Introduction à XML

Fabrice Rossi

http://apiacoa.org/contact.html.

Université Paris-IX Dauphine

Plan du cours

1. XML :
   (a) introduction
   (b) le langage
   (c) les DTD
   (d) les espaces de noms
2. les schémas
3. API de manipulation :
   (a) SAX
   (b) DOM
4. les transformations : XSLT

Documents : `http://apiacoa.org/teaching/xml/`

INTRODUCTION

XML ?

XML (*eXtensible Markup Language*) :

- norme du W3C (`http://www.w3.org/TR/REC-XML`), datant du 10/02/98
- XML sert à **stocker des données structurées dans un fichier texte** :
  - données structurées = arbre
  - *Markup* = balise $\Rightarrow$ structuration
- XML $\neq$ HTML :
  - n'est pas limité au Web
  - doit être adapté à chaque utilisation
  - pas toujours très lisible
- XML ressemble à HTML :
  - langages à balise
  - dérivés de SGML

Exemple

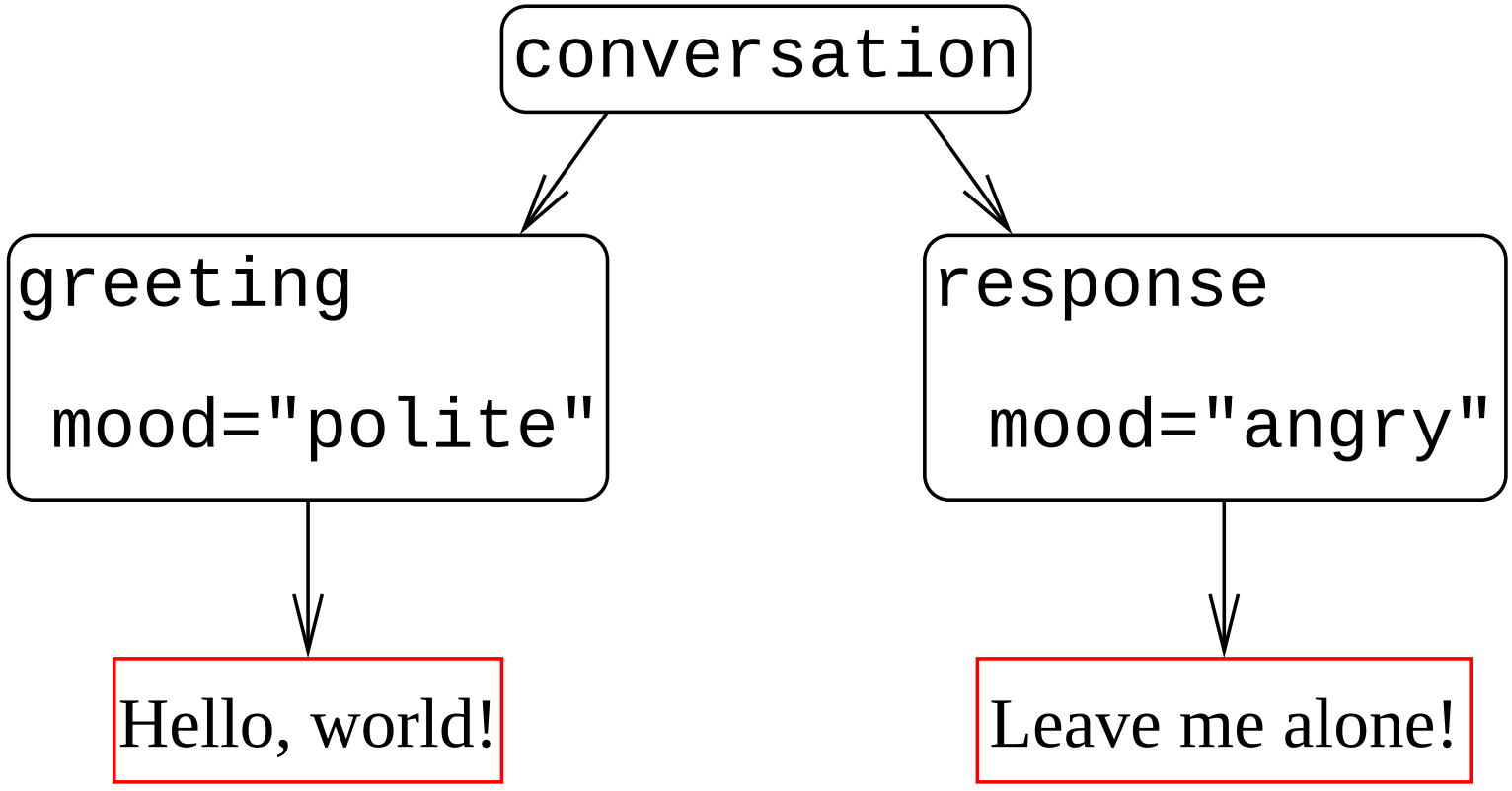
Exemple de fichier XML :

hello.xml

```
1  <?xml version="1.0" standalone="yes"?>
2  <conversation>
3    <greeting mood="polite">Hello, world!</greeting>
4    <response mood="angry">Leave me alone!</response>
5  </conversation>
```

