.: XML :.

eXtended Markup Language

I. Introduction
II. Document XML
III. DTD
IV. Mise en page - Présentation (XSL)
V. Manipulation des documents XML
VI. UML et XML

Bibliographie

- "XML" Collection e-Poche Ed. Micro Application
- "XML pour l'entreprise" (D.Girard et T.Crusson) http://www.application-servers.com/livresblancs/xml/
- "Modélisation d'applications XML avec UML" (David Carlson) Ed. Eyrolles
- "Webmaster in a nutshell" Ed. O'Reilly
- et une série de documents disponibles sur internet ...

I. Introduction

- Historique
- Définitions
- Avantages et comparaisons
- Technologie incontournable
- Famille de langages
- Normes métiers
- Terminologies XML
- UML et XML

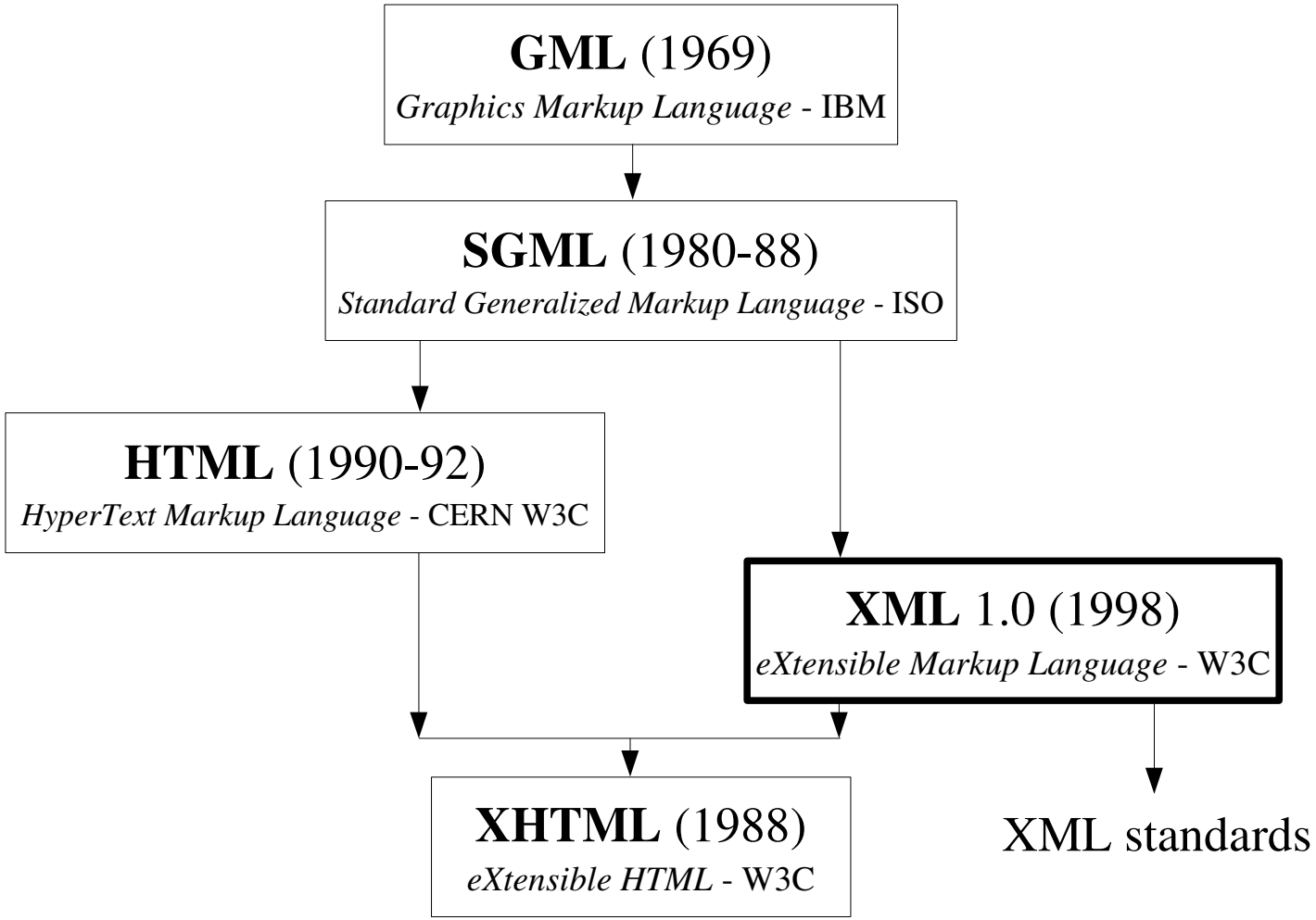
Les documents structurés

- Besoin: un format standard pour les documents informatiques pour,
  - les échanger entre applications
  - les distribuer sur plusieurs supports (Web, papier, ...)
  - faire des recherches « intelligentes »

Avant XML

- Il existait :
  - un langage de balisage normalisé, riche en sémantique mais complexe : **SGML** (*Standard Generalized Markup Language* - norme ISO 8879)
  - un langage destiné et donc adapté au Web mais dont les applications sont limitées par une bibliothèque figée et réduite de balises essentiellement destinées à la présentation : **HTML** (*HyperText Markup Language*).

