R4.02 - Qualité logicielle - CM n°2

Pierre Nerzic

février-mars 2024

Ce cours présente la norme XML pour représenter des informations.

Pourquoi ?

De nombreux outils liés à la qualité logicielle utilisent XML, et c'est aussi un format de stockage et d'échange de données qui propose plusieurs concepts à connaître :

- Représentation d'informations : données, configurations, etc.
- Validation par un schéma
- Interrogation avec XPath
- Transformation en un autre schéma (pas dans ce cours)

Les formats JSON et YAML sont des alternatives plus récentes mais moins polyvalentes.

Pierre Nerzic

Présentation rapide de la norme XML

Pierre Nerzic

XML ?

Un fichier XML représente des informations structurées :

```
<?xml version="1.0" encoding="utf-8"?>
<!-- itinéraire fictif -->
<itineraire>
  <etape distance="0km">départ</etape>
  <etape distance="13km">tourner à droite</etape>
  <etape distance="22km">arrivée</etape>
</itineraire>
```

Cet exemple modélise un itinéraire composé d'étapes.

XML = *Extensible Markup Language*. Il permet de choisir la représentation des données sans aucune contrainte. On choisit les balises et les attributs comme on le souhaite et on construit la structure qu'on veut.

Arborescence d'éléments

Un document XML est composé de plusieurs parties :

- Entête de document précisant la version et l'encodage,
- Des règles optionnelles pour vérifier si le document est valide,
- Un arbre d'*éléments* basé sur un élément appelé *racine*
  - Un *élément* possède un *nom*, des *attributs* et un *contenu*
  - Le contenu d'un élément peut être :
    - rien : élément vide noté `<nom attributs.../>`
    - du texte
    - d'autres éléments (les éléments enfants).
  - Un élément non vide est délimité par une *balise ouvrante* et une *balise fermante*.
    - une balise ouvrante est notée `<nom attributs...>`
    - une balise fermante est notée `</nom>`

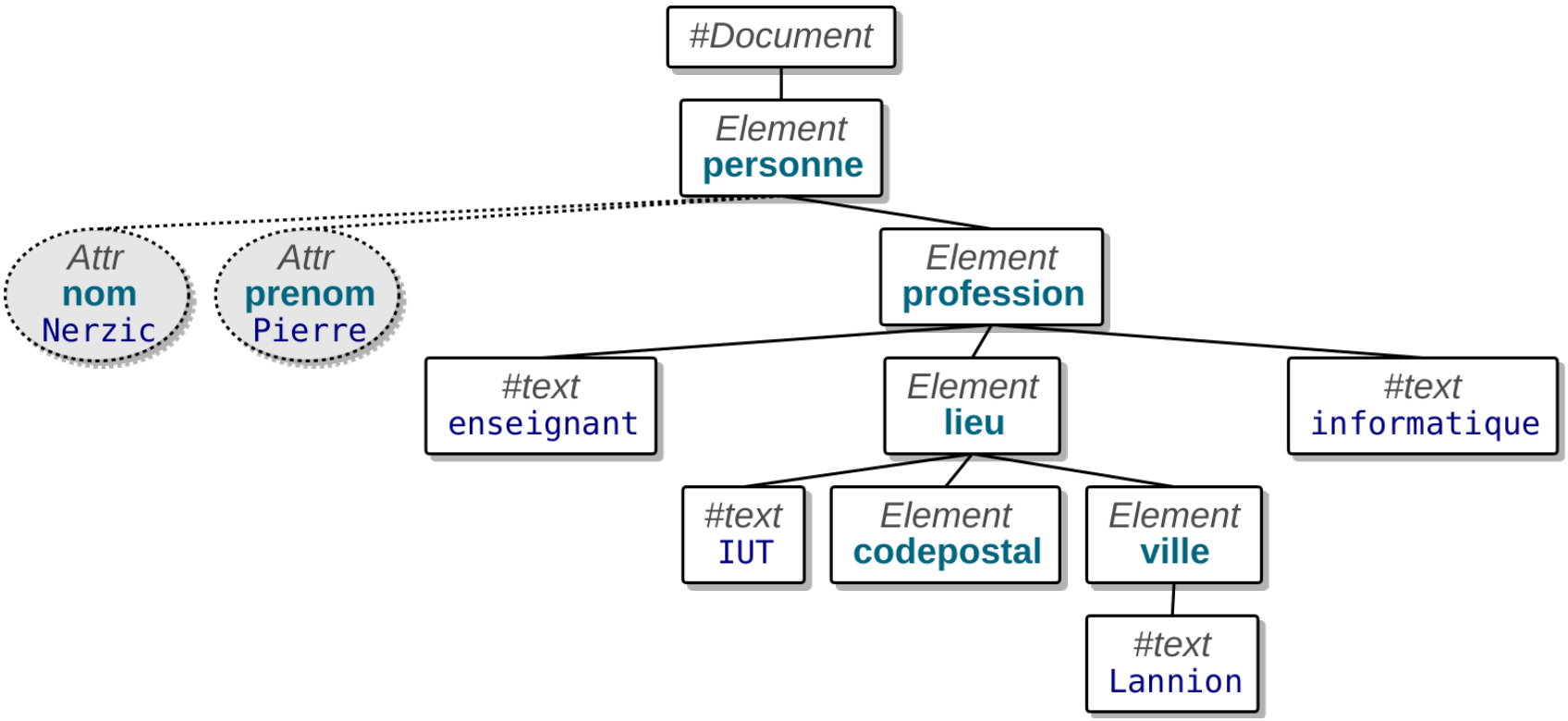
Exemple complet

Voici un document XML représentant une personne :

```
<?xml version="1.0" encoding="utf-8"?>
<personne nom="Nerzic" prenom="Pierre">
  <profession>
    enseignant
    <lieu>
      IUT
      <codepostal/>
      <ville>Lannion</ville>
    </lieu>
    informatique
  </profession>
</personne>
```

Les textes peuvent être délimités par des balises, ou non.

Représentation graphique



Pierre Nerzic

Explications

En interne, le document XML est représenté par un arbre composé de plusieurs types de nœuds (*node* en anglais) :

nœuds éléments ils sont associés aux balises `<element>`. Ce sont des nœuds qui peuvent avoir des enfants en dessous.

nœuds `#text` ils représentent le texte situé entre deux balises. Les nœuds texte sont des feuilles dans l'arbre.