1. en-tête “presque” obligatoire : c’est une PI (*processing instruction*)
2. balise ouvrante (ici la racine de l’arbre) : débute un nœud (un élément)
3. `mood` est un attribut, `polite` est sa valeur
5. balise fermante correspondant à la BO de la ligne 2

Arbre de l'exemple



- l'inclusion textuelle traduit la relation mère/fille
- les attributs précisent les nœuds

Intérêts de XML

Deux points importants :

1. XML est un **standard ouvert et accepté**
2. XML est une solution complète de stockage, manipulation, transformation, etc. de données structurées

Standard $\Rightarrow$ économie de développement. Produits disponibles :

1. éditeur
2. analyseur syntaxique
3. validateur (vérification de cohérence)
4. moteur de transformation
5. etc.

XML comme format de fichier

Le standard et ses extensions proposent :

- support de très nombreux encodages (Unicode, ISO Latin, etc.) $\Rightarrow$ international
- inclusion et macro (très élémentaires)
- contraintes de structure :
  - *Document Type Definition* (DTD), l'ancienne solution
  - schéma, les nouvelles solutions
- méta-organisation : *namespace*
- liens évolués : *XPointer* et *Xlink*
- stockage :
  - bases de données XML (langage de requêtes XQuery en cours de standardisation)
  - compression spécialisée
  - calcul de différences (gestion de version)

XML pour le programmeur

Le standard et ses extensions proposent :

- analyse syntaxique :
  - modèle objet : *Document Object Model* (DOM)
  - modèle évènementiel : *Simple API for Xml* (SAX)
  - sérialisation
- validation (en général intégrée à l'analyseur) :
  - DTD
  - schéma
- moteur de transformation (d'un document XML en un autre) : *XSLT*
- expressions rationnelles XML : *XPath*

Outils Open Source par le groupe apache `http://xml.apache.org` :

- Xerces : analyse syntaxique et validation
- Xalan : transformation et expressions rationnelles (voir aussi Saxon, `http://saxon.sourceforge.net/`)

Applications

Outils et normes utiles :

- base de connaissances : *Ressource Description Framework* (RDF)
- transformation vers HTML et WML : *XSLT*
- impression de qualité : *XSL :FO*
- dessin vectoriel : *Scalable Vector Graphics* (*SVG*)
- animation : *SMIL*
- mathématiques : *MathML*
- RPC en XML : *xmlrpc* et *SOAP*
- etc.

Outils Open Source par le groupe apache `http://xml.apache.org` :

