skieurs » soient conformes à ces définitions, la description de ce guide pourrait être la suivante :

```
<guide xmlns:livre="monSite/dtdLivre.xml">
<livre:titre>Itinéraires skieurs dans la Vallée de la
   Clarée</livre:titre>
<livre:auteur>Jean-Gabriel Ravary</livre:auteur>
<livre:editeur>Le Polygraphe</livre:editeur>
<livre:annee>1991</livre:annee>
   …
<vallon>
<nom>Vallon des Muandes</nom>
…
</guide>
```

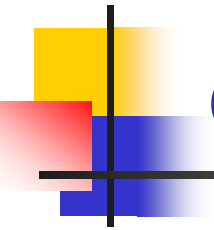
Codage des caractères (1)

- 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.



Codage des caractères (2)

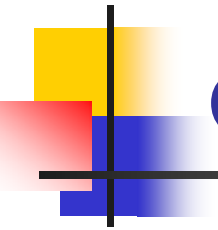
- Plusieurs codages de transformation ont été définis :
  - UTF-8 : permet de coder les caractères de l'UCS en longueur variable en codant sur un octet les caractères ASCII qui sont les plus fréquents.
  - UTF-16 : permet d'inclure des caractères de l'UCS-4 dans une chaîne codée en UCS-2.



Codage des caractères (3)

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

Parcours : le langage XPath



Objectifs du langage XPath

- Le langage XPath opère sur un document vu comme un arbre : l'**arbre du document**.
- Le langage XPath permet de sélectionner l'ensemble des nœuds que l'on peut atteindre en suivant, à partir d'un nœud donné de l'arbre d'un document, tous les chemins conformes à un modèle appelé **chemin de localisation**.

Exemple de chemin de localisation

- Soit le document XML :

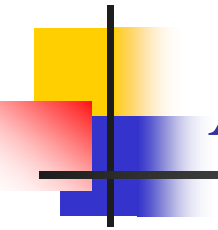
```
<?xml version="1.0"?>
<vallon>
  <nom>Vallon des Muandes</nom>
  <itineraire>
    <nom>Mont Thabor</nom>
    <altitude>3178</altitude>
  </itineraire>
</vallon>
```

- Le chemin de localisation :

```
/vallon/itineraire[altitude > 3000]/nom/text()
```

sélectionne le texte :

```
Mont Thabor.
```



Arbre d'un document (1)

- L'arbre d'un document comporte 7 types de nœuds associés à chaque constituant d'un document :
  - **racine** (associé au document lui-même),
  - **élément**,
  - **texte**,
  - **attribut**,
  - **espace de noms**,
  - **commentaire**,
  - **instruction de traitement**.

Arbre d'un document (2)

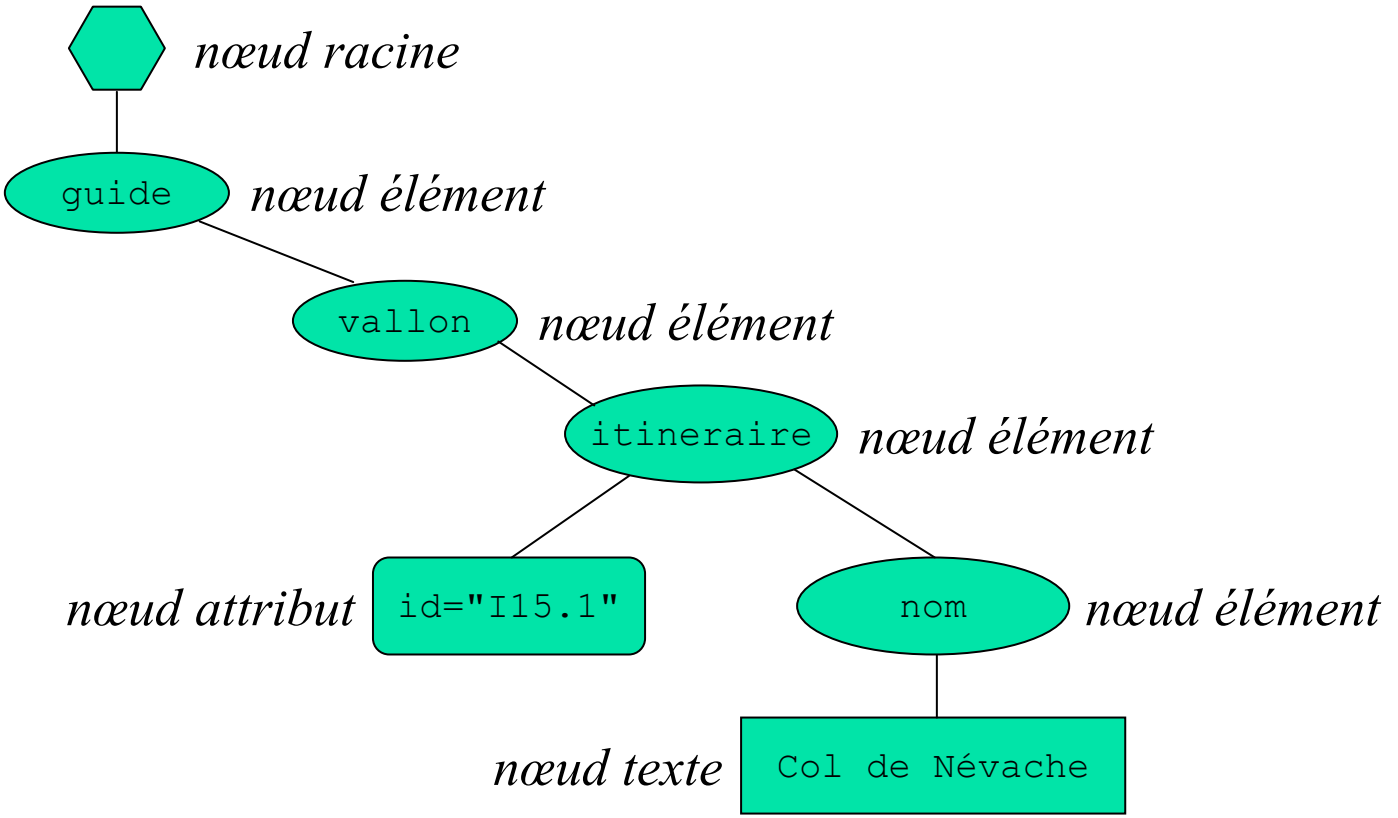
- Les **enfants** du nœud racine sont ses nœuds fils de type commentaire ou instruction de traitement.
- Les **enfants** d'un nœud élément sont ses nœuds fils de type élément, texte, commentaire ou instruction de traitement.
- Les **descendants** du nœud racine ou d'un nœud élément sont ses enfants ou les enfants de ses enfants.

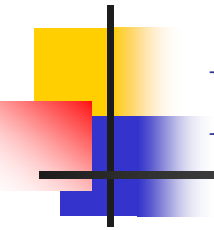
⇒ Les enfants et les descendants d'un nœud ne sont donc pas des nœuds de type attribut ou espace de noms.

Arbre d'un document (3)

- L'ensemble des nœuds de l'arbre d'un document est muni d'un ordre : l'**ordre du document** qui est l'ordre de lecture, dans le document XML, des constituants représentés par chaque nœud.
- Chaque nœud a une **valeur textuelle** :
  - la valeur textuelle du nœud racine est la concaténation des valeurs textuelles de ses descendants de type texte, dans l'ordre du document,
  - la valeur textuelle d'un nœud élément est la concaténation des valeurs textuelles de ses descendants de type texte, dans l'ordre du document,
  - la valeur textuelle d'un nœud texte est la chaîne de caractères constituant ce texte,
  - la valeur textuelle d'un nœud attribut est la chaîne de caractères constituant la valeur de cet attribut.