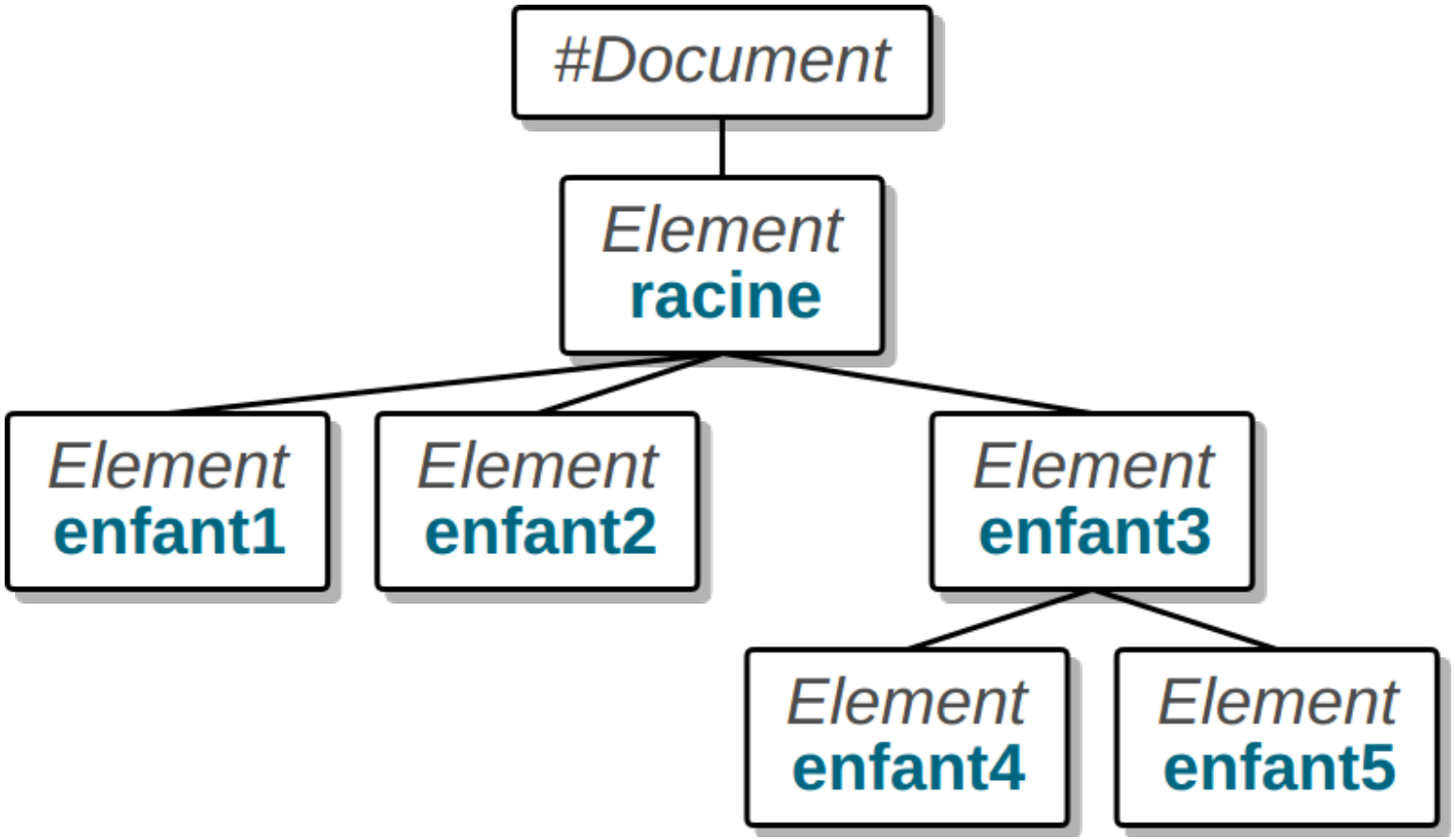
nœuds attributs ils contiennent les attributs des éléments

Notez que différents textes peuvent être entrelacés avec des éléments. Voir le cas de `<lieu>` dans le contenu de `<profession>`. Il est possible de le faire, mais ce n'est pas forcément souhaitable.

D'autres types de nœuds existent : commentaires, données brutes...

Vocabulaire

Soit cet arbre XML :



Pierre Nerzic

Vocabulaire (suite)

Voici comment on désigne les différents nœuds les uns par rapport aux autres :

- `<racine>` est le nœud **parent** du nœud **enfant** (*child*) `<enfant3>`, lui-même parent de `<enfant4>` et `<enfant5>`,
- `<racine>`, `<enfant3>` sont des nœuds **ancêtres** (*ancestors*) de `<enfant4>` et `<enfant5>`,
- `<enfant4>` et `<enfant5>` sont des **descendants** (*descendants*) de `<racine>` et `<enfant3>`,
- `<enfant1>` est un nœud **frère** (*sibling* = fratrie) de `<enfant2>` et réciproquement.

Attributs

Les attributs caractérisent un élément. Ce sont des couples `nom="valeur"` ou `nom='valeur'`. Ils sont placés dans la balise ouvrante.

```
<?xml version="1.0" encoding="utf-8"?>
<meuble id="765" type='table'>
  <prix monnaie='€'>74,99</prix>
  <dimensions unites="cm" longueur="120" largeur="80"/>
  <description langue='fr'>Belle petite table</description>
</meuble>
```

Remarques :

- Il n'y a pas d'ordre entre les attributs d'un élément,
- Un même attribut ne peut être présent qu'une seule fois.