- FOP : support de *XSL :FO*
- Batik : support de *SVG*

LE LANGAGE XML

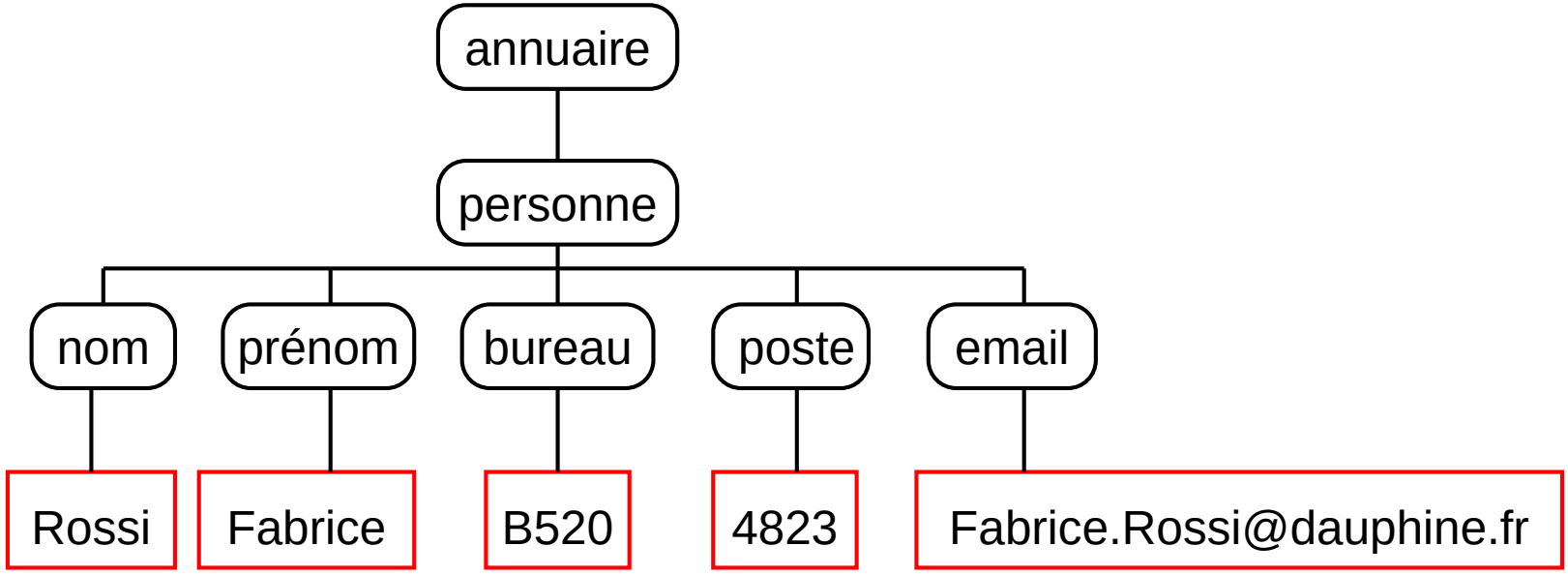
Données structurées

XML permet de représenter des données structurées :

- données textuelles (binaire : codage, par exemple basé sur *mime*)
- organisées :
  - on manipule un **document** constitué d'**éléments**
  - un **élément** peut être constitué simplement de **texte** ou contenir d'autres éléments (ou un mélange des deux)
  - un élément peut être associé à des informations complémentaires, les **attributs**
- la structure est celle d'un arbre :
  - un document XML = un arbre
  - un élément = un nœud de l'arbre
- le standard indique comme traduire l'arbre en un texte XML, pas comment organiser les données

Exemple

- but : stocker l'annuaire de Dauphine (nom, prénom, bureau, numéro de poste, *email*)
- le texte du document : les informations !
- organisation : s'arranger pour que les informations restent correctement groupées (ne pas mélanger les données !)
- une possibilité :



Exemple (suite)

Traduction en XML de l'arbre :