Arbre d'un document (extrait)





Expressions XPath (1)

- Une expression XPath a une valeur qui a l'un des 4 types suivants :
  - **ensemble de nœuds**,
  - **chaîne** de caractères UCS,
  - **nombre** (nombre flottant conforme à la norme IEEE 754),
  - **booléen**.
- Une expression XPath est évaluée dans un **contexte** constitué par :
  - un nœud : le **nœud contexte,**
  - deux entiers : la **position contexte** et la **taille contexte**,
  - un environnement (ensemble de liaisons variable-valeur),
  - une bibliothèque de fonctions prédéfinies,
  - les déclarations d'espaces de noms visibles dans l'expression.

Expressions XPath (2)

- 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,
  - d'opérateurs sur les ensembles de nœuds : parcours de chemin, filtrage,
  - 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.

Chemins de localisation

- Un **chemin de localisation** est un modèle de chemin dans l'arbre d'un document.
- Un chemin de localisation peut être **absolu** ou **relatif**.
  - Un chemin de localisation relatif commence au nœud contexte et est constitué d'une suite de pas de localisation.

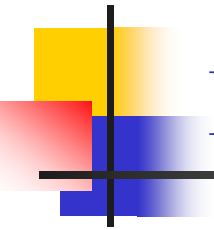
    Syntaxe :

    pas de localisation$_1$ / … / *pas de localisation*$_n$
  - Un chemin de localisation absolu est un chemin de localisation relatif qui commence à la racine du document.

    Syntaxe :

    /

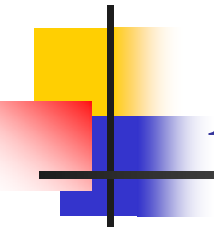
    / *chemin de localisation relatif*
- La valeur d'un chemin de localisation est l'ensemble des nœuds extrémité des chemins conforme à ce modèle.



Pas de localisation

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

  axe **: :** *test de nœud* *prédicat*$_1$ … *prédicat*$_n$

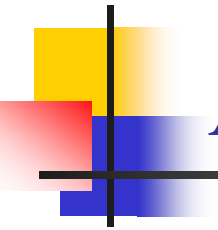


Axes (1)

- Un axe sélectionne, dans l'arbre du document et à partir du nœud contexte, l'ensemble des nœuds qui peuvent être atteints en suivant une certaine direction :
  - `child` : sélectionne les enfants du nœud contexte ;
  - `descendant` : sélectionne les descendants du nœud contexte ;
  - `parent` : sélectionne le père du nœud contexte, s'il en existe un ;
  - `ancestor` : sélectionne les ancêtres du nœud contexte ;
  - `following-sibling` : sélectionne les frères suivants du nœud contexte (si le nœud contexte n'est pas un élément, cet axe est vide) ;
  - `preceding-sibling` : sélectionne les frères précédents du nœud contexte (si le nœud contexte n'est pas un élément, cet axe est vide) ;

Axes (2)

- `following` : sélectionne les nœuds qui sont après le nœud contexte dans l'ordre du document excepté ses descendants ainsi que les nœuds attributs et espace de noms ;
- `preceding` : sélectionne les nœuds qui sont avant le nœud contexte dans l'ordre du document excepté ses ancêtres ainsi que les nœuds attributs et espace de noms ;
- `attribute` : sélectionne les nœuds de type attribut, fils du nœud contexte (si le nœud contexte n'est pas un élément, cet axe est vide) ;
- `namespace` : sélectionne les nœuds de type attribut espace de noms, fils du nœud contexte (si le nœud contexte n'est pas un élément, cet axe est vide) ;



Axes (3)

- `self` : sélectionne uniquement le nœud contexte ;
- `descendant-or-self` : sélectionne le nœud contexte et les nœuds descendants du nœud contexte ;
- `ancestor-or-self` : sélectionne le nœud contexte et les nœuds ancêtres du nœud contexte.

Axes (3)

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

Test de nœud

- Un **test de nœud** sélectionne parmi les nœuds de l'axe, ceux qui sont d'un certain type.
- Un test de nœud a l'une des formes suivantes :
  - *n* où *n* est un nom : sélectionne les nœuds de l'axe ayant le même type que le type principal de l'axe et dont le nom étendu est égal au nom étendu de *n* ;
  - `*` : sélectionne les nœuds de l'axe ayant le même type que le type principal de l'axe ;
  - `node()` : sélectionne tout nœud de l'axe ;
  - `text()` : sélectionne tout nœud de l'axe de type texte ;
  - `comment()` : sélectionne tout nœud de l'axe de type commentaire ;
  - `processing-instruction(`*n*`)` : sélectionne tout nœud de l'axe représentant une instruction de traitement de nom *n*.

Prédicat et filtrage (1)

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

  Syntaxe : `[`*expression*`]`
- Le filtrage par un prédicat est réalisé relativement à un axe que nous appellerons **axe de filtrage**.
  - Si le prédicat appartient à un pas de localisation, l'axe de filtrage est celui de ce pas,
  - Si le prédicat appartient à une expression de filtrage, l'axe de filtrage est l'axe `child`.

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

Prédicat et filtrage (2)

- Le filtrage d'un ensemble de nœuds E_1 par un prédicat [*b*] produit un nouvel ensemble de nœuds E_2 construit de la façon suivante :
  - taille contexte := cardinalité de E_1
  - E_2 := {}
  - Pour chaque nœud *n* de E_1 :
    - nœud contexte := *n*
    - position contexte := position de proximité de *n* dans E_1
    - si *valeur*(*b*) = `true`, après conversion éventuelle, alors
      - $E_2 := E_2 \cup \{n\}$

Valeur d'un chemin de localisation

- Un chemin de localisation relatif

  $p_1/\ldots/p_n$

  a pour valeur l'ensemble de nœuds E_2 construit de la façon suivante :
  - E_1 := {nœud contexte}
  - Pour i de 1 à n :
    - E_2 := {}
    - Pour chaque nœud n de E_1 :
      - nœud contexte := n
      - $E_2 := E_2 \cup valeur(p_i)$
    - $E_1 := E_2$
- La valeur d'un chemin de localisation absolu

  $/c$

  est $valeur(c)$ dans un contexte où le nœud contexte est le nœud racine.

Valeur d'un pas de localisation

- La valeur d'un pas de localisation

  $a::t\ \ p_1 \ldots p_n$

  à partir d'un nœud contexte n est l'ensemble de nœuds E_{n+2} construit de le façon suivante :

  - E_1 := ensemble des nœuds sélectionnés par l'axe a à partir de n
  - E_2 := sous-ensemble des nœuds de E_1 qui vérifient le test de nœud t
  - Pour i de 1 à n :
    - E_{i+2} := filtrage de E_{i+1} par le prédicat p_i

Fonctions de conversion

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

Fonctions de comparaison

- L'expression v_1 θ v_2 (où θ est =, !=, <, <=, > ou >=) est vraie dans les cas suivants et fausse dans les autres :
  - v_1 et v_2 sont des ensembles de nœuds et il existe un nœud n_1 de v_1 et un nœud n_2 de v_2 tel que *valeur-textuelle*(n_1) θ *valeur-textuelle*(n_2) est vraie ;
  - v_1 (resp. v_2) est un ensemble de nœuds et
    - v_2 (resp. v_1) est un booléen et `boolean`(v_1) θ v_2 (resp. v_1 θ `boolean`(v_2)) est vraie ;
    - v_2 (resp. v_1) est une chaîne ou un nombre et il existe un nœud n de v_1 (resp. v_2) tel que *valeur-textuelle*(n) θ v_2 (resp. v_1 θ *valeur-textuelle*(n)) est vraie ;
  - aucun des opérandes n'est un ensemble de nœuds et
    - θ est = ou != et
      - l'un des opérandes est un booléen et `boolean(`v_1`)` θ `boolean(`v_2`)` est vraie ;
      - l'un des opérandes est un nombre et `number(`v_1`)` θ `number(`v_2`)` est vraie ;
      - l'un des opérandes est une chaîne et `string(`v_1`)` θ `string(`v_2`)` est vraie ;
    - θ est <, <=, > ou >= et *number*(v_1) θ *number*(v_2) est vraie.