Texte

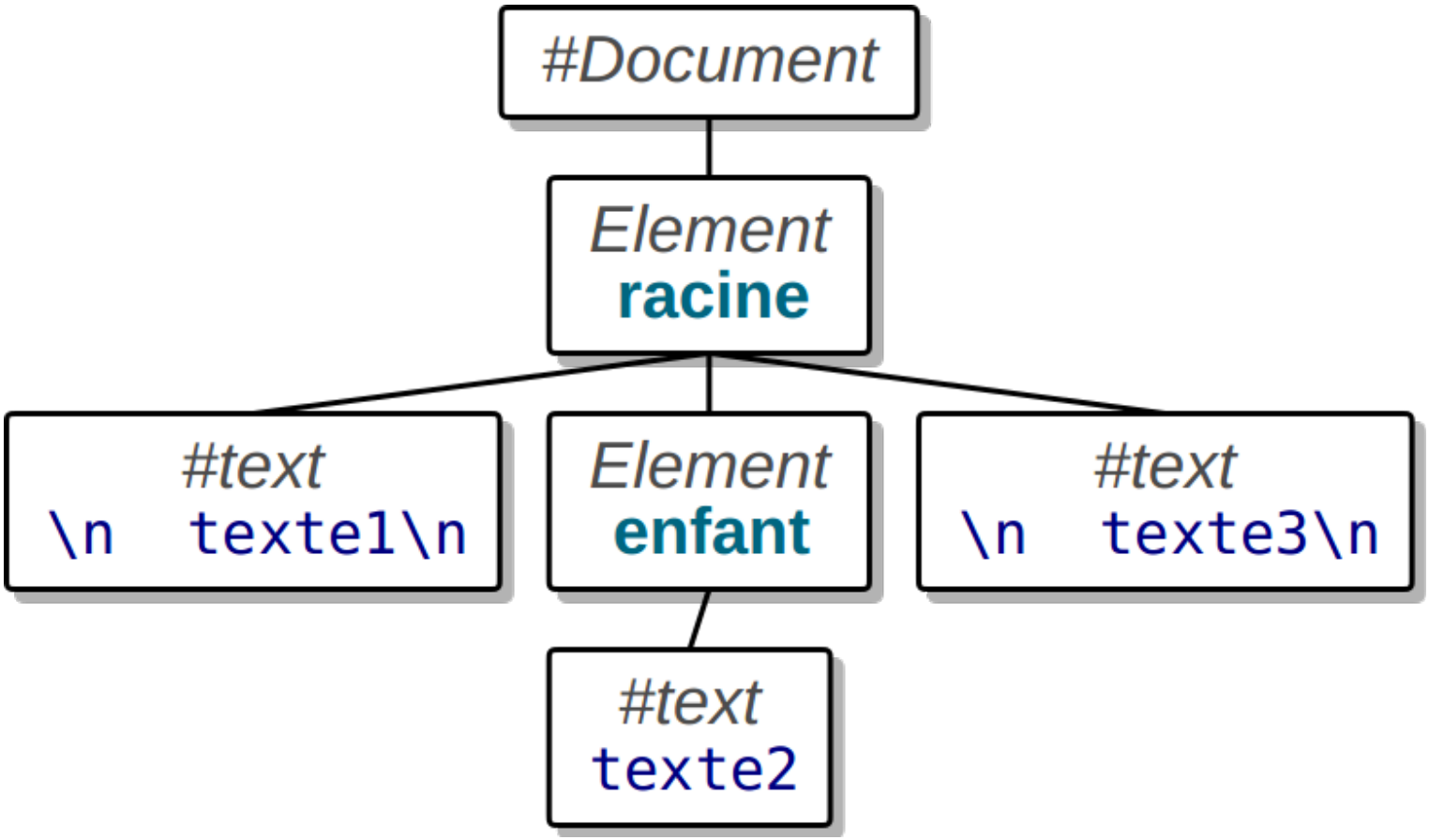
Les textes font partie du contenu des éléments et sont vus comme des nœuds enfants.

```
<?xml version="1.0" encoding="utf-8"?>
<racine>
  texte1
  <enfant>texte2</enfant>
  texte3
</racine>
```

Il faut bien comprendre que tous les fragments de texte situés entre les balises, y compris les espaces et les retours à la ligne sont considérés comme faisant partie du même texte. Dans un programme, il faut penser à nettoyer les textes extraits.

Arbre correspondant

Voici l'arbre correspondant à l'exemple précédent. Notez les retours à la ligne et espaces présents dans les textes sauf `texte2`.



Pierre Nerzic

Noms des éléments

```
<?xml version="1.0" encoding="utf-8"?>
<élément_avec_un_nom_compliqué>
    <iut:département>INFO</iut:département>
</élément_avec_un_nom_compliqué>
```

Les noms des éléments peuvent employer de nombreux caractères Unicode (correspondant au codage déclaré dans le prologue) mais pas les signes de ponctuation.

Le caractère « : » permet de séparer le nom en deux parties, préfixe et *nom local*. Le tout s'appelle *nom qualifié*. Par exemple `iut:département` est un nom qualifié préfixé par `iut`.

$$\textit{nom qualifié} = \textit{préfixe} : \textit{nom local}$$

Le préfixe permet de définir un *espace de nommage* (*namespace*).

Espaces de nommage

Un espace de nommage définit une famille de noms afin d'éviter les confusions entre des éléments qui auraient le même nom mais pas le même sens. Cela arrive quand le document XML modélise les informations de plusieurs domaines.

Voici un exemple de document qui modélise une table (avec 4 pieds) et aussi un tableau HTML pour afficher ses dimensions. On voit la confusion entre les deux éléments `<table>`.

```
<meuble id="765">
  <table prix="74,99€">acajou</table>
  <table border="1">
    <tr><th>longueur</th><th>largeur</th></tr>
    <tr><td>120cm</td><td>80cm</td></tr>
  </table>
</meuble>
```


Pierre Nerzic

Définition d'un espace de nommage

On doit choisir un URI (désignation d'une ressource internet), un URL ou un URN, pour identifier l'espace de nommage.

Un URL a cette syntaxe :

`schéma://[user:passwd@]hôte[:port]][/]chemin[?requête][#fragment]`,

par exemple

`http://en.wikipedia.org/wiki/Uniform_Resource_Identifier#Syntax`.

Les URN sont au format `urn:NID:NSS`, ex: `urn:iutlan:xmlsem1`

Ensuite on rajoute un attribut spécial à la racine du document :

```
<?xml version="1.0" encoding="utf-8"?>
<racine xmlns:PREFIXE="URI">
    <PREFIXE:balise>...</PREFIXE:balise>
</racine>
```

Exemple revu

Voici l'exemple précédent, avec deux *namespaces*, un URN pour les meubles et un URL pour HTML :

```
<?xml version="1.0" encoding="utf-8"?>
<meuble:meuble id="765"
        xmlns:meuble="urn:iutlan:meubles"
        xmlns:html="http://www.w3.org">
  <meuble:table prix="74,99€">acajou</meuble:table>
  <html:table border="1">
    <html:tr><html:th>longueur</html:th>...</html:tr>
    <html:tr><html:td>120cm</html:td>...</html:tr>
  </html:table>
</meuble:meuble>
```

Notez le préfixe également appliqué à la racine du document.

Pierre Nerzic

Namespace par défaut

Lorsqu'un élément définit un attribut `xmlns="URI"`, alors lui-même et ses descendants sont placés dans ce namespace, sans préfixe.

```
<?xml version="1.0" encoding="utf-8"?>
<book xmlns="http://docbook.org/ns/docbook">
   <title>Livre très simple</title>
</book>
```

Autre exemple :

```
<meuble id="765" xmlns="urn:iutlan:meubles">
  <table prix="74,99€">acajou</table>
  <table border="1" xmlns="http://www.w3.org">
    <tr><th>longueur</th>...</tr>
    <tr><td>120cm</td>...</tr>
  </table>
</meuble>
```

Namespace par défaut, attributs et valeurs

Définir un *namespace* par défaut associe seulement les éléments à ce *namespace*, mais pas leurs attributs.

Les attributs qui n'ont pas de préfixe n'ont pas de *namespace*. L'interprétation des attributs dépend de l'élément dans lequel ils se trouvent. Voir cette réponse sur StackOverflow.

Cependant, pour clarifier la situation, on rajoute de préférence un préfixe devant les attributs. Par exemple, `android:layout_width`. Avec le préfixe, il n'y a plus d'ambiguïté possible : l'attribut appartient au *namespace* concerné.

On retrouve le même problème avec les valeurs de certains attributs, voir les schémas XML plus loin.

Pierre Nerzic

Validité d'un document

Introduction

Il y a deux niveaux de correction pour un document XML :

- Un document XML **bien formé** (*well formed*) respecte les règles syntaxiques d'écriture XML : écriture des balises, imbrication des éléments, entités, etc. C'est la plus facile des vérifications.
- Un document **valide** respecte des règles supplémentaires sur les noms, les attributs et l'organisation des éléments.