annuaire1.xml

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <annuaire>
3  <personne>
4  <nom>Rossi</nom>
5  <prénom>Fabrice</prénom>
6  <bureau>B520</bureau>
7  <poste>4823</poste>
8  <email>Fabrice.Rossi@dauphine.fr</email>
9  </personne>
10 <!-- suite de l'annuaire -->
11 </annuaire>
```

- inclusion textuelle $\Leftrightarrow$ relation mère-fille dans l'arbre
- balise ouvrante ou fermante $\Leftrightarrow$ nom d'un nœud
- texte $\Rightarrow$ feuille de l'arbre

Ne pas confondre les éléments (information) et les balises (syntaxe).

Organisation

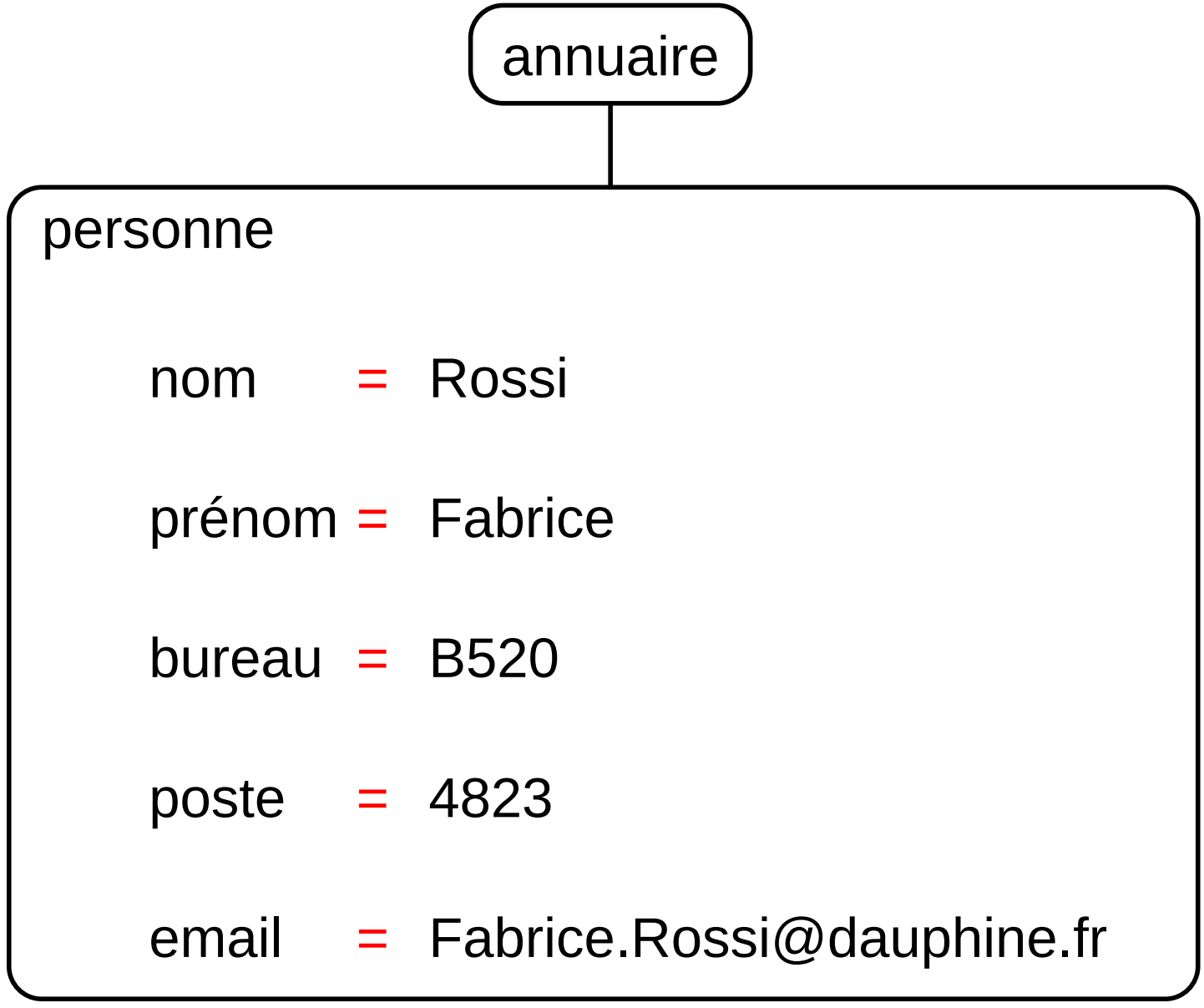
On organise les données en décidant de la structure de l'arbre :

- le nom des éléments
- l'ordre des éléments
- les relations d'inclusion
- la position des données (c'est-à-dire du texte)
- les contraintes sur les données (texte quelconque, valeur numérique, etc.)
- les attributs

Une organisation particulière forme un **dialecte** XML, par exemple :

- MathML : pour décrire des équations
- xbel : pour décrire des *signets*
- SVG : dessin vectoriel
- XHTML : HTML re-spécifié en XML
- etc.

Autre solution pour l'annuaire



Traduction XML

annuaire2.xml

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <annuaire>
3  <personne nom="Rossi"
4            prénom="Fabrice"
5            bureau="B520"
6            poste="4823"
7            email="Fabrice.Rossi@dauphine.fr"/>
8  <!-- suite de l'annuaire -->
9  </annuaire>
```

- attributs $\Leftrightarrow$ annotations d'un nœud
- élément vide $\Rightarrow$ feuille

Différences

Le choix de la structure n'est pas anodin :

- l'intérieur d'un attribut n'est pas structuré
- au maximum un seul exemplaire d'un attribut dans un élément
- impossible de faire ce qui suit avec des attributs sans introduire lourdeurs et limitations :

annuaire3.xml

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <annuaire>
3  <personne>
4  <nom>Rossi</nom>
5  <prénom>Fabrice</prénom>
6  <prénom>Dominique</prénom>
7  <bureau>B520</bureau>
8  <poste>4823</poste>
9  <email><nom>Fabrice.Rossi</nom><domaine>dauphine.fr</domaine></email>
10 <email><nom>rossi</nom><domaine>ceremade.dauphine.fr</domaine></email>
11 </personne>
12 </annuaire>
```

Une solution (lourde !) par attributs

On peut proposer :

annuaire4.xml

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <annuaire>
3  <personne nom="Rossi"
4            prénom="Fabrice"
5            prénom2="Dominique"
6            bureau="B520"
7            poste="4823"
8            email-nom="Fabrice.Rossi"
9            email-domaine="dauphine.fr"
10           email-nom2="rossi"
11           email-domaine2="ceremade.dauphine.fr"/>
12 </annuaire>
```

- très lourd
- oblige à prévoir tous les cas (redondance dans les traitements ultérieurs) : on peut toujours structurer le contenu d'un élément *a posteriori*, pas celui d'un attribut

Syntaxe XML

Deux niveaux syntaxiques :

1. bas niveau : document **bien formé**.
2. haut niveau : document **valide** (respectant une DTD). haut niveau $\Rightarrow$ bas niveau.