Conversion automatique

- Si la valeur *v* d'un prédicat n'est pas un booléen, elle est convertie en booléen de la façon suivante :
  - si *v* est un nombre égal à la position contexte, *v* est convertie en `true` sinon *v* est convertie en `false` ;
  - si *v* n'est pas un nombre, elle est convertie en `boolean(`*v*`)`.

  On peut donc écrire `[`*i*`]` au lieu de `[position() = `*i*`]`.
- Si la valeur *v* de l'argument d'une fonction prédéfinie n'a pas le type attendu (c.-à-d. celui de l'argument formel correspondant), *v* est automatiquement convertie, si cela est possible, en utilisant les fonctions de conversion `string`, `number` et `boolean`.

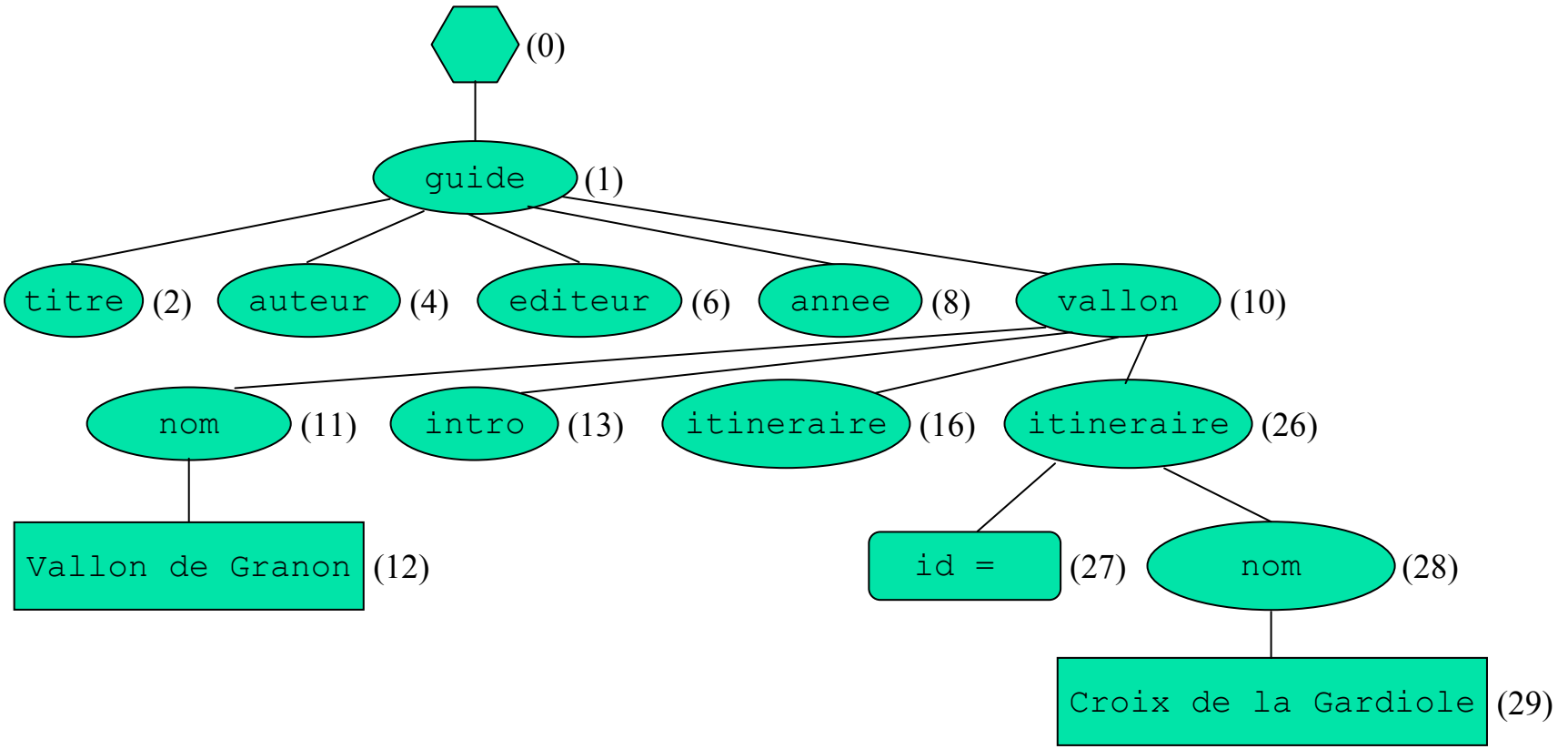
Fonctions sur les ensembles de nœuds (1)

- Position et taille contexte
  - Les expressions `position()` et `last()` ont pour valeurs respectives la position contexte et la taille contexte.
- Union
  - L'expression e_1 | e_2 a pour valeur l'union des ensembles de nœuds *valeur*(e_1) et *valeur*(e_2).
- Parcours de chemin
  - L'expression *e*/*p* a pour valeur l'ensemble des nœuds extrémité des chemins de localisation conformes au modèle *p* commençant à chaque nœud de l'ensemble de nœuds *valeur*(*e*).
  - L'expression *e*//*p* est équivalente à *e*/`descendant-or-self::node()`/*p*

Fonctions sur les ensembles de nœuds (2)

- Filtrage
  - L'expression *e*`[`*p*`]` produit l'ensemble de nœuds obtenu par filtrage de l'ensemble *valeur*(*e*) par le prédicat *p*. L'axe de filtrage est l'axe `child`.
- Accès par identifiant :
  - L'expression `id(`*s*`)` où *s* est une chaîne de caractères composée d'une suite d'identificateurs $n_1, \ldots, n_k$ est l'ensemble des nœuds élément identifiés par ces identificateurs.

Exemple de parcours d'un chemin de localisation : l'arbre du document



Exemple de parcours d'un chemin de localisation : le chemin à parcourir

- Nom du 2ème itinéraire du Vallon de Granon :

```
/descendant::vallon[child::nom = "Vallon de
  Granon"]
/child::itineraire[position() = 2]
/child::nom
/child::text()
```

Exemple de parcours d'un chemin de localisation : le parcours (1)

- 1er pas :
  - nœud contexte (0)
    - `descendant` ⇒ {(1), (2), …, (10), …, (110), …}
    - `vallon` ⇒ {(10), (110), …}
    - `child::nom = "Vallon de Granon"`
      - nœud contexte (10)
        - `child::nom` ⇒ {(11)}
        - {11} = `"Vallon de Granon"` ⇒ `true`, car *valeur-textuelle*(11) = `"Vallon de Granon"`

        ⇒ le nœud (10) est sélectionné
      - aucun autre nœud ne vérifie le prédicat

⇒ {(10)}

Exemple de parcours d'un chemin de localisation : le parcours (2)

- 2ème pas :
  - nœud contexte : (10)
    - `child::itineraire` ⇒ {(16), (26), (40), (54), (66), (84), (98)}
    - `[position() = 2]`
      - nœud contexte (16)
        - `position()` ⇒ `2`
        - `position() = 1` ⇒ `true`

        ⇒ le nœud (10) est éliminé
      - nœud contexte (26)
        - `position()` ⇒ `2`
        - `position() = 2` ⇒ `true`

        ⇒ le nœud (26) est sélectionné
      - les nœuds (40), (54), (66), (84), (98) sont éliminés

⇒ {(26)}

Exemple de parcours d'un chemin de localisation : le parcours (3)

- 3ème pas :
  - nœud contexte : (26)
    - `child::nom` ⇒ {(28)}

  ⇒ {(28)}
- 4ème pas :
  - nœud contexte : (28)
    - `child::nom` ⇒ {(29)}

  ⇒ {(29)}
- Le nœud atteint est le nœud texte « Croix de la Gardiole ».