La validation est cruciale pour une entreprise, p.ex. une banque, qui gère des transactions représentées en XML. S'il y a des erreurs dans les documents, cela peut compromettre l'entreprise. Il vaut mieux être capable de refuser un document invalide plutôt qu'essayer de le traiter et pâtir des erreurs qu'il contient.

Processus de validation

D'abord, il faut disposer d'un fichier contenant des règles. Il existe plusieurs langages pour faire cela : DTD, Schemas XML, RelaxNG et Schematron. Ces langages modélisent des règles de validité plus ou moins précises et d'une manière plus ou moins lisible.

On ne verra pas les DTD, des antiquités, mais les schémas XML.

Chaque document XML est comparé à ce fichier de règles, à l'aide d'un outil de validation : `xmlstarlet`, `xmllint`...

La validation indique :

- soit le document est valide, conforme aux règles,
- soit il contient des erreurs comme : tel attribut de tel élément contient une valeur interdite par telle contrainte, il manque tel sous-élément ou attribut dans tel élément, etc.

Schémas XML

Les Schémas XML sont une norme W3C pour spécifier le contenu d'un document XML.

Pour se faire une idée, voici un exemple de schéma :

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="reference" type="ElemReference" />
  <xs:complexType name="ElemReference">
    <xs:sequence>
      <xs:element name="titre" type="xs:string" />
      <xs:element name="auteur" type="xs:string" />
      <xs:element name="ISBN" type="xs:string" />
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```


Pierre Nerzic

Validation d'un document XML par un schéma

Quand on possède un schéma (un fichier `.xsd`) et un document XML, on peut valider le document par une de ces commandes Unix :

- `xmllint --schema schema.xsd --noout document.xml`
- `xmlstarlet val --xsd schema.xsd -e document.xml`

Soit ça passe, soit il y a une erreur de validation : le document ne respecte pas le schéma.

C'est un peu le même principe qu'un test unitaire, mais cela concerne la structure et le contenu d'un document XML.

Principes généraux des Schémas XML

Un schéma permet de définir des éléments, leurs attributs et leurs contenus, avec une notion de typage forte.

Avec un schéma, il faut définir des **types de données** :

- la nature des données : chaîne, nombre, date, etc.
- les contraintes qui portent dessus : domaine de définition, valeurs possibles, expression régulière, etc.

Et avec ces types, on définit les éléments :

- noms et types des attributs
- sous-éléments possibles avec leurs répétitions, les alternatives...

Pierre Nerzic

Structure générale d'un schéma

Un schéma est contenu dans un arbre XML de racine <schema>. Le contenu du schéma définit les éléments qu'on peut trouver dans le document. Voici un squelette de schéma :

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="itineraire" type="ElemItineraire" />
  ... définition du type ElemItineraire ...
</xs:schema>
```

Ce schéma valide le document partiel suivant :

```
<?xml version="1.0" encoding="utf-8"?>
<itineraire>
    ...
</itineraire>
```


Pierre Nerzic

Définition d'éléments

Un élément <*nom*>*contenu*</*nom*> du document est défini par un élément `<xs:element name="`*nom*`" type="`*TypeContenu*`">` dans le schéma.

Dans l'exemple suivant, le type est `xs:string`, un texte quelconque, donc l'élément peut/doit contenir du texte :

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="message" type="xs:string"/>
</xs:schema>
```

Ce schéma valide le document suivant :

```
<?xml version="1.0" encoding="utf-8"?>
<message>Tout va bien !</message>
```


Pierre Nerzic

Définition de types de données

L'exemple précédent indique que l'élément `<message>` doit avoir un contenu de type `xs:string`, c'est à dire du texte. Ce type est un « type simple ». Il y a de nombreux types simples prédéfinis, dont :

- chaîne :
  - `xs:string` est le type le plus général
  - `xs:token` vérifie que c'est une chaîne nettoyée des sauts de lignes et espaces d'indentation
- date et heure :
  - `xs:date` correspond à une chaîne au format `AAAA-MM-JJ`
  - `xs:time` correspond à `HH:MM:SS.s`
  - `xs:datetime` valide `AAAA-MM-JJTHH:MM:SS`, on doit mettre un `T` entre la date et l'heure.

Types de données (suite)

- nombres :
  - `xs:float`, `xs:decimal` valident des nombres réels
  - `xs:integer` valide des entiers
  - il y a de nombreuses variantes comme `xs:nonNegativeInteger`, `xs:positiveInteger`...
- autres :
  - `xs:ID` pour une chaîne identifiante, `xs:IDREF` pour une référence à une telle chaîne
  - `xs:boolean` permet de n'accepter que `true`, `false`, `1` et `0` comme valeurs dans le document.
  - `xs:base64Binary` et `xs:hexBinary` pour des données binaires.
  - `xs:anyURI` pour valider des URI (URL ou URN).

Restrictions sur les types

Lorsque les types ne sont pas suffisamment contraints et risquent de laisser passer des données fausses, on peut rajouter des contraintes. Elles sont appelées *facettes* (*facets*).

Dans ce cas, on doit définir un type `simpleType` et lui ajouter des restrictions, comme dans cet exemple :

```
<xs:element name="temperature" type="TypeTemperature" />

<xs:simpleType name="TypeTemperature">
  <xs:restriction base="xs:decimal">
    <xs:minInclusive value="-30"/>
    <xs:maxInclusive value="+40.0"/>
  </xs:restriction>
</xs:simpleType>
```


Pierre Nerzic

Définition de restrictions

La structure d'une restriction est :

```
<xs:restriction base="type de base">
    <xs:CONTRAINTE value="PARAMETRE"/>
    ...
</xs:restriction>
```

Par exemple, un type « numéro de sécurité sociale » :

```
<xs:simpleType name="TypeNumeroSecu">
  <xs:restriction base="xs:string">
    <xs:whiteSpace value="collapse"/>
    <xs:pattern value="[12][0-9]{12}( [0-9]{2})?"/>
  </xs:restriction>
</xs:simpleType>
```

Les contraintes qu'on peut mettre dépendent du type de données.

Restriction sur `string`

Ces restrictions (*facettes*) s'appliquent au type `string` :

- longueur : `length`, `maxLength`, `minLength`. Ces contraintes vérifient que la valeur possède la bonne longueur.
- `whiteSpace` indique ce qu'on doit faire avec les caractères espaces, tabulations et retours à la ligne éventuellement présents dans les données, pour vérifier les facettes :
  - `value="preserve"` : on les garde tels quels
  - `value="replace"` : on les remplace par des espaces
  - `value="collapse"` : on les supprime tous (à utiliser avec un motif).