Du point de vue utilisateur/concepteur :

1. le bas niveau est **obligatoire** : mal formé $\Rightarrow$ pas XML.
2. le bas niveau est **fixé** par la norme.
3. le haut niveau est **facultatif** : bien formé $\Rightarrow$ XML.
4. le haut niveau est **entièrement de la responsabilité** du concepteur : il définit les contraintes syntaxiques (noms de éléments, organisation, etc.)
5. le haut niveau peut se mettre en œuvre de différentes façons (DTD, schémas W3C, Relax NG, etc.)

Documents XML bien formés

Les éléments :

- `<truc>` : **balise ouvrante** :
  1. doit toujours correspondre à une **balise fermante** (parenthèsage correct), ici `</truc>`
  2. le texte entre <> est le nom de l'**élément** : constitué de lettres, chiffres, '.', '-', '_' et ' :'
- `</quantité>` : **balise fermante** (depuis une balise ouvrante jusqu'à une balise fermante : le contenu d'un **élément**, un **nœud** de l'arbre)
- `<et_hop/>` : **balise mixte**, ouvrante et fermante, pour les éléments vides

Exemples :

- `<a><b></a></b>` : mal formé
- `<p>bla, bla, bla<br>bla, bla, bla</p>` : mal formé
- `<nom.pas:très_bien-choisit/>` : bien formé

Documents XML bien formés (2)

Les attributs :

`<font name="times">` : `name` est un **attribut** de l'élément `font`, de **valeur** `times` :

- ne peut apparaître que dans une balise ouvrante ou mixte
- doit **toujours** avoir une valeur
- la valeur est **toujours** délimitée par des guillements `"` ou des apostrophes `'`
- dans la valeur, `<` est interdit
- pour le nom d'un attribut, même contrainte que pour les éléments
- dans une même balise ouvrante ou mixte, chaque attribut ne peut apparaître qu'une fois

Exemples

Quelques fragments de documents mal formés :

- `<p align=center>du texte centré</p>`
- `<img src="truc.gif" noborder/>`
- `<un/truc>par exemple</un/truc>`
- `<a>contenu de l'élément</A>`
- `<formule valeur='(3+4)<2'/>`
- `<a><b>dddd</b val='big'></a>`
- `<a x='1' x='2'/>`

Fragments bien formés :

- `<pas texte='de "problème"'></pas>`
- `<a nom.pas:très_bien-choisit='une valeur'/>`

Remarque : contrairement à HTML, XML est *case sensitive*.

Documents XML bien formés (3)

Grammaire de base :
un document XML est un arbre d'éléments :

- la racine est **unique**
- le contenu d'un élément est :
  - d'autres éléments
  - du texte (les *character data*) : `<` et `&` interdits

Exemples :

- `<a></a><b></b>` : mal formé
- `<a>3<2</a>` : mal formé
- `<a>3>2</a>` : bien formé (déconseillé)
- `<a>bla <br/>bla</a>` : bien formé

Constructions utiles

- commentaires :

  <!-- ce qu'on veut sauf deux - à la suite -->
- CDATA (texte) :

cdata.xml

```
1 <![CDATA[ <a>contenu<b> non interprété,
2 non analysé, ne fait pas</a> partie
3 de l'arbre</b> ]]>
```

- entités :
  - gestion de la structure physique des documents XML
  - mécanisme de "macro" XML :
    - inclusion d'un document dans un autre
    - référence externe
    - remplacement d'un texte par un autre
    - *customization*

Les entités

Entités de bas niveau :

- syntaxe : `&nom;`
- “prédéfinies” pour les caractères spéciaux :

caractère	<	>	&	’	"
entitée	`<`	`>`	`&`	`'`	`"`

- accès aux caractères par leur code UNICODE : `&#`nombre en base 10`;` ou `&#x`nombre en base 16`;`. Par exemple `'` correspond à ’.