Abréviations

- Afin de faciliter la lecture des chemins de localisation, les abréviations suivantes sont autorisées :
  - l'axe `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()/`.
  - le prédicat `[`*i*`]`, est l'écriture abrégée du prédicat `[position() =` *i*`]`.

Exemples d'expressions XPath (1)

- Nom des itinéraires du vallon des Muandes cotés ** et ne comportant pas de notes de prudence.

```
/guide/vallon[nom = "Muandes"]
/itinéraire[cotation = "**" and
            not para/note[@type = "prudence"]]
/nom/text()
```

- Nom des vallons possédant au moins deux itinéraires cotés **** ?

```
/guide
/vallon[count(itineraire[cotation = "****"])
        > 1]
/nom/text()
```

Exemples d'expressions XPath (2)

- Nom du deuxième 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 nœud par rapport à sa position de proximité. Ici on cherche le deuxième des itinéraires **.

- Textes de toutes les notes de type « prudence »

```
//note[@type = "prudence"]/text().
```

- Paragraphes citant le Refuge des Drayères.

```
//para[contains(text(), "Refuge des Drayères")].
```

Exemples d'expressions XPath (3)

- Nom du vallon qui précède et nom du vallon qui suit le vallon des Muandes ?

```
(((//vallon[nom = "Vallon des Muandes"]
  /preceding-sibling::vallon[1]) |
 (//vallon[nom = "Vallon des Muandes"]
  /following-sibling::vallon[1]))/nom/text()
```

(Cette expression comme les suivantes n'est pas un chemin de localisation car elle ne commence pas par / ou par un pas de localisation.)

Exemples d'expressions XPath (4)

- Nom du vallon dans lequel est situé l'itinéraire I15.2 ?

```
id("I15.2")/parent::vallon/nom/text()
```

- Nom de l'itinéraire cité dans la description de l'itinéraire I15.2 ?

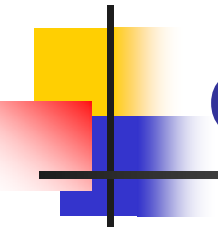
```
id(//itinéraire[@id = "I15.2"]/voir/@cible)
/nom/text()
```

Pointeurs X

- Un **pointeur X** permet de localiser une ressource distante constituée par un fragment de document.
- Un pointeur X est exprimé en termes du langage XPointer qui est une extension du langage XPath.
- Par exemple la ressource constituée par le premier itinéraire du Vallon des Muandes pourra être identifiée par l'une des deux URI suivantes :

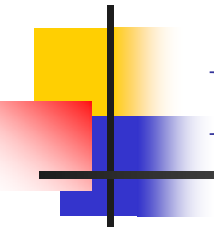
```
mon_site/itineraires_skieurs.xml#xpointer(id("I15.1"))
mon_site/itineraires_skieurs.xml#
   xpointer(guide/vallon[nom = "Vallon des
   Muandes"]/itineraire[1])
```

Liens hypertextuels : le langage XLink



Objectifs du langage XLink

- La description de la structure hypertextuelle d'un document XML est faite au moyen :
  - du langage XLink qui permet de décrire un lien en termes d'éléments XML,
  - du langage XPointer qui permet de localiser les fragments de documents qui constituent les extrémités d'un lien.

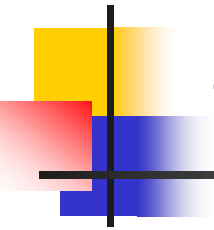


Liens simples et liens étendus

- Le langage XLink permet de représenter deux sortes de liens :
  - les **liens étendus**,
  - les **liens simples** qui sont des cas particuliers de liens étendus.

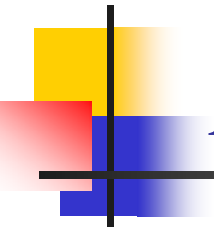
Liens et ressources

- Un lien étendu relie plusieurs **ressources** appelées **ressources participantes**.
- Une ressource est une unité d'information identifiée de façon unique par son **URI** (Uniform Ressource Identifier).
- Une ressource peut être un document ou un fragment de document, une image, un programme…
- Lorsqu'une ressource est un fragment de document XML, son URI est composée d'une base qui identifie le document et d'un pointeur X (voir ci-après) qui localise le fragment dans le document.
- Parmi les ressources participantes on distingue :
  - les **ressources locales** incluses dans la description du lien,
  - les **ressources distantes** décrites hors du lien et qui sont identifiées par leur URI.



Sémantique

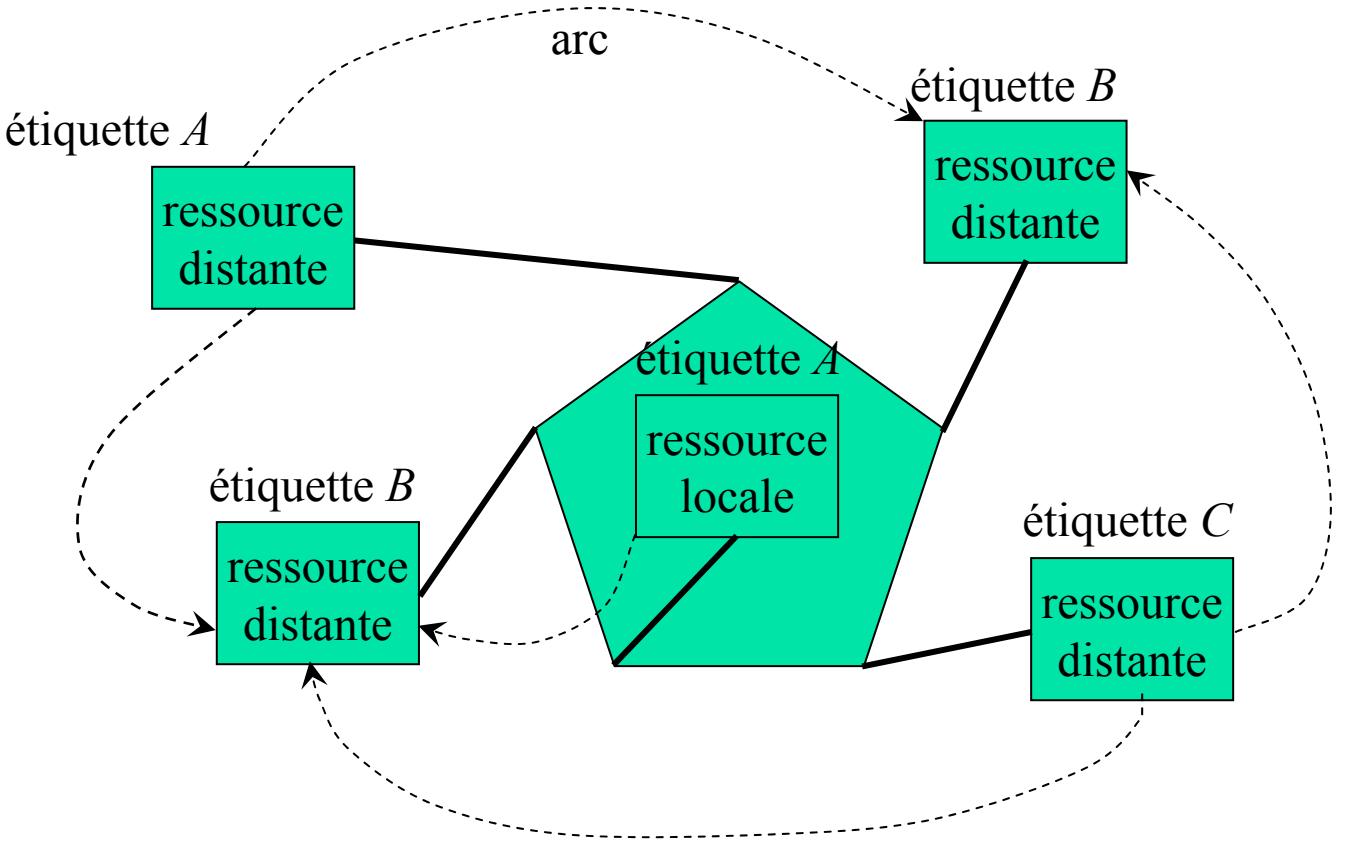
- Une sémantique peut être attachée à
  - un lien dans son entier,
  - à chacune de ses ressource participantes,
  - aux arcs qui les unissent (voir ci-après).
- La sémantique d'un lien, d'une ressource ou d'un arc en spécifie :
  - son **rôle** : un nom,
  - son **titre** : une description plus ou moins détaillé de ce rôle.