Restrictions sur `string` (suite)

- expression régulière étendue (egrep) avec `pattern` :

```
<xs:simpleType name="TypeTemperature">
  <xs:restriction base="xs:string">
    <xs:pattern value="[-+]?[1-9][0-9]?°C"/>
    <xs:whiteSpace value="collapse"/>
  </xs:restriction>
</xs:simpleType>
```

- liste des valeurs possibles avec `enumeration` :

```
<xs:simpleType name="TypeFreinsVélo">
  <xs:restriction base="xs:string">
    <xs:enumeration value="disque"/>
    <xs:enumeration value="patins"/>
    <xs:enumeration value="rétropédalage"/>
  </xs:restriction>
</xs:simpleType>
```

Restrictions sur les dates et nombres

Les dates et nombres possèdent quelques contraintes sur la valeur exprimée :

- bornes inférieure et supérieure :
  - `minExclusive` et `minInclusive`
  - `maxExclusive` et `maxInclusive`
- nombre de chiffres :
  - `totalDigits` : vérifie le nombre de chiffres total (partie entière et fractionnaire, sans compter le point décimal)
  - `fractionDigits` : vérifie le nombre de chiffres dans la partie fractionnaire.
- Les facettes `pattern` et `enumeration` sont utilisables aussi.

Types à alternatives

Comment valider une donnée qui pourrait être de plusieurs types possibles, par exemple, valider les deux premiers éléments et refuser le troisième :

```
<couleur>Chartreuse</couleur>
<couleur>#7FFF00</couleur>
<couleur>02 96 46 93 00</couleur>
```

Si on déclare l'élément <couleur> comme contenant n'importe quelle valeur chaîne, comme ceci :

```
<xs:element name="couleur" type="xs:string"/>
```

on ne pourra rien vérifier.

Types à alternatives (suite)

Alors on crée un « type à alternatives » qui est équivalent à plusieurs possibilités. Attention, ce n'est pas comme déclarer une énumération de *valeurs possibles*. Ici, on parle de *types possibles*.

Pour exprimer qu'un type peut correspondre à plusieurs autres types, il faut le définir en tant que `<union>` et mettre les différents types possibles dans l'attribut `memberTypes` :

```
<xs:simpleType name="TYPE_ALTERNATIF">
  <xs:union memberTypes="TYPE1 TYPE2 ..."/>
</xs:simpleType>
```

Les types possibles sont séparés par un espace.

Exemple de type à alternatives

Voici un exemple pour les couleurs :

```
<xs:simpleType name="TypeCouleurs">
  <xs:union memberTypes="TypeCouleursNom TypeCouleursHex"/>
</xs:simpleType>

<xs:simpleType name="TypeCouleursNom">
  <xs:restriction base="xs:string">
    <xs:pattern value="([A-Z][a-z]+)+"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="TypeCouleursHex">
  <xs:restriction base="xs:string">
    <xs:pattern value="#[0-9A-F]{6}"/>
  </xs:restriction>
</xs:simpleType>
```

Contenu d'éléments

On revient maintenant sur les éléments. Nous avons vu comment définir un élément contenant un texte, un nombre, etc. :

```
<xs:element name="NOM" type="TYPE"/>
```

Ça définit une balise <NOM> pouvant contenir des données du type indiqué par TYPE :

```
<?xml version="1.0" encoding="utf-8"?>
<NOM>données correspondant à TYPE</NOM>
```

Comment définir un élément dont le contenu peut être d'autres éléments, ainsi que des attributs ? En fait, c'est la même chose que les simpleType, sauf que le type est « complexe ». Un type complexe peut contenir des sous-éléments et des attributs.

Exemple de type complexe

Pour modéliser un élément <personne> ayant deux éléments enfants <prénom> et <nom>, il suffit d'écrire ceci :

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="personne" type="ElemPersonne"/>
  <xs:complexType name="ElemPersonne">
    <xs:sequence>
      <xs:element name="prénom" type="xs:string"/>
      <xs:element name="nom" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

La structure <xs:sequence> contient une liste d'éléments qui doivent se trouver dans le document à valider, dans l'ordre.

Contenu d'un type complexe

Un <xs:complexType> peut contenir trois sortes d'enfants :

```
<xs:complexType name="ElemPersonne">
  <xs:sequence> ou <xs:choice> ou <xs:all>...
</xs:complexType>
```

Les enfants peuvent être :

- <xs:sequence>éléments...</xs:sequence> : ces éléments doivent arriver dans cet ordre
- <xs:choice>éléments...</xs:choice> : le document à valider doit contenir l'un des éléments
- <xs:all>éléments...</xs:all> : le document à valider doit contenir ces éléments une et une seule fois chacun, et dans n'importe quel ordre.

Exemple de choix

Pour représenter une limite temporelle, par exemple la date de fin d'une garantie, soit on mettra un élément <date_fin> soit un élément <durée> :

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="limite" type="ElemLimiteTemps"/>
  <xs:complexType name="ElemLimiteTemps">
    <xs:choice>
      <xs:element name="date_fin" type="xs:date"/>
      <xs:element name="durée" type="xs:positiveInteger"/>
    </xs:choice>
  </xs:complexType>
</xs:schema>
```


Pierre Nerzic

Exemple avec all

Ici les éléments <nom>, <prénom> peuvent être dans n'importe quel ordre :

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="personne" type="ElemPersonne"/>
  <xs:complexType name="ElemPersonne">
    <xs:all>
      <xs:element name="prénom" type="xs:string"/>
      <xs:element name="nom"    type="xs:string"/>
    </xs:all>
  </xs:complexType>
</xs:schema>
```


Pierre Nerzic

Imbrication de structures

On peut imbriquer plusieurs structures pour définir des éléments à suivre et en option :

```
<xs:complexType name="ElemPersonne">
  <xs:sequence>
    <xs:element name="prénom" type="xs:string"/>
    <xs:element name="nom" type="xs:string"/>
    <xs:choice>
      <xs:element name="age" type="xs:string"/>
      <xs:element name="date_naiss" type="xs:date"/>
    </xs:choice>
  </xs:sequence>
</xs:complexType>
```

Par contre, on ne peut pas faire de mélange avec `<xs:all>`.

Nombre de répétitions

Dans le cas des structures `<xs:sequence>` et `<xs:all>`, il est possible de spécifier un nombre de répétition pour chaque sous-élément.

```
<xs:complexType name="ElemPersonne">
  <xs:sequence>
    <xs:element name="prénom" type="xs:string"
                minOccurs="1" maxOccurs="2"/>
    <xs:element name="nom" type="xs:string"
                minOccurs="0" maxOccurs="1"/>
  </xs:sequence>
</xs:complexType>
```

Par défaut, les nombres de répétitions min et max sont 1. Pour enlever une limite sur le nombre maximal, il faut écrire `maxOccurs="unbounded"`.