Entités de haut niveau :

- à définir dans la DTD
- inclusion
- remplacement
- *customization*

Exemple complet

adresses.xml

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <carnet>
3  <fiche>
4  <nom>Rossi</nom><prénom>Fabrice</prénom>
5  <adresse>
6  <service>UFR MD</service>
7  <rue>Place du Maréchal de Lattre de Tassigny</rue>
8  <code>75016</code><ville>Paris</ville>
9  </adresse>
10 <téléphone>
11 <fixe>01 44 05 48 23</fixe>
12 <fax>01 44 05 40 36</fax>
13 <portable>06 06 06 06 06</portable>
14 </téléphone>
15 <email><nom>Fabrice.Rossi</nom><domaine>dauphine.fr</domaine></email>
16 </fiche>
17 </carnet>
```

Entête

- il est **conseillé** de commencer un document XML par : `<?xml version="1.0"?>`
- si on donne l'en-tête, `version="1.0"` est **obligatoire**
- l'attribut `encoding` permet d'indiquer la représentation physique des caractères du fichier :
  - `<?xml version="1.0" encoding="UTF-16"?>`
  - `<?xml version="1.0" encoding="UTF-8"?>` par défaut (supporte l'ASCII)
  - `<?xml version="1.0" encoding="ISO-8859-1"?>` sous linux
- `<?nom?>` : une *Processing Instruction*. Indique aux logiciels comment traiter le document :
  - encodage
  - associer une feuille de style à un document

Validation

LES DTD

Documents valides

- syntaxe de haut niveau (grammaire)
- précisée par une DTD :
  - existe depuis la norme XML
  - outils stables
  - limitées : structure simple
  - syntaxe non XML
- précisée par un schéma :
  - deux grandes technologies (d'autres existent) :
    - les schémas du W3C (recommandation du 2 Mai 2001)
    - RELAX NG du consortium OASIS (spécification du 12 Décembre 2001)
  - syntaxe XML
  - très puissants
  - assez complexes

DTD et schéma

Principes communs :

- éléments autorisés
- modèle du contenu d'un élément (et donc grammaire)
- attributs autorisés
- modèle de la valeur d'un attribut

Différences :

- syntaxe :
  - DTD : issue de SGML
  - Schéma : un document XML comme un autre
- contraintes :
  - DTD : contraintes très simples
  - Schéma : très évoluées (contenu entier, date, expressions régulières, etc.)

Certains spécialistes (J. Clark par exemple) proposent l'abandon total des DTDs. Peut être pour XML 2.0...

DTD

Entête :

- DTD externe (fichier `carnet.dtd`) :

```
<!DOCTYPE carnet SYSTEM "carnet.dtd">
```

- DTD interne :

adresses-with-dtd-part.xml

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <!DOCTYPE carnet [
3 <!-- contenu de la DTD -->
4 ]>
5 <carnet>
6 <!-- contenu supprimé -->
7 </carnet>
```

- dans les deux cas : le nom qui suit `DOCTYPE` est celui de l'élément racine

Déclaration d'un élément

Pour déclarer un élément :

- `<!ELEMENT carnet` modèle du contenu `>`
- modèles possibles :
  - `ANY` : quelconque
  - `EMPTY` : élément vide
  - `(#PCDATA)` : du texte
  - modèle basé sur les expressions régulières :
    - séquence : `,`, alternative : `|`
    - 1 au moins : `+`, nombre libre : `*`, 1 au plus : `?`
    - combinaison grâce aux parenthèses
  - mixte : texte plus éléments :
    - seulement une alternative entre le texte et des éléments
    - 1 occurrence ou un nombre quelconque (i.e., par de `+` ni de `?`)
- un seul modèle pour un nom donné

Exemples

- `<!ELEMENT carnet (fiche*)>`
  L'élément `carnet` peut contenir un nombre arbitraire d'éléments `fiche`.
- `<!ELEMENT fiche (nom, prénom, téléphone, email*)>`
  L'élément `fiche` contient exactement un `nom`, un `prénom`, un `téléphone` et autant d'`email` qu'on le souhaite (le tout dans cet ordre).
- `<!ELEMENT fixe (#PCDATA)>`
  L'élément `fixe` contient exclusivement du texte.
- `<!ELEMENT téléphone (fixe|fax|portable)*>`
  L'élément `téléphone` contient dans n'importe quel ordre et en n'importe quelle quantité des éléments `fixe`, `fax` et `portable`.