Arbre généalogique



Définitions

- XML signifie eXtensible Markup Language (Langage de balisage extensible)
- XML est un sous-ensemble au sens strict de SGML (Standard Generalized Markup Language)
- XML est standardisé par la spécification W3C XML 1.0 du 10/02/98
- XML n'est pas un langage de programmation
- XML est un méta-langage exploitable pour créer d'autres langages
- XML bien formé signifie que le texte XML obéit aux règles syntaxiques de XML
- XML valide signifie que le texte XML est bien formé et répond à une structure définie par une DTD
- XML est destiné à l'échange d'informations et de documents
- XML est une solution pour la modélisation des contenus et la standardisation de modèles de contenus

Avantages

- Lisibilité : aucune connaissance ne doit théoriquement être nécessaire pour comprendre un contenu d'un document XML (autodescriptif et extensible)
- Structure arborescente : permettre de modéliser la majorité des problèmes informatiques
- Universalité et portabilité : ASCII + encodage UNICODE
- Déployable : être facilement distribué par n'importe quels protocoles capables de transporter du texte (comme HTTP)
- Intégrabilité : être utilisable par toute application capable d'analyser du code XML (*parser*)
- Exensibilité (métalangage) : être utilisable dans tous les domaines d'applications

Comparaisons

- XML vs HTML:
  - syntaxe plus stricte
  - aucune balise prédéfinie, fixe et figée : XML est un méta-langage
  - X pour eXtensible: définir ses balises selon ses besoins
  - gère tous les jeux de caractères
  - séparation contenu/présentation
- XML vs SGML:
  - syntaxe fixe : plus simple à analyser (parser)
  - définition des balises optionnelle
  - plus facilement éditable à la main
  - moins complexe à mettre en oeuvre

Technologie incontournable

- XML est présent dans des contextes aussi variés que :
  - les applications distribuées
  - la configuration de produits
  - les annuaires
  - l'édition de documents
  - la diffusion de contenu sur le web
  - la gestion de la connaissance
  - ...

Une famille de langages

- XML est plus qu'un langage, c'est une famille de langages.
- Actuellement on estime que plusieurs centaines de « langages » basés sur XML ont été décrits.

XHTML, XSL, XSLT, XSLFO, Xpath, XLink, XPointer, XML-Schema, RSS, MathML, SVG, ...

La galaxie XML

- Espaces de noms (*NameSpace*) : éviter les conflits entre noms de balises
- XSL (*eXtensible Stylesheet Language*), XSLT : tranformations d'un document XML et XSL-FO : formatage en vue de l'affichage
- XPath : mécanisme pour se référer à une partie d'un document XML
- XLink, XPointer : liens entre documents
- XML Query : langage de requêtes
- Les DTD et XML Schema : définir la syntaxe des balises et la structure
  - DTD (*Document Type Definition*) : vient de SGML, non-XML
  - XML Schema : plus riche, écrite en XML

Normes métiers

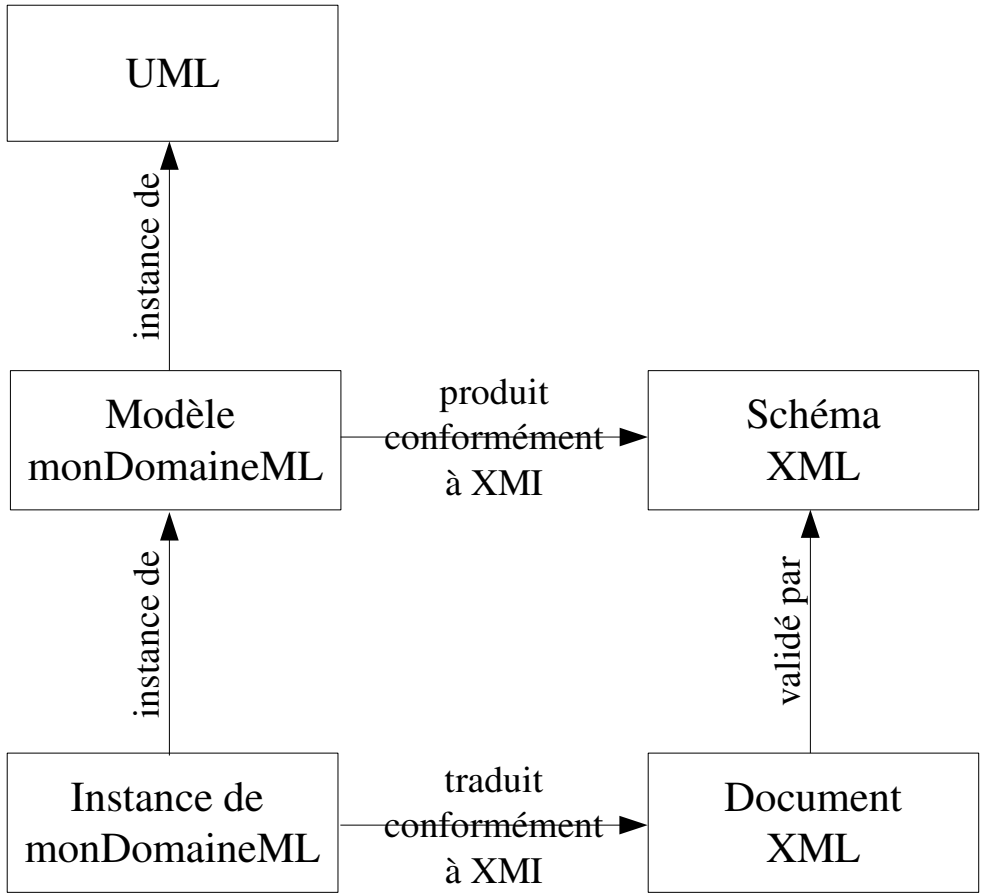
- Au delà de la syntaxe commune XML, il convient de définir des mots et des phrases en constituant un vocabulaire propre à son application, à son domaine d'activité.
- De nombreux organismes tentent de définir ces vocabulaires communs. Cependant, il est tout à fait possible de définir son propre schéma.
- Normes XML métier : par exemple, ebXML pour les échanges de commerce éléctronique.
- C'est le consortium international OASIS qui fédère les informations sur les normes métiers disponibles : http://oasis-open.org/