Définition d'attributs

Les attributs se déclarent dans un <complexType> :

```
<xs:complexType name="ElemPersonne">
    ...
    <attribute name="NOM" type="TYPE" OPTIONS/>
</xs:complexType>
```

nom	le nom de l'attribut
type	le type de l'attribut, ex: `string` pour un attribut quelconque
options	mettre `use="required"` si l'attribut est obligatoire, mettre `default="valeur"` s'il y a une valeur par défaut.

Cas spéciaux

Plusieurs situations sont assez particulières et peuvent sembler très compliquées :

- éléments vides sans ou avec attributs
- éléments textes sans ou avec attributs
- éléments avec enfants sans ou avec attributs
- éléments avec textes et enfants sans ou avec attributs

Voici comment elles sont modélisées avec des Schémas XML.

Élément vide sans attribut

C'est le cas le plus simple :

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="test" type="ElemTest"/>

  <xs:complexType name="ElemTest"/>

</xs:schema>
```

Appliqué au document entier, ça permet de ne valider que :

```
<?xml version="1.0" encoding="utf-8"?>
<test/>
```

Élément vide avec attribut

On rajoute un attribut obligatoire :

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="test" type="ElemTest"/>

  <xs:complexType name="ElemTest">
    <attribute name="att" type="xs:string" use="required"/>
  </xs:complexType>

</xs:schema>
```

Appliqué au document entier, ça permet de ne valider que :

```
<?xml version="1.0" encoding="utf-8"?>
<test att="ok"/>
```

Élément texte sans attribut

Il suffit de définir l'élément en tant que `simpleType` avec un `<xs:restriction>` pour définir son contenu :

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="test" type="ElemTest"/>

  <xs:simpleType name="ElemTest">
    <xs:restriction base="xs:integer"/>
  </xs:simpleType>
</xs:schema>
```

Appliqué au document entier, ça permet de ne valider que :

```
<?xml version="1.0" encoding="utf-8"?>
<test>123</test>
```

Élément texte avec attribut

On doit faire ainsi :

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="test" type="ElemTest"/>
  <xs:complexType name="ElemTest">
    <xs:simpleContent>
      <xs:extension base="xs:integer">
        <xs:attribute name="att" type="xs:string"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:schema>
```

Appliqué au document entier, ça permet de ne valider que :

```
<?xml version="1.0" encoding="utf-8"?>
<test att="ok">456</test>
```


Pierre Nerzic

Éléments enfants sans attribut

C'est comme précédemment, par exemple une séquence : 

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="test" type="ElemTest"/>
  <xs:complexType name="ElemTest">
    <xs:sequence>
      <xs:element name="test1"/>
      <xs:element name="test2" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Appliqué au document entier, ça permet de ne valider que : 

```
<?xml version="1.0" encoding="utf-8"?>
<test><test1/><test2>texte</test2></test>
```


Éléments enfants avec attribut

Pour valider des attributs sur l'élément parent :

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="test" type="ElemTest"/>
  <xs:complexType name="ElemTest">
    <xs:sequence>
      <xs:element name="test1"/>
      <xs:element name="test2" type="xs:string"/>
    </xs:sequence>
    <attribute name="att" type="xs:string"/>
  </xs:complexType>
</xs:schema>
```

Appliqué au document entier, ça permet de ne valider que :

```
<?xml version="1.0" encoding="utf-8"?>
<test att="ok"><test1/><test2>texte</test2></test>
```


Pierre Nerzic

Éléments enfants avec texte mélangé

Rajouter l'attribut `mixed="true"` à `<complexType>` :

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="test" type="ElemTest"/>
  <xs:complexType name="ElemTest" mixed="true">
    <xs:sequence>
      <xs:element name="test1"/>
      <xs:element name="test2" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Appliqué au document entier, ça permet de ne valider que :

```
<?xml version="1.0" encoding="utf-8"?>
<test>texte<test1/>texte<test2>texte2</test2>texte</test>
```


Pierre Nerzic

XPath

Présentation rapide

XPath est un mécanisme (syntaxe + fonctions) permettant d'extraire des informations d'un document XML. Par exemple, dans le document `messages.xml` dont voici un extrait,

```
<messages>
  <message numero="1" date="2024-01-01">
    <dest>promo2024</dest>
    <dest bcc="oui">Pierre Nerzic</dest>
    <contenu>Bonne année !</contenu>
  </message>
  ...
```

extraire le contenu du message n°4 s'écrit ainsi en XPath :

```
/messages/message[@numero=4]/contenu
```

Évaluation d'une expression XPath

Pour évaluer une expression en ligne de commande, il y a :

- `xmlstarlet sel --template --value-of` *expression document.xml*
- `xmllint --xpath` *expression document.xml*

On peut aussi travailler avec un navigateur, car XPath est disponible en JavaScript. Je vous fournis ce formulaire. Étudiez son source. C'est basé sur une `XMLHttpRequest` et la méthode `evaluate` sur le document XML qu'on reçoit.

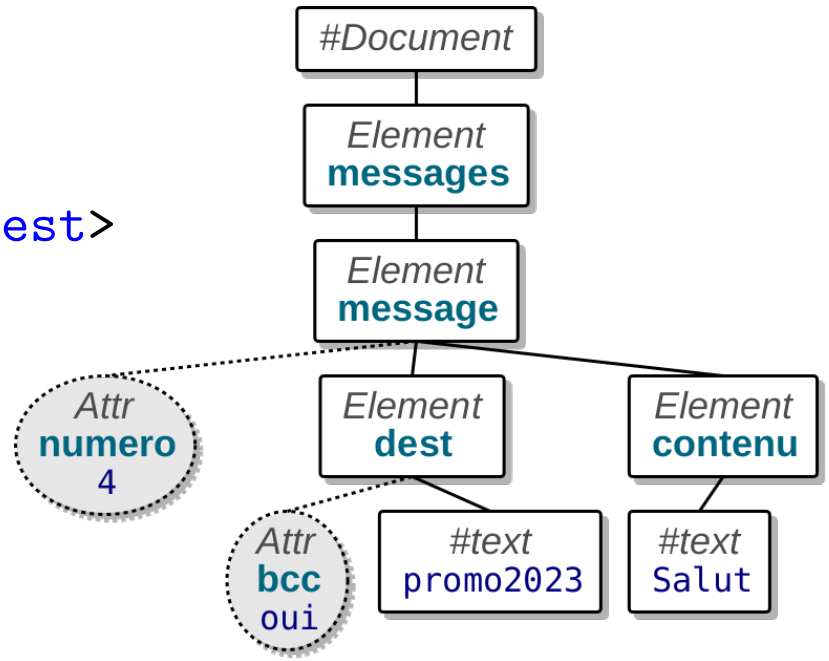
Cadre général

XPath sert à trouver des informations (éléments, attributs, textes) dans un arbre XML à l'aide d'une sorte de chemin.

Soit ce document XML :