Contenu mixte

Utile, mais pas assez de contraintes. Exemple :

texte.xml

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <texte>
3  Un exemple de <code>XML</code> qui mélange <bf>éléments</bf>
4  et <emph>texte classique</emph>, ce qui peut poser
5  des <bf><underline>problèmes</underline></bf>
6  <underline>délicats</underline>.
7  </texte>
```

Modèle (une partie) :

texte.dtd

```
1  <!ELEMENT texte (#PCDATA|code|bf|emph|underline)*>
2  <!ELEMENT code (#PCDATA|code|bf|emph|underline)*>
```

Problème : on peut s'éloigner du texte d'exemple.

Déclaration d'attributs

Pour déclarer un attribut :

- `<!ATTLIST élément attribut` définition de l'attribut >
- plusieurs attributs :

attribut-dtd-part.xml

```
1  <!ATTLIST bookmark
2              href     CDATA #REQUIRED
3              visited  CDATA #IMPLIED
4              modified CDATA #IMPLIED >
```

- il est conseillé de n'avoir qu'une `ATTLIST` par élément
- définition d'attribut : type et valeur par défaut
- types possibles (il en existe d'autres plus complexes et moins utiles) :
  - `CDATA` : texte
  - `ID` : label ; `IDREF` et `IDREFS` : référence à un label
  - énuméré : valeurs possibles séparées par des `|` et encadrées par des parenthèses

Collisions de noms

- chaque élément a son propre espace de noms pour ses attributs : un même nom d'attribut peut être utilisé différemment dans deux éléments distincts
- l'espace de noms des attributs est distinct de celui des éléments : un attribut et un élément peuvent avoir le même nom
- exemple (fragment correct) :

DisquesML-part.dtd

```
1  <!ATTLIST groupe nom ID #REQUIRED>
2  <!ELEMENT nom (#PCDATA) >
3  <!ATTLIST interprète nom IDREF #REQUIRED>
```

Valeurs par défaut pour les attributs

Pour la partie “valeur par défaut”, on peut indiquer :

- `#REQUIRED` : attribut **obligatoire** et pas de valeur par défaut.
- `#IMPLIED` : attribut **facultatif** et pas de valeur par défaut.
- une valeur : c’est la valeur par défaut
- `#FIXED` suivi d’une valeur : valeur obligatoire pour l’attribut

Exemples :

attribut-dtd-part-2.xml

```
1  <!ATTLIST xbel version CDATA #FIXED "1.0" >
2  <!ATTLIST metadata owner CDATA #REQUIRED >
3  <!ATTLIST folder
4              id ID #IMPLIED
5              folded (yes|no) 'yes' >
```

Les références croisées

Les types `ID`, `IDREF` et `IDREFS` permettent des références croisées au sein d'un document :

- `ID` :
  - pour associer un label à un élément
  - un seul attribut `ID` par élément
  - `#REQUIRED` ou `#IMPLIED`
  - contenu unique dans un document
- `IDREF` et `IDREFS` :
  - pour faire une référence à un label
  - doit obligatoirement faire référence à un label existant
  - `IDREFS` : plusieurs labels séparés par des espaces

Exemple (XML)

disques.xml

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <!DOCTYPE disques SYSTEM "DisquesML.dtd">
3  <disques>
4  <groupe nom="muse">
5  <nom>MUSE</nom>
6  <membre>Matthew Bellamy</membre>
7  <membre>Dominic Howard</membre>
8  <membre>Chris Wolstenholme</membre>
9  </groupe>
10 <disque>
11 <interprète nom="muse"/>
12 <titre>Showbiz</titre>
13 </disque>
14 <disque>
15 <interprète nom="muse"/>
16 <titre>Origin of symmetry</titre>
17 </disque>
18 </disques>
```

Exemple (DTD)

DisquesML.dtd

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <!ELEMENT disques (groupe*, disque*)  >
3  <!ELEMENT groupe (nom,membre+) >
4  <!ATTLIST groupe nom ID #REQUIRED>
5  <!ELEMENT nom (#PCDATA) >
6  <!ELEMENT membre (#PCDATA) >
7  <!ELEMENT disque (interprète, titre)>
8  <!ELEMENT interprète EMPTY >
9  <!ATTLIST interprète nom IDREF #REQUIRED>
10 <!ELEMENT titre (#PCDATA) >
```

- le validateur vérifie les références croisées
- permet (entre autres) d'éviter la redondance dans un fichier XML

Entités

Une entité est une ***storage unit*** d'un document XML :

- entité analysée (*parsed entity*) : un morceau du texte d'un document XML
- entité non analysée (*unparsed entity*) : une ressource externe référencée par un document XML (par exemple une image)
- entité générale : utilisée dans le corps d'un document XML (cf exemples précédents : référence à un caractère, caractères spéciaux)
- entité paramètre : utilisée seulement dans la DTD

Partie la plus lourde de XML !

Entités comme macros

Entité générale :

- définition : `<!ENTITY nom "texte de remplacement">`
- utilisation : `&nom;`
- utilisation interdite dans la DTD (sauf dans le texte de remplacement d'une autre entité)
- **remplacement récursif** au moment de l'utilisation (sauf pour les références à des caractères, remplacées à la définition)

Exemple :

entity-1.xml

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <!DOCTYPE texte [
3    <!ELEMENT texte (#PCDATA)>
4    <!ENTITY auteur "Fabrice Rossi">
5  ]>
6  <texte>&auteur;</texte>
```