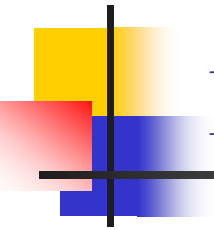


Arcs

- Un arc spécifie comment passer, à l'intérieur d'un lien, d'une **ressource de départ** à une **ressource d'arrivée**.
- Afin de définir les arcs, les ressources participantes peuvent être identifiées par une **étiquette**.
- Un arc est défini par :
  - l'étiquette de la ressource de départ et celle de la ressource d'arrivée,
  - sa sémantique,
  - son **comportement** : comment et quand présenter la ressource d'arrivée.
- Un arc entre :
  - une ressource locale et une ressource distante est dit **sortant**,
  - une ressource distante et une ressource locale est dit **entrant**.

Lien étendu

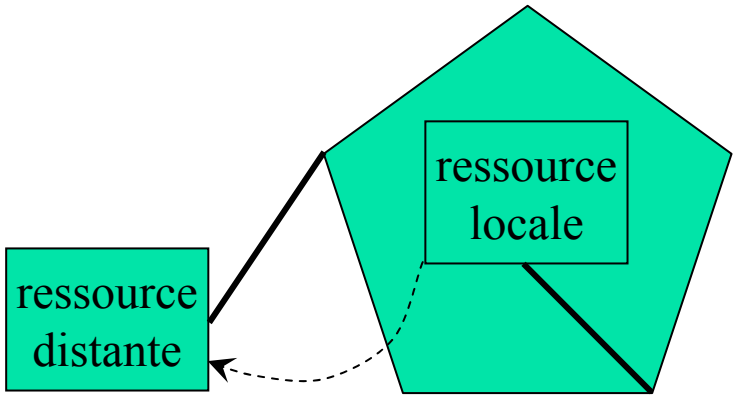

arc
étiquette A
ressource distante
étiquette B
ressource distante
étiquette A
ressource locale
étiquette B
ressource distante
étiquette C
ressource distante

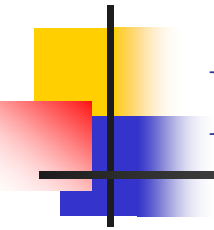


Lien simple

- Un lien simple est un cas particulier de lien étendu réduit à :
  - une ressource locale,
  - une ressource distante,
  - un arc entre ces deux ressources.
- Dans un lien simple la sémantique de la ressource distante et celle de l'arc sont confondues.

Lien simple





Eléments de liaison

- Un lien étendu est décrit par un élément XML composé de sous-éléments (**éléments de liaison**) qui décrivent les constituants de ce lien :
  - localiseurs des ressources distantes,
  - ressources locales,
  - titre (sémantique du lien).

Elément « resource »

- Un élément de type « resource » décrit une ressource locale.
- Il doit avoir l'attribut suivant :
  - `type="resource"`
- Il peut avoir l'un des attributs suivants :
  - `role="`*URI d'une ressource contenant une explication du rôle de la ressource (CDATA)*`"`
  - `title="`*description de la ressource (CDATA)*`"`
  - `label="`*étiquette de la ressource (NMTOKEN)*`"`
- Il peut avoir un sous-élément `title` pour décrire la ressource quand cette description n'est pas un simple texte.

Elément « locator »

- Un élément de type « locator » décrit une ressource distante.
- Il doit avoir les attributs suivants :
  - `type="locator"`
  - `href="`*URI de la ressource distante (CDATA)*`"`
- Il peut avoir, comme les éléments « locator » les attributs `role`, `title` ou `label` ou un sous-élément `title`.

Elément « arc »

- Un élément de liaison « arc » décrit un type d'arc.
- Il doit avoir l'attribut suivant :
  - `type="arc"`
- Il peut avoir les attributs suivants :
  - `arcrole` de type `CDATA`

    URI d'une ressource expliquant le rôle de ce type d'arc
  - `title` de type `CDATA`

    Explication du rôle de ce type d'arc
  - `from` et `to` de type `NMTOKEN`

    Etiquettes (« label ») de la ressource origine et de la ressource cible.
  - `show`
  - `actuate`

Elément « arc » : attribut `show`

- L'attribut `show` définit la présentation de la ressource d'arrivée
- Il est de type : `(new, replace, embed, other, none)`
  - `new` : la ressource d'arrivée est affichée dans une nouveau contexte de présentation (fenêtre, frame…)
  - `replace` : la ressource d'arrivée est affichée dans le même contexte de présentation,
  - `embed` : la ressource d'arrivée est affichée à la place de la ressource de départ.
  - `other` : un autre composant du lien (élément ou attribut) indique le mode de présentation.
  - `none` : l'application est maître du choix du mode de présentation

Elément « arc » : attribut `actuate`

- L'attribut arc définit quand est chargée la ressource d'arrivée :
- Il est de type : `(onLoad, onRequest, other, none)`
  - `onLoad` : l'accès à la ressource d'arrivée est réalisée au moment du chargement de la ressource distante ;
  - `onRequest` : l'accès à la ressource d'arrivée est commandée par un événement postérieur au chargement de la ressource de départ (un click souris, par exemple) ;
  - `other` : un autre composant du lien (élément ou attribut) indique le moment du chargement ;
  - `none` : l'application est maître du choix du moment du chargement.