Terminologies XML

- Fondamentalement, une terminologie est la **liste des termes** (**grammaire**) utilisés pour communiquer.
- La **sémantique** détermine les contraintes d'utilisation d'un terme en association avec d'autres.
- Elle spécifie également une **taxinomie** (structure de classification) des concepts abstraits et de la spécialisation des termes.
- Si la grammaire et la taxinomie contribuent à la **sémantique** des éléments d'une terminologie, l'**ontologie** rend compte de relations et de contraintes plus riches entre ces termes.
- La plupart des terminologies XML actuelles ne répondent que partiellement à de tels concepts : certaines se limitent simplement à des listes de termes.

UML et XML

- Il faut une représentation de la **terminologie** qui soit à la fois lisible par les humains et exploitable pas les applications.
- Les modèles UML sont en mesure de rendre compte de la sémantique du modèle d'un domaine applicatif et en standardisant la notation graphique permettre l'utilisation par des lecteurs humains.
- La liaison entre UML et XML se fera par la spécification **XMI** (*XML Metadata Interchange*), elle même reposant sur l'utilisation de la spécification **MOF** (*Meta Object Facility*) qui permet de définir la langage UML.
- Il existe aussi le format **UXF** (*UML eXchange Format*)



II . Document XML

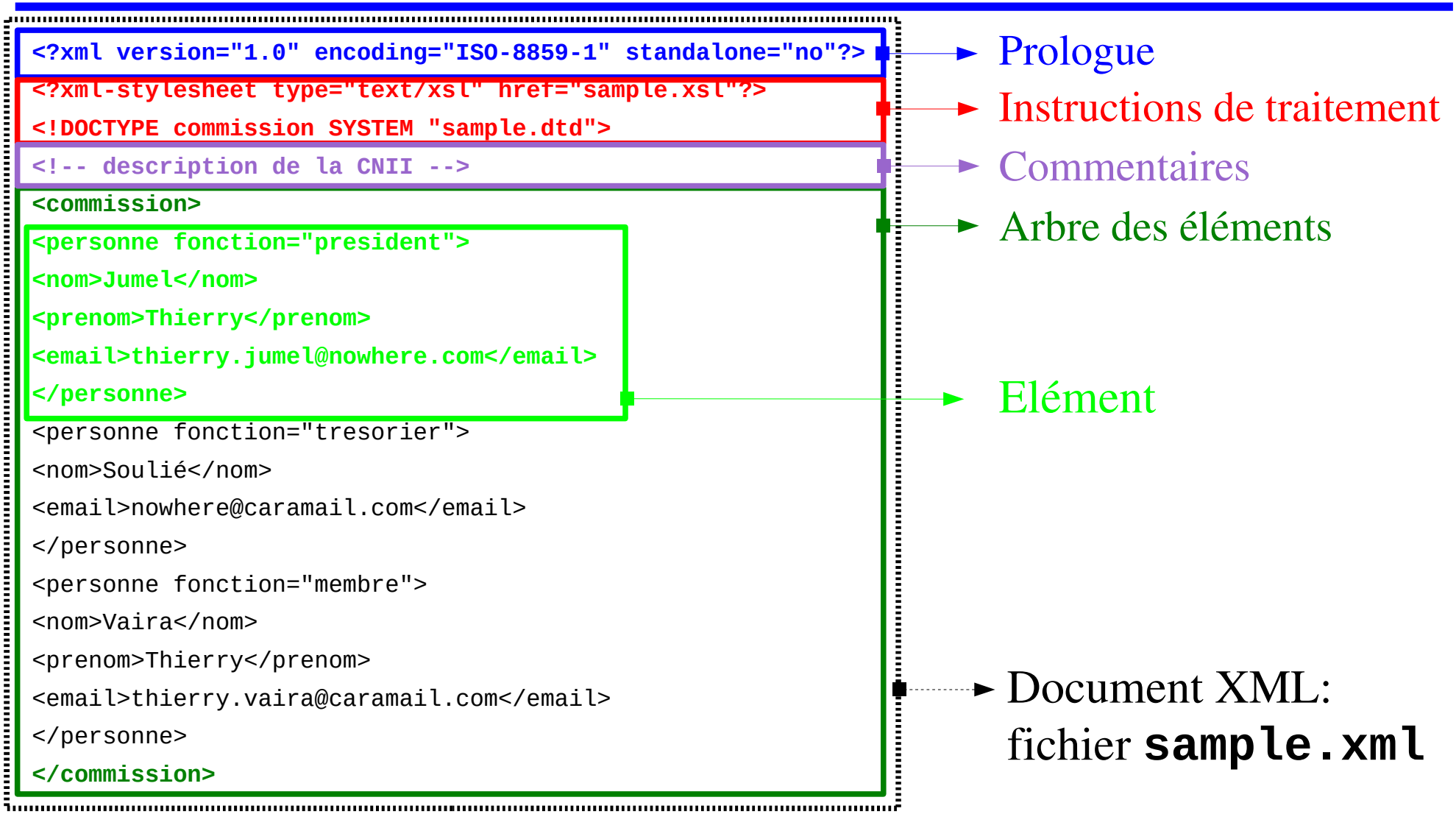
- Composition
  - Exemple
  - Prologue
- Instructions de traitement
  - Arbre des éléments
  - Les éléments
  - Les attributs
  - Les entités
- Types de qualification (bien formé et valide)
  - Outils de validation
- Les espaces de nommage