```
<?xml version="1.0" ...?>
<messages>
  <message numero="4">
    <dest bcc="oui">promo2023</dest>
    <contenu>Salut</contenu>
  </message>
</messages>
```

Voici son arbre XML :



NB: le document complet contient tous les messages.

Chemin dans l'arbre du document

Le but d'XPath est d'aller chercher les informations voulues dans le document XML. Ex: quels sont les destinataires du message n°2 ?

Cela se fait à l'aide d'un chemin d'accès qui ressemble beaucoup à un nom complet Unix, avec ajout de conditions écrites entre `[]`.

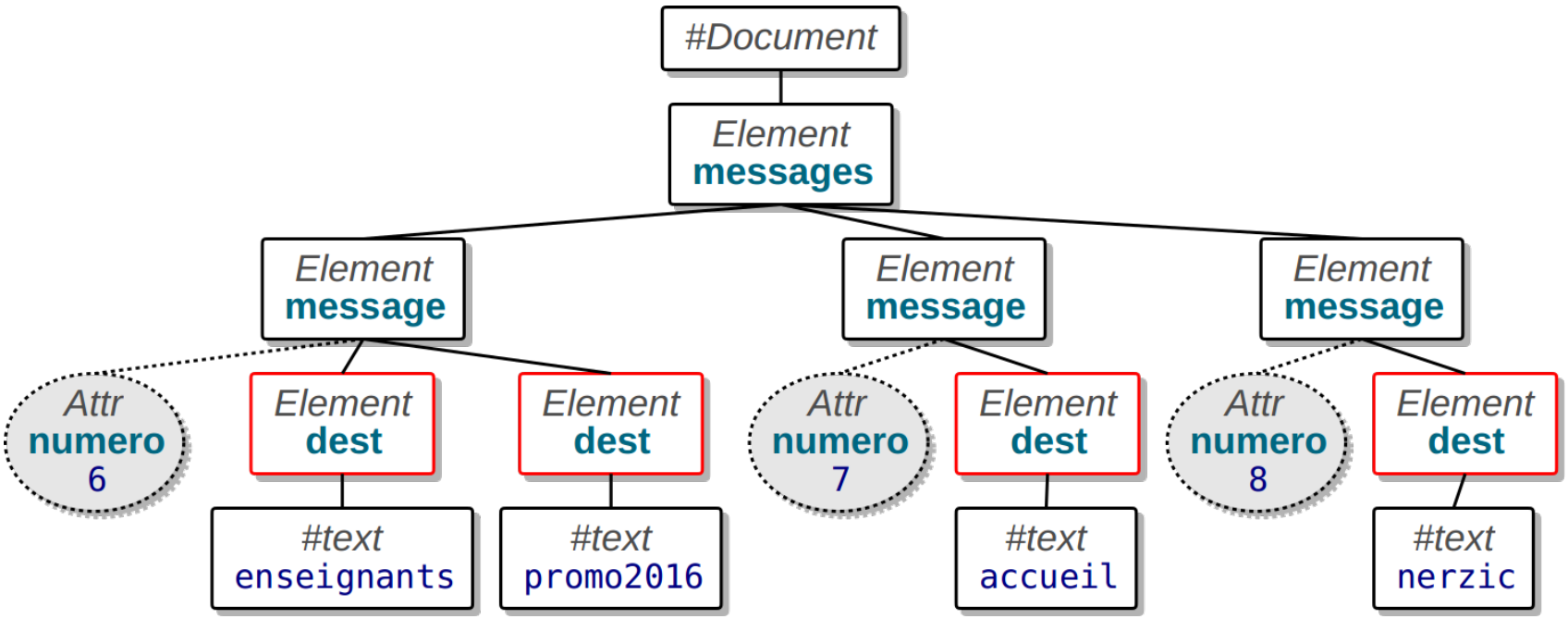
Exemple :