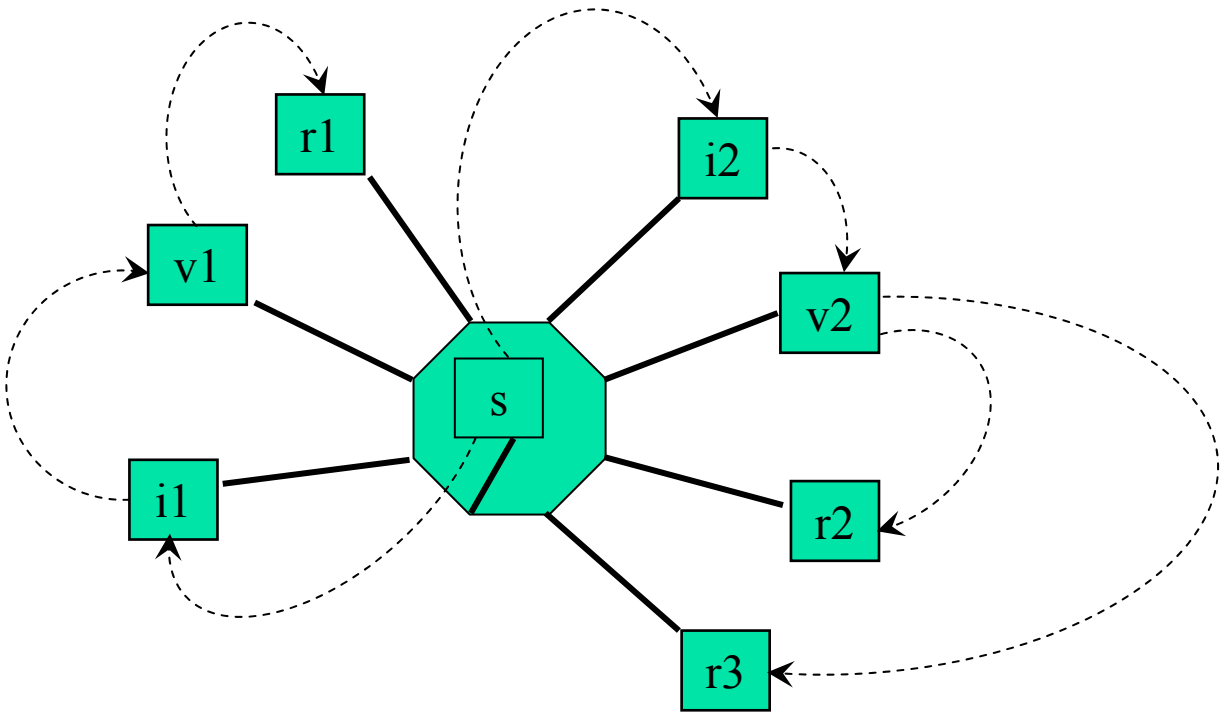
Attributs d'un élément de liaison

	type d'élément	type
href	locator	CDATA
role	simple, extended, locator, resource	CDATA
arcrole	simple, arc	CDATA
title	simple, extended, locator, arc, resource	CDATA
show	simple, arc	(new, replace, embed, other, none)
actuate	simple, arc	(onLoad, onRequest, other, none)
label	locator, resource	NMTOKEN
from	arc	NMTOKEN
to	arc	NMTOKEN

Exemple de lien étendu

- Supposons que l'on veuille construire des fiches dont chacune :
  - est associée à un sommet, le Mont Thabor, par exemple ;
  - décrit :
    - les itinéraires qui conduisent à ce sommet ;
    - le vallon dans lequel se déroulent un itinéraire, sur demande de l'utilisateur ;
    - le(s) refuge(s) situés au départ de ces vallons, également sur demande de l'utilisateur.
- Chacune de ces fiches peut être représentée par un lien étendu dont :
  - les ressources sont : le sommet, les itinéraires, les vallons et les refuges.
  - les arcs permettent l'accès :
    - aux itinéraires depuis un sommet,
    - au vallon depuis un itinéraire,
    - aux refuges depuis un vallon.

Exemple de lien étendu : ressources et arcs



s = Mt Thabor
i1 = itinéraire n°5 du vallon des Muandes
i2 = itinéraire n°5 du vallon du Mt Thabor
v1 = vallon des Muandes
v2 = vallon du Mt Thabor
r1 = refuge des Drayères
r2 = refuge I Re Magi
r3 = refuge Tre Alpini

Exemple de lien étendu : déclaration des éléments de liaison (1)

```
<!ELEMENT fiche-rando
   (sommet,
    itineraire+,
    vallon+,
    refuge*,
    vers-itineraire+,
    vers-vallon+,
    vers-refuge*)>
<!ATTLIST fiche-rando
   xmlns:xlink CDATA #FIXED "http://www.w3.org/1999/xlink"
   xlink:type (extended) #FIXED "extended">
<!ELEMENT sommet (#PCDATA)>
<!ATTLIST sommet
   xlink:type (resource) #FIXED "resource"
   xlink:label (s) #FIXED "s">
```

Exemple de lien étendu : déclaration des éléments de liaison (2)

```
<!ELEMENT itineraire EMPTY>
<!ATTLIST itineraire
   xlink:locator (locator) #FIXED "locator"
   xlink:href CDATA #REQUIRED
   xlink:label NMTOKEN #REQUIRED>
<!ELEMENT vallon EMPTY>
<!ATTLIST vallon
   xlink:locator (locator) #FIXED "locator"
   xlink:href CDATA #REQUIRED
   xlink:label NMTOKEN #REQUIRED>
<!ELEMENT refuge EMPTY>
<!ATTLIST refuge
   xlink:locator (locator) #FIXED "locator"
   xlink:href CDATA #REQUIRED
   xlink:label NMTOKEN #REQUIRED>
```

Exemple de lien étendu : déclaration des éléments de liaison (3)

```
<!ELEMENT vers-itineraire EMPTY>
<!ATTLIST vers-itineraire
   xlink:type (arc) #FIXED "arc"
   xlink:from (s) #FIXED "s"
   xlink:to NMTOKEN #REQUIRED
   xlink:show (new) #FIXED "new"
   xlink:actuate (onLoad) #FIXED "onLoad">
<!ELEMENT vers-vallon EMPTY>
<!ATTLIST vers-vallon
   xlink:type (arc) #FIXED "arc"
   xlink:from NMTOKEN #REQUIRED
   xlink:to NMTOKEN #REQUIRED
   xlink:show (embed) #FIXED "embed"
   xlink:actuate (onRequest) #FIXED "onLoad">
```

Exemple de lien étendu : déclaration des éléments de liaison (4)

```
<!ELEMENT vers-itineraire EMPTY>
<!ATTLIST vers-itineraire
   xlink:type (arc) #FIXED "arc"
   xlink:from NMTOKEN #REQUIRED
   xlink:to NMTOKEN #REQUIRED
   xlink:show (embed) #FIXED "embed"
   xlink:actuate (onLoad) #FIXED "onLoad">
```

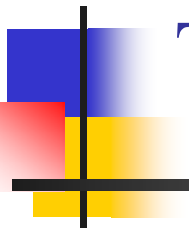
Exemple de liens étendus : fichiers contenant les ressources

- On suppose que :
  - Le fichier « claree.xml » contient la description en XML du guide « Itinéraires skieurs » telle que présentée dans ce cours.
  - Le fichier « refuges.xml » contient la description en XML des refuges utilisables pour ces itinéraires :

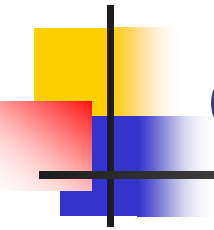
```
<refuges>
  …
  <refuge ="R8"> Refuge des Drayères</refuge>
  …
  <refuge ="R10"> Refuge I Re Magi</refuge>
  <refuge ="R11"> Refuge Tre Alpini</refuge>
<refuges>
```