Composition d'un document XML

- Un document XML est structuré en trois parties :
  - **un prologue** : qui indique la version XML utilisée, le jeu de caractères (*encoding*) et la présence d'une DTD interne ou externe (*standalone*).
  - **des instructions de traitement** (IT) ou "*processing instruction*" (PI) : permettent aux documents de contenir des instructions relatives à des applications qui traiteront le document (par exemple : **`<?xml-stylesheet type="text/xsl" href="style.xsl"?>`**)
  - **l'arbre des éléments** : le contenu
- Par ailleurs, une feuille XML peut contenir **des commentaires** de la même manière qu'un document HTML : **`<!-- un commentaire -->`**

Document XML = prologue + [instructions] + arbre

Exemple de document XML



Prologue

- Un document XML commence toujours par un prologue.
- Un prologue contient systématiquement une déclaration qui spécifie la version de XML utilisée : **`<?xml version="1.0" ?>`**
- Il existe également deux autres attributs : *encoding* et *standalone*.
  - L'attribut ***encoding*** : permet de définir le type de codage du jeu de caractères utiliser dans la feuille XML. Les types les plus utilisés sont : UTF-8, UTF-16, ISO-8859-1 (alphabet français)
  - L'attribut ***standalone*** détermine si une DTD est utilisée (="no") ou non (="yes").
- Valeurs par défaut des attributs : **`encoding="UTF-8" standalone="yes"`**
- Un prologue complet : **`<?xml version="1.0" encoding="UTF-8" standalone="no" ?>`**

Instructions de traitement

- Par la suite, le prologue peut contenir différentes instructions relatives aux données de traitement :
  - les paramètres relatifs à la mise en page par une **feuille de style** de type XSL (*eXtended Stylesheet Language*) ou CSS (*Cascading Style Sheet*)
  - la déclaration d'un fichier externe qui contiendrait les données de mise en page (XSL ou CSS)
  - Exemple: `<?xml-stylesheet type="text/xsl" href="sample.xsl" ?>`
  - la définition d'une **DTD** (forme interne)
  - la déclaration d'un fichier externe contenant contenant la DTD (forme externe)
  - Exemple: `<!DOCTYPE commission SYSTEM "sample.dtd">`

Arbre des éléments

- Un document XML est composé d'éléments désignés par des balises.
- Ces éléments, qui forme le contenu de la feuille, sont structurés sous la forme d'un arbre (arbre d'éléments).
- Les éléments doivent être rattachés à un élément racine (ou élément document ou *root*).
- Donc, un document XML possède toujours un et un seul élément racine qui pourra contenir d'autres éléments afin de créer l'arbre des données (structure arborescente).
- Les éléments sont aussi appelés **noeuds** ou *nodes* (par référence à la théorie des graphes).