Entités comme macro dans les DTD

Entité paramètre :

- définition : `<!ENTITY % nom "texte de remplacement">`
- utilisation : `%nom;`
- utilisable seulement dans la DTD
- **remplacement récursif immédiat**

Exemple :

entity-2.xml

```
1  <!ENTITY % url.att      "href     CDATA #REQUIRED
2                           visited  CDATA #IMPLIED
3                           modified CDATA #IMPLIED"
4  >
5  <!ELEMENT bookmark (title?, info?, desc?)>
6  <!ATTLIST bookmark
7              %url.att;
8  >
```

Entités pour l'inclusion

Entité externe (paramètre ou générale) :

- définition : `<!ENTITY nom SYSTEM "URI">`
- utilisation : `&nom;`
- principe : le fichier référencé par l'URI est inclus dans le fichier référençant **en cas de validation**

doc1.xml

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <texte>et hop</texte>
```

doc2.xml

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <!DOCTYPE recueil [
3    <!ELEMENT recueil (texte)*>
4    <!ELEMENT texte (#PCDATA)>
5    <!ENTITY contenu SYSTEM "doc1.xml">
6  ]>
7  <recueil>&contenu;</recueil>
```

LES ESPACES DE NOMS

Les *namespaces*

Un document XML peut contenir des éléments et des attributs qui correspondent à plusieurs domaines distincts (i.e., à plusieurs dialectes).

Problème : comment gérer les collisions ?

Solution $\Rightarrow$ Les *namespaces* :

- recommandation du 14/01/99
- permet d'introduire des collections de noms utilisables pour les éléments et les attributs d'un document XML
- principes :
  - chaque collection est identifiée par un URI
  - une construction (un attribut) permet d'associer un URI à un préfixe
- pas de réel support des *namespaces* par les DTDs

Exemple

namespaces-complet.xml

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <html:html xmlns:html="http://www.w3.org/TR/REC-html40">
3    <html:head>
4      <html:title>Démonstration</html:title>
5    </html:head>
6    <html:body>Un contenu</html:body>
7  </html:html>
```

- l'attribut `xmlns:html` associe le préfixe `html` à l'URI `"http ://www.w3.org/TR/REC-html40"`
- l'association n'est valable que dans les descendants (au sens large) de l'élément qui contient `xmlns:html`
- l'association s'applique aux éléments et aux attributs

Cas général

Déclaration d'un *namespace* :

- `xmlns:`préfixe`="URI"` : association du préfixe à l'URI
- `xmlns="URI"` : définition de l'URI associé à l'espace de noms par défaut (sans préfixe)

Nom qualifié :

- préfixe`:`nom local
- peut être utilisé pour les attributs et les éléments
- le préfixe **doit** être déclaré par un ascendant

Remarques :

- c'est l'URI qui assure l'absence d'ambiguïté, pas le préfixe
- le dernier qui parle a raison

Les attributs

Une subtilité :

- les attributs avec un nom qualifié sont traités comme les éléments, i.e., globalement
- les attributs sans préfixe sont dans un espace de noms local propre à l'élément dans lequel ils apparaissent (c'est le cas classique sans *namespace*)

Dans un élément donné, les attributs doivent être disctints, c'est-à-dire :

- ou bien être des attributs qualifiés qui se distinguent par leur partie locale ou par leur URI
- ou bien être des attributs sans préfixe distincts

Exemples

namespaces-html.xml

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <html xmlns="http://www.w3.org/TR/REC-html40">
3    <head>
4      <title>Démonstration</title>
5    </head>
6    <body>Un contenu</body>
7  </html>
```

attribut-ns.xml

```
1  <x xmlns:n1="http://www.w3.org"
2     xmlns="http://www.w3.org" >
3    <good a="1"      b="2" />
4    <good a="1"      n1:a="2" />
5  </x>
```