Exemple de lien étendu : itinéraires du Mont Thabor

```
<fiche-rando>
   <sommet>Mont Thabor</sommet>
   <itineraire href="claree.xml#xpointer(id("I15.5"))" label="i1"/>
   <itineraire href="claree.xml#xpointer(id("I17.5"))" label="i2"/>
   <vallon href="claree.xml#xpointer(id("V15"))" label="v1"/>
   <vallon href="claree.xml#xpointer(id("V17"))" label="v2"/>
   <refuge href="refuges.xml#xpointer(id("R8"))" label="r1"/>
   <refuge href="refuges.xml#xpointer(id("R10"))" label="r2"/>
   <refuge href="refuges.xml#xpointer(id("R11"))" label="r3"/>
   <vers-itineraire from="s" to="i1"/>
   <vers-itineraire from="s" to="i2"/>
   <vers-vallon from="i1" to="v1"/>
   <vers-vallon from="i2" to="v2"/>
   <vers-refuge from="v1" to="r1"/>
   <vers-refuge from="v2" to="r2"/>
   <vers-refuge from="v2" to="r3"/>
</fiche-rando>
```



Transformation : le langage XSLT



Objectifs du langage XSLT

- XSLT permet de produire un nouveau document XML 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**.
- L'espace de noms associé au langage XSLT est :

  `http://www.w3.org/1999/XSL/Transform`

  dont le préfixe usuel est `xsl`.

Règle

- Une règle est décrite sous forme d'un élément XML :

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

où :

  - *p* est un chemin de localisation,
  - le contenu de la règle est une séquence de caractères ou d'éléments dont certains sont des instructions XSLT.
- D'autres attributs permettent :
  - de nommer une règle pour l'invoquer par ce nom,
  - de donner une priorité à une règle : un nombre d'autant plus grand que la règle a une priorité forte,
  - d'associer à une règle un mode sous lequel elle pourra être activée.

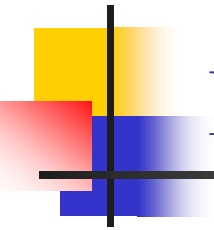
Instruction

- 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,
  - instructions de tri et de groupement,
  - définitions de variables,

  …



Moteur de transformation XSLT

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

Application des règles

- Pour chaque nœud de la liste des nœuds à traiter, appelé **nœud contexte** :
  - Chercher les règles `<xsl:template match="`*p*`">`*t*`</...>` telles que le nœud contexte est l'extrémité d'un chemin *p*.
  - S'il y en a plusieurs choisir la plus spécifique i. e. 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

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

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

  - règle prédéfinie **pour les nœuds texte et attribut**, produisant la recopie de leurs valeurs dans le flot de sortie :

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

Quelques instructions classiques

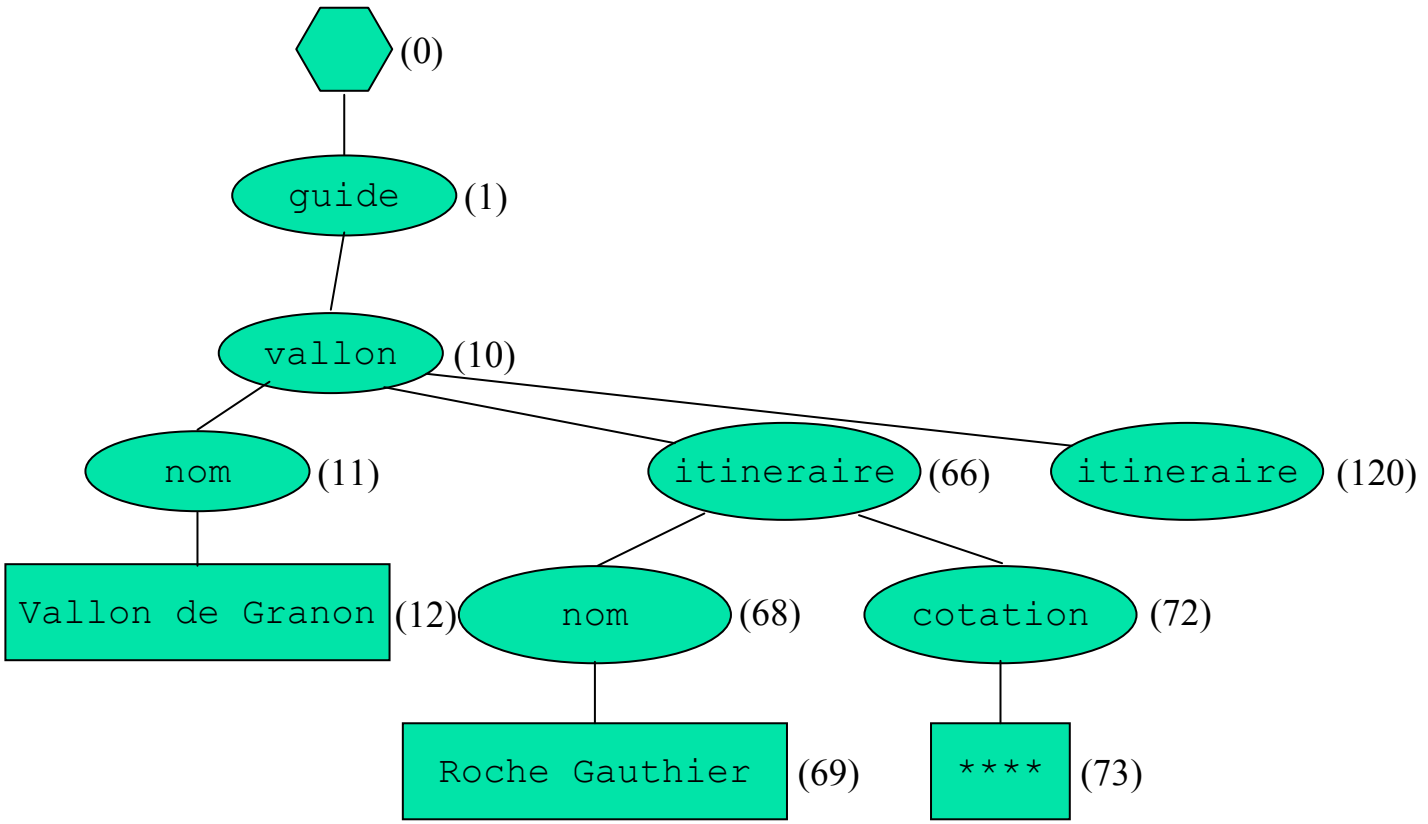
- `<xsl:apply-templates/>`
  - La liste des nœuds à traiter est constituée des nœuds fils du nœud contexte.
  - **Appliquer les règles**.
- `<xsl:apply-templates select="`*p*`"/>`
  - La liste des nœuds à traiter est constituée des nœuds atteints par le chemin *p* depuis le nœud contexte.
  - **Appliquer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

Exemple 1 : Itinéraires les plus difficiles

- On veut produire un document XML listant les noms des itinéraires **** avec pour chacun :
  - son nom,
  - le nom du vallon dans lequel il se déroule.
- Le document produit aura la forme suivante :

```
<best-of>
  <nom>Roche Gauthier (Vallon de Granon)</nom>
  …
</best-of>
```

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



Exemple 1 : la feuille de style

```
1.  <xsl:stylesheet version="1.0" xmlns:xsl="...">
2.   <xsl:output method="xml"/>
3.   <xsl:template match="/">
4.     <best-of>
5.       <apply-templates select="itineraire[cotation='****']/>
6.     </best-of>
7.   </xsl:template>
8.   <xsl:template match="itineraire">
9.     <nom>
10.      <xsl:value-of select="nom"/>
11.        (
12.      <xsl:value-of select="ancestor::vallon/nom"/>
13.      )
14.    </nom>
15.  </xsl:template>
16. </xsl:stylesheet>
```

On produit un document XML.