```
/messages/message[@numero="2"]/dest
```

C'est un peu comme un nom complet absolu dans Unix. Cependant, il y a énormément plus de possibilités pour écrire ces chemins et d'autre part, les chemins peuvent retourner plusieurs résultats.

Réponses multiples

Un point très important est qu'une expression XPath peut retourner plusieurs réponses. En effet, contrairement à Unix, un élément parent peut contenir plusieurs exemplaires du même élément enfant. Le chemin `/messages/message/dest` sélectionne 4 élements.



Pierre Nerzic

Structure d'une expression XPath simple

Une expression XPath est une suite d'étapes séparées par des séparateurs : `[sep] étape1 sep étape2 sep étape3...` Les étapes sont les noms des éléments dans lesquels il faut aller successivement.

Cependant cela dépend du séparateur employé :

- `/` Ce séparateur se comporte comme dans Unix : s'il est mis au début du chemin, il représente le document entier ; s'il est mis entre les étapes, c'est un simple séparateur.
- `//` Ce séparateur signifie de sauter un nombre quelconque d'éléments quelconques. C'est un peu comme si on écrivait `/*/*/*.../` un nombre indéfini de fois, y compris 0. S'il est au début du chemin, cela signifie alors que la première étape est à chercher n'importe où dans l'arbre.

Attributs des éléments

Pour désigner un attribut et non pas un sous-élément, on met un `@` devant le nom de l'attribut.

Exemples :

- `/messages/message/@numero` sélectionne tous les attributs nommés `numero` des éléments `<message>`.
- `//@numero` sélectionne tous les attributs portant ce nom sur n'importe quel élément du document.

NB: XPath retourne des nœuds de type Attribut sous la forme `nom="valeur"`. Pour avoir seulement la valeur, il faut écrire `string(chemin)`. Par exemple, `string(//message[dest="promo2024"]/@numero)`. Attention `string()` exige qu'il n'y ait qu'un seul attribut sélectionné par l'expression XPath.

Autres étapes d'un chemin

D'autres étapes peuvent être employées :

- `.` désigne le nœud courant,
- `..` désigne le nœud parent,
- `*` désigne tous les éléments de ce niveau. C'est plus restreint que `//`.

Exemples :

- `/messages/*/@numero` sélectionne tous les attributs nommés `numero` des éléments situés sous `<messages>`
- `//contenu[../@numero=3]` sélectionne les éléments `<contenu>` tels que leur élément parent a un attribut appelé `numero` et valant 3.

Pierre Nerzic

Conditions sur les étapes

Les conditions sur les étapes (éléments et attributs) sont appelées *prédicats*. Un prédicat se met entre `[...]` juste après l'élément dont il filtre l'un des enfants. On peut enchaîner plusieurs prédicats.

Exemple :

- `//message[@numero=5]/contenu` sélectionne les `<contenu>` des éléments `<message>` dont l'attribut `numero` vaut 5.
- `//message[contenu=""]/@numero` sélectionne les attributs `numero` des `<message>` ayant un sous-élement `<contenu>` vide.
- `//message[dest="iut"][@date="2024-01-01"]` sélectionne les éléments ayant un sous-élément `<dest>` valant « iut » et un attribut date valant « 2024-01-01 ».

Conditions sur les étapes, suite

Attention à bien lier le prédicat à l'élément concerné :

- `//message[./@numero="3"]` sélectionne seulement les messages ayant un attribut `numero` valant 3.
- `//message[//@numero="3"]` sélectionne *tous* les messages, si jamais on trouve quelque part un attribut `numero` valant 3

Dans `/element[condition]`, il faut que la condition concerne seulement l'élément.

Pierre Nerzic

Syntaxe des prédicats

Pour écrire les prédicats, XPath propose ces opérateurs un peu différents de ceux du C :

- arithmétique : `+ - * div mod` (et non pas `/` et `%`)
- comparaisons : `< <= = != >= >` (et non pas `==`)
- logique : `and or not(condition)`

Exemples :

- `//message[not(@numero < 5 or @numero >= 9)]` sélectionne les `<message>` dont l'attribut `numero` est entre 5 et 8.
- `//message[@numero mod 5 = 0]` sélectionne les `<message>` dont l'attribut `numero` est un multiple de 5.

Syntaxe des prédicats, suite

À part les conditions basées sur des comparaisons, il y a aussi :

- la notation [*index*] sélectionne l'élément ayant cet index (1 à n) dans la liste de son parent,
- le prédicat [*enfant*] est vrai si l'élément contient cet enfant.

Exemples :

- `//message[7]/contenu` sélectionne le `<contenu>` du 7e élément `<message>` du document.
- `//message[contenu and not(@date)]` sélectionne les `<message>` qui ont un `<contenu>` mais pas d'attribut date.

Fonctions XPath

XPath possède de très nombreuses fonctions, dont :

- Fonctions sur les éléments :
  - `string(s)` retourne le texte de l'expression `s`
  - `position()` retourne l'index de l'élément dans son parent (premier = n°1)
  - `last()` retourne le n° du dernier élément dans son parent

Exemples :

- `string(/messages/message[2]/contenu)` retourne le contenu du 2e message.
- `/messages/message[position()<=3]` sélectionne les 3 premiers éléments `<message>` du document.
- `//dest[position()>last()-3]` sélectionne les `<dest>` qui sont parmi les trois derniers enfants de leur parent.

Fonctions XPath (suite)

Une fonction est particulièrement utile : `count(expression)`. Elle compte le nombre de nœuds XML (élément, attributs, textes...) sélectionnés par l'expression. On l'utilise dans des conditions.

Attention, count() compte les nœuds sélectionnés, chacun dans son parent *séparément*. Ça conduit à faire des erreurs si on croit que `count` peut regrouper différents comptages.

Exemples :

- `//message[count(dest)>2]` retourne les éléments `<message>` ayant plus de deux enfants `<dest>`.
- `//message[count(//dest)>2]` retourne tous les éléments `<message>` s'il y a plus de deux éléments `<dest>` quelque part dans le document.

Pierre Nerzic

Fonctions XPath (suite)

- Fonctions sur les chaînes :
  - `string-length(s)` retourne la longueur de la chaîne `s`
  - `concat(s1, s2,...)` concatène les chaînes passées
  - `substring(s,deb,lng)` retourne `lng` caractères de `s` à partir du n°`deb` (premier = 1)
  - `contains(s1, s2)` vrai si `s1` contient `s2`
  - `starts-with(s1,s2)` et `ends-with(s1,s2)`
  - `matches(s, motif)` vrai si `s` correspond au motif

Exemple :

- `//message[string-length(contenu)<=15 and not(starts-with(dest, "promo"))]/@numero` retourne les numéros des messages dont le contenu ne fait pas plus de 15 caractères et aucun destinataire ne commence par « promo ».

Fonctions XPath (suite)

- Fonctions mathématiques :
  - `abs(nb)`, `ceiling(nb)`, `floor(nb)`, `round(nb)`
- Fonctions sur les dates et heures :
  - `year-from-dateTime(dt)`, `month-from-dateTime(dt)`, `day-from-dateTime(dt)`, `hours-from-dateTime(dt)`, `minutes-from-dateTime(dt)`, `seconds-from-dateTime(dt)`
  - `year-from-date(d)`, `month-from-date(d)`, `day-from-date(d)`
  - `hours-from-time(t)`, `minutes-from-time(t)`, `seconds-from-time(t)`

Retour sur les composants d'un chemin

Un chemin XPath est constitué de `[sep] étape1 sep étape2 sep étape3...` Chaque étape est soit le nom d'un élément, soit `@` et le nom d'un attribut ; chacune suivie éventuellement d'un prédicat entre crochets :

```
/racine/element1[filtre1]/.../@attribut[filtre3]
```

Les étapes sont appelées *sélecteurs*. À la fin d'une expression, on peut employer des sélecteurs spéciaux comme :

- `text()` sélectionne tous les textes sous l'élément courant, y compris dans tous ses descendants.
- `node()` sélectionne tous les nœuds enfants de l'élément.

Exemple :

- `/messages/message/contenu/text()`

Pierre Nerzic

Axes

XPath permet de rajouter encore une « décoration » sur chaque étape, la *direction* dans laquelle aller à partir de l'étape courante. Cette direction est appelée *axe*. Cela donne la syntaxe :

```
/racine/axe1::element1[filtre1]/axe2::element2[filtre2]/...
```

Par défaut, on descend toujours vers les enfants du nœud courant au niveau de chaque étape. Cet axe s'appelle `child`.

Exemple, ces deux syntaxes signifient la même chose :

- `/messages/message/@numero`
- `/messages/child::message/attribute::numero`

D'autres axes existent.

Axes

L'axe définit vers quels nœuds aller à chaque étape. Voici quelques axes utiles à connaître parmi ceux qui existent :

`child::` parcourir les nœuds enfants du contexte ; c'est l'axe utilisé par défaut.

`descendant::` parcourir tous les nœuds enfants et petit-enfants ; ça revient un peu à utiliser `//`.

`parent::` parcourir le nœud parent du contexte ; ça revient à utiliser `..` mais avec un test sur le parent voulu.

`ancestor::` parcourir tous les nœuds parent et grand-parents.

`preceding-sibling::` parcourir tous les nœuds frères précédents

`following-sibling::` parcourir tous les nœuds frères suivants

`attribute::` parcourir les nœuds attributs du contexte ; c'est l'axe par défaut pour une étape commençant par un `@`.

Exemples de chemins avec axes

- `/messages/message[last()]/child::contenu` retourne le contenu du dernier message du document.
- `/messages/message[@numero=7]/descendant::dest` sélectionne tous les nœuds situés sous le message n°7.
- `//message[@numero=5]/preceding-sibling::message` sélectionne les messages situés avant le n°5.
- `//dest[@bcc="oui"]/parent::node()` sélectionne le nœud parent d'un élément `<dest>` dont l'attribut `bcc` vaut oui.
- `//message[3]/attribute::numero` retourne l'attribut numéro du 3e message présent dans le document.

C'est tout pour aujourd'hui

Cette présentation est finie. Rendez-vous avec le TP5 et les suivants pour mettre cela en pratique.

Pierre Nerzic