Les éléments

- Les éléments sont définis par des balises qui ont trois formes possibles :
  - <element> : balise d'ouverture
  - </element> : balise de fermeture
  - <element /> : balise vide (ou atomique)
- Les différentes balises d'ouverture et de fermeture ne doivent pas se recouvrir mais s'appartenir ou se différencier.
- Le contenu du document XML est composé de données analysables ou non :
  - Les données analysables sont des caractères, formant éventuellement du texte.
  - Les données non analysables sont des ressources qui peuvent contenir tout type de données (fichier, image).

Les attributs

- Un attribut est une paire clé/valeur écrit sous la forme cle="valeur", ainsi une balise affectée d'un attribut aura la syntaxe suivante: **`<balise cle="valeur">`**
- L'ordre des attributs dans une feuille XML n'a aucune importance.
- Les attributs peuvent être de plusieurs types (voir DTD ou XML Schema).
- La nom d'un attribut est toujours en minuscules.

Les entités

- On distingue deux types d'entités suivant le contenu de l'entité :
  - **entité analysable** (*parsed entity*) : elles ne peuvent contenir que des caractères (texte). Le contenu est appelé texte de remplacement.
  - **entité non analysable** (*unparsed entity*) : sont des ressources qui peuvent contenir tout type de données (fichier, images).
- Ces deux groupes peuvent être encore décomposés :
  - Si l'entité est destinée à être utilisée dans le contenu du document, on parlera d'**entité générale** : **`<!ENTITY nom valeur>`**
  - Si l'entité est destinée à se faire dans la DTD, on parlera d'**entité paramètre** : **`<!ENTITY %nom valeur>`**

Les entités internes et externes

- D'autre part, une entité peut être :
  - **interne** : sa portée est limitée au document XML qui l'a contient. Sa définition est la même qu'une entité générale : **`<!ENTITY exemple "Exemple">`**

    Donc, à chaque fois que **`&exemple`** sera rencontré dans le document XML, il sera remplacé par **`"Exemple"`**.
  - **externe** : permet de définir une valeur se situant dans un autre document que celui dans laquelle elle est appelée.

Utilisation des entités externes

- Il y a plusieurs utilisations possibles des entités externes :
  - l'entité externe est de type XML :

```
<!ENTITY nom SYSTEM "http://www.foo.com/exemple.xml">
```

  - l'entité externe est de type non analysable, par exemple une image gif :

```
<!ENTITY image SYSTEM "http://www.foo.com/images/foo.gif" NDATA gif>
```

- Pour cela on utilise le mot clé **NDATA**. Par ailleurs, comme nous faisons appel à une donnée non textuelle, nous devons indiquer à XML quelle application devra être utilisée avec cette entité.
- C'est le rôle des notations : `<!NOTATION gif SYSTEM "/usr/local/bin/display">`
- L'utilisation sera :

```
<exemple>
<logo src="image">
```

Types de qualification

- Un document XML supporte deux types de qualification :
  - **bien formé** = respecte la syntaxe XML (obligatoire)
  - **valide** = respecte la structure définie par une DTD (facultatif)

Document bien formé

- Pour être jugé « bien formé », un document doit respecter les règles suivantes :
  - possèder un prologue conforme
  - imbriquer correctement les balises
  - toutes les balises doivent être fermées (<balise/>)
  - les valeurs des attributs doivent être entre guillements
  - le nom des attributs est écrit en minuscules
  - le document doit posséder un seul élément racine (*root*)
- **Un document XML doit toujours être bien formé**. S'il n'est pas bien formé, par définition, ce n'est pas un document XML !

Document valide

- Le document sera jugé valide lorsqu'il possède et respecte sa DTD (*Definition Type Document*).
- La DTD permet de décrire la structure et l'ordonnancement employés dans un document XML (= grammaire)
- La structure fournie par une DTD décrit les balises et les attributs valides.

Outils de validation

- On peut vérifier la validité d'un document XML en utilisant un outil local ou web :
  - **`nsgmls -s -wxml sample.xml`**
  - **`rxp -V sample.xml`**
  - **`xmllint -valid sample.xml`**
  - http://validator.w3.org/
  - http://www.cogsci.ed.ac.uk/~richard/xml-check.html
  - http://www.stg.brown.edu/service/xmlvalid/
  - http://frontier.userland.com/stories/storyReader$1092

Les espaces de nommage (1)

- Les "espaces de nommage" (*XMLNamespaces*) sont spécifiés dans une recommandation du W3C.
- Ils permettent :
  - de mélanger du vocabulaire XML provenant de plusieurs grammaires
  - d'identifier de manière unique les balises XML.
  - et donc d'éviter les collisions au niveau de l'interprétation du vocabulaire (même terme et sens différent)

Les espaces de nommage (2)

- Le mécanisme des "namespaces" consiste à utiliser des préfixes, par exemple "html:title" et "book:title" pour éviter l'ambiguité du terme title.
- Les espaces de noms sont utilisés aussi bien pour les éléments que pour les attributs.
- La définition de l'espace de noms se fait à l'aide de l'attribut **`xmlns`** dans la balise racine (*root*) du document :

```
<racine xmlns:nom="URI" ...>
```

- La valeur de l'attribut est une URI (*Universal Ressource Identifier*) qui définit l'emplacement du "namespace" utilisé (fictif ou non, référence à l'organisme garant du vocabulaire en question).

III . DTD (*Definition Type Document*)

- Introduction
  - Rôle
- DTD interne, externe, publique
- Composants
- Exemple

DTD (*Definition Type Document*)

- Concept provenant de SGML
- Très utilisé en XML
- Une DTD n'est pas écrite en XML (autre langage)
- Nouvelle proposition : **XML Schema**
- Pour associer à un document une DTD :

```
<?xml version="1.0" standalone="no">
<!DOCTYPE élément-racine SYSTEM "exemple.dtd">
```

Rôle d'une DTD

- La DTD permet de décrire la « grammaire » suivante :
  - les noms de balises autorisés (= élément)
  - un ordre autorisé pour les balises
  - quel élément contient quel élément
  - quels éléments sont optionnels (= occurrence)
  - quels sont les attributs autorisés

DTD interne, externe et publique

- **Interne** au fichier XML (1)

```
<!DOCTYPE commission [ ...instructions... ]>
```

- **Externe** au fichier, dans un fichier "voisin" (2)

```
<!DOCTYPE commission SYSTEM "(http://x.y/)exemple.dtd">
```

- **Externe** au fichier, **publique** (sur le web) (3)

```
<!DOCTYPE commission PUBLIC "ident-connu" "URL-sinon">
```

- On peut combiner 1+2 ou 1+3
- Avantage de la forme externe :
  - Elle permet la réutilisation pour d'autres documents XML
- Avantage de la forme externe publique :
  - Elle définit un standard pour certains documents (CV, facture, publications, ...)

Composants d'une DTD

- Déclaration des noms de balises autorisés :

```
<!ELEMENT le_nom (le_contenu)>
```

- Déclaration du contenu :
  - soit l'ordonnancement des éléments (les éléments fils)
  - soit ce que contient l'élément (type)
- Déclaration des attributs (noms et type) :

```
<!ATTLIST élément attribut_1 type_1 valeurDéfaut_1 ...>
```

Exemple DTD

élément racine

contient un ou plusieurs élément **personne**

```
<!ELEMENT commission (personne+)>
<!ELEMENT personne (nom,prenom?,email*)>
<!ATTLIST personne fonction (president | tresorier | membre) #REQUIRED>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT prenom (#PCDATA)>
<!ELEMENT email (#PCDATA)>
```

l'élément **personne** contient les éléments ordonnés suivants :

- un seul **nom**
- aucun ou un **prenom** (?)
- aucun, un ou plusieurs **email** (*)

l'élément **personne** possède un attribut **fonction** qui doit obligatoirement (`#REQUIRED`) prendre la valeur :

president OU **tresorier** OU **membre**

les données contenues dans l'élément **email** ne seront composées que de texte et rien d'autre

PCDATA (*Parsed Character DATA*)

IV . Mise en page - Présentation

- Solutions
- XSLT (*eXtensible StyleSheet Language Transformation*)
- Fonctionnement et principe
- Exemple
- XSL-FO (*XSL Formatting Objects*)

Solutions

- XML étant un format de description de données (et non de représentation), la mise en page doit être assurée par un langage tiers.
- Il existe plusieurs solutions :
  - **CSS** (*Cascading StyleSheet*) très utilisé actuellement, étant donné qu'il s'agit d'un standard qui a déjà fait ses preuves avec HTML
  - **XSL** (*eXtensible StyleSheet Language*), un langage de feuilles de style extensible développé spécialement pour XML.
- En fait XSL est une spécification composée de deux parties :
  - **XSLT** (*eXtensible StyleSheet Language Transformation*) : langage permettant de transformer un document XML en un autre document XML (ou HTML, WML, texte)
  - **XSL-FO** (*eXtensible StyleSheet Formatting Objects*) : un langage de traitement permettant de transformer le document XML en d'autres formes de documents plus avancés (le PDF par exemple)

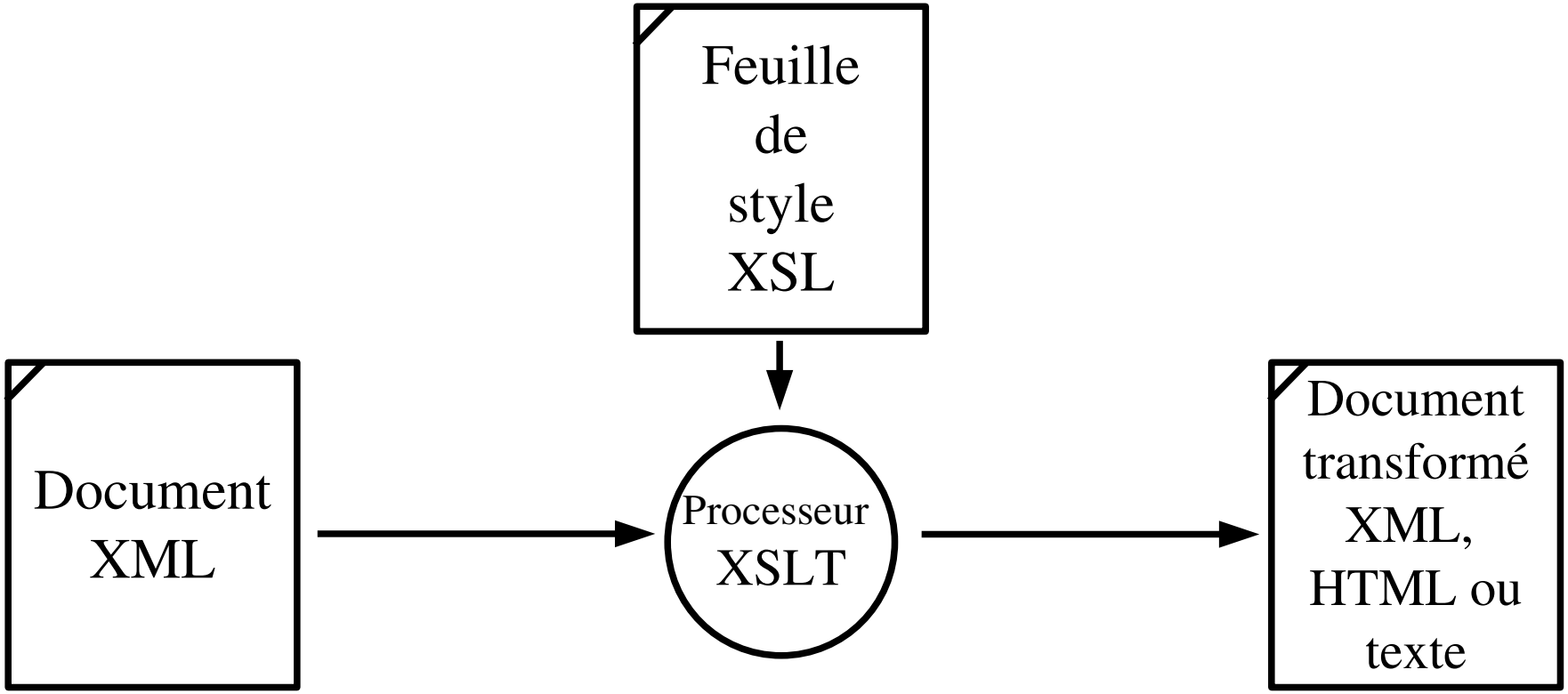
XSLT (*eXtensible StyleSheet Language Transformation*)

- XSLT est un langage contenant 35 éléments définis par le W3C (1.0 en 1999).
- XSLT permet : la déclaration, la transformation et le formatage de données.
- XSLT transforme une structure de données en une autre correspondant au format de sortie désiré et indiqué au niveau de la feuille de style.
- Pour faire cette transformation, XSLT utilise des feuilles de style contenant des règles appelées *template rules* (règles de gabarit) : chaque règle (de *template*) définit un traitement à appliquer à l'élément qui lui correspond (*match*).
- Différence avec CSS: XSL permet de réaliser un traitement en manipulant le contenu.

Fonctionnement de XSLT

- 1° étape: création du « *source tree* » en mémoire après *parsage* du document XML
- 2° étape: création du « *result tree* » après les transformations contenues dans les règles du document XSLT
- L'extraction est réalisée à l'aide de *pattern matching* et de *templates* :
  - Les *patterns* représentent les informations à repérer dans l'arbre correspondant au document XML.
  - Les *templates* correspondent aux traitements à y appliquer.

Principe


Feuille de style XSL
Document XML
Processeur XSLT
Document transformé XML, HTML ou texte

Exemple

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:template match="/">
<html><body>
<xsl:for-each select="commission/personne">
<xsl:value-of select="nom"/> - <xsl:value-of select="prenom"/>
<br />
Email : <xsl:value-of select="email"/><br />
</xsl:for-each>
</body></html>
</xsl:template>
</xsl:stylesheet>
```

permet de définir une règle de *template* qui pourra être appliqué à l'élément repéré par *match*. Ici le **/** indique que tous les éléments à partir de la racine sont concernés.

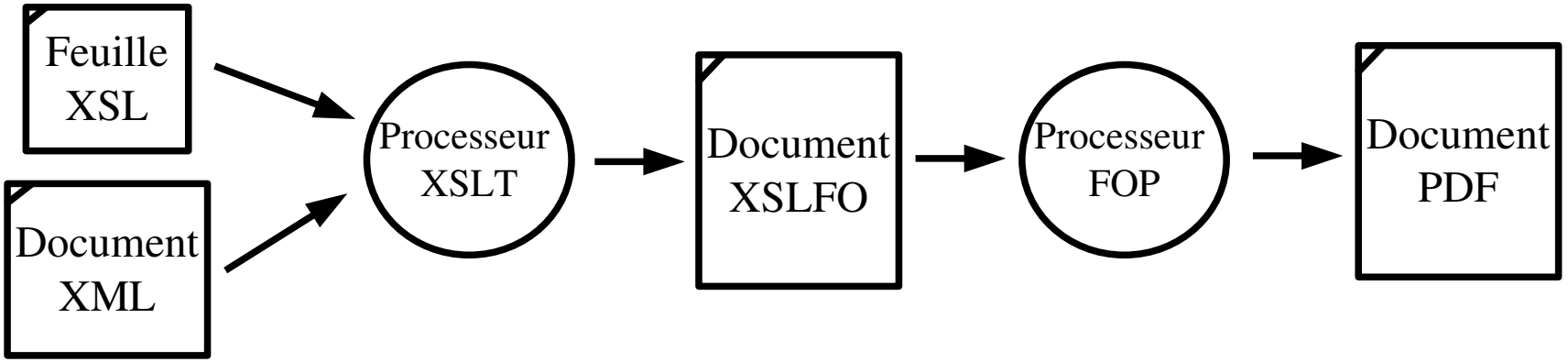
permet d'itérer au travers de chaque élément d'un ensemble d'éléments.
On utilise la notation **XPath** dans *select*.

permet de récupérer la valeur d'un élément et de l'écrire (ou l'afficher)

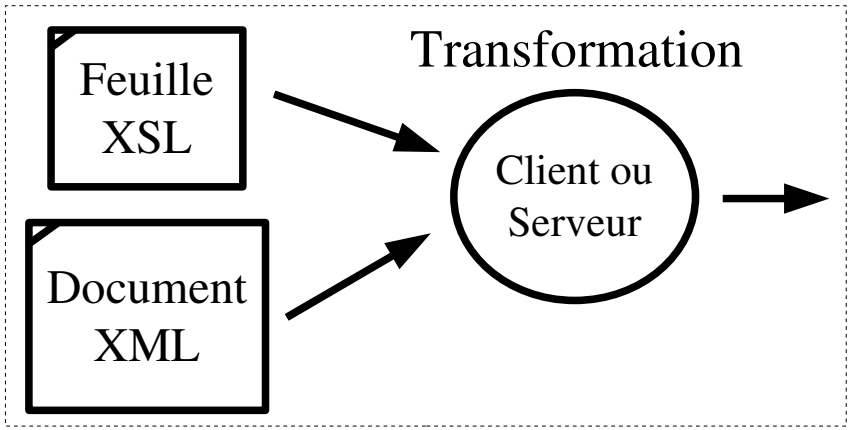
marque le début et la fin d'une feuille de style qui contient tous les éléments XSLT.
La définition du *namespace* est importante (et indispensable)

XSL-FO (*XSL Formatting Objects*)

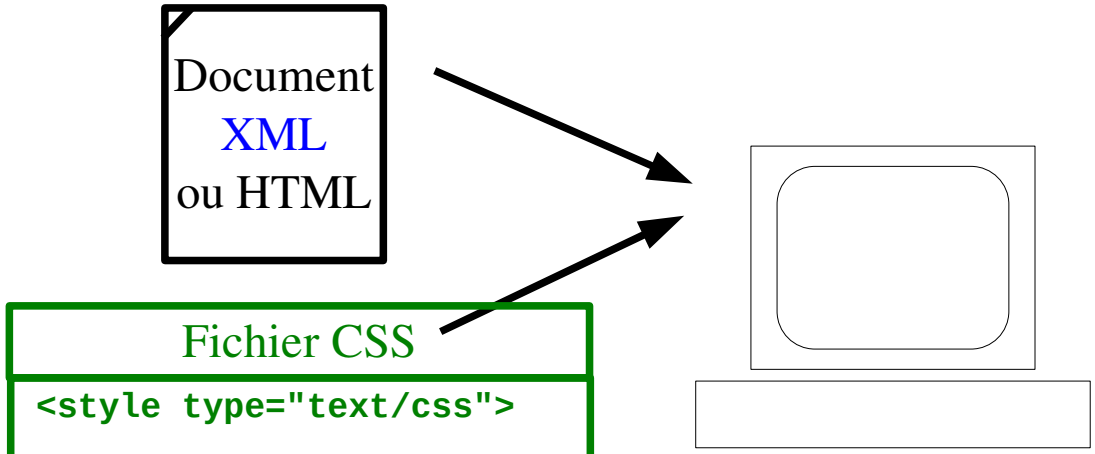
- XSL-FO permet d'exprimer de manière très précise le rendu d'un document en gérant : la pagination d'un document, les notes de bas de pages, les marges, ... Il est possible de préciser avec exactitude l'emplacement des différents objets sur la page, les polices de caractères, l'affichage de tableaux, etc.
- Un document XSLFO est habituellement générer en utilisant XSLT.
- Ce langage doit sa renaissance à FOP (*Formating Object Processor*), un outil, développé dans le cadre du projet Apache, qui permet à partir d'un document XSL-FO de générer un document PDF (format Acrobat de la société Adobe). Il représente une bonne solution pour l'édition de factures ou de contrats sur le Web.



CSS (*Cascading StyleSheet