Exemple 1 : la transformation

- La liste des nœuds à traiter est [0].
  - Le nœud contexte est le nœud 0. La règle 2 s'y applique.
    - On écrit `<best-of>` dans le flot de sortie (ligne 3).
    - On applique les règles : la liste des nœuds à traiter est [66, 120, …]
      - Le nœud contexte est le nœud 64. La règle 7 s'y applique.
        - On écrit `<nom>` dans le flot de sortie (ligne 8).
        - On exécute l'instruction 9 : on écrit `Roche Gauthier` dans le flot de sortie.
        - On écrit `(` dans le flot de sortie (ligne 10).
        - On exécute l'instruction 11 : on écrit `Vallon de Granon` dans le flot de sortie.
        - On écrit) dans le flot de sortie (ligne 12).
        - On écrit `</nom>` dans le flot de sortie (ligne 13).
      - Le nœud contexte est le nœud 120…
    - On écrit `</best-of>` dans le flot de sortie.

Exemple 1 : le résultat

```
<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>
   <nom>Roche Gauthier (Vallon du Creuset)</nom>
   <nom>Aiguille rouge (Vallon des Thures)</nom>
   <nom>Le Sommet Rond (Vallon des Thures)</nom>
   ...
</best-of>
```

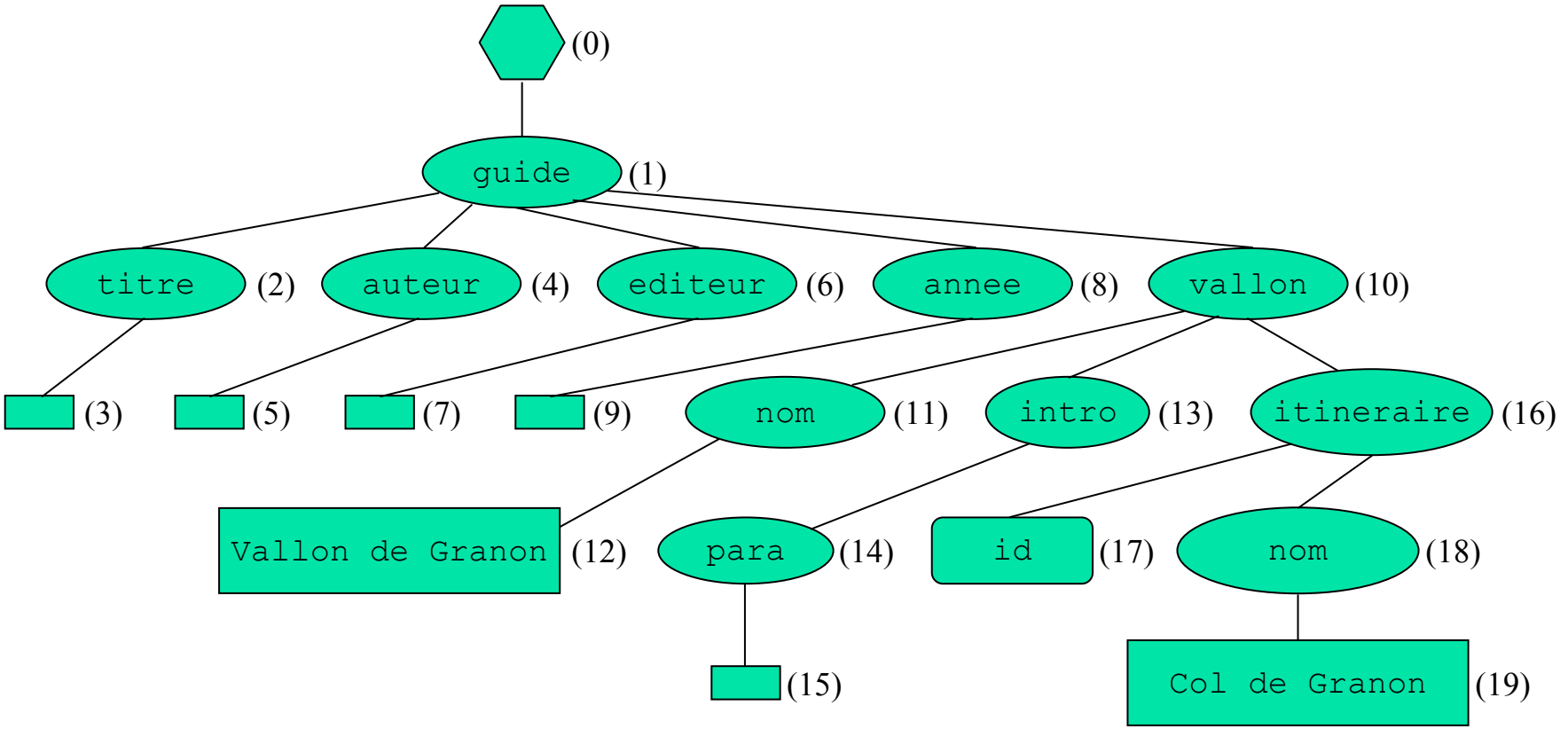
Exemple 2 : Liste des vallons et des itinéraires

- On veut produire un document XML listant les noms des vallons et de leurs itinéraires, ayant la forme suivante :

```
<vallons>
    <nom>Vallon du Granon</vallon>
    <itineraires>
      <nom>Col du Granon</nom>
      …
    </itineraires>
    …
</vallons>
```

- Cet exemple illustre l'emploi de la règle prédéfinie sur les nœuds racine et élément afin de parcourir récursivement les nœuds jusqu'à en atteindre un auquel s'applique une règle définie.

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



Exemple 2 : la feuille de style

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

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

Exemple 2 : la transformation

- La règle 1 s'applique sur le nœud 0 :
  - On écrit `<vallons>` dans le flot de sortie.
  - La règle prédéfinie sur les nœuds racine et élément s'applique sur les nœuds 2, 4, 6 et 8 et la règle 16 s'applique sur les nœuds 3, 5, 7 et 9.
  - La règle 7 s'applique sur le nœud 10 :
    - On écrit `<nom>Vallon de Granon</nom>` dans le flot de sortie.
    - On écrit `<itineraires>` dans le flot de sortie :
      - La règle prédéfinie sur les nœuds racine et élément s'applique récursivement sur les nœuds 13 et 14 et la règle 16 s'applique sur les nœuds 15 et 17.
      - La règle 13 s'applique sur le nœud 16 : on écrit `<nom>Col de Granon</nom>` dans le flot de sortie.

      …
    - On écrit `</itineraires>` dans le flot de sortie.

    …
  - On écrit `</vallons>` dans le flot de sortie.

Exemple 2 : le résultat

```
<vallons>
  <vallon>
    <nom>Vallon de Granon</nom>
    <itineraires>
      <nom>Col de Granon</nom>
      <nom>Croix de la Gardiole</nom>
      <nom>Col de l'Oule</nom>
      <nom>Rocher du Loup</nom>
      <nom>Roche Gauthier</nom>
      <nom>Col de Lenlon</nom>
    </itineraires>
  </vallon>
  <vallon>
    <nom>Vallon des Acles</nom>
    <itineraires>
      <nom>Le Guyon</nom>
      <nom>Le col des Acles</nom>
    …
</vallons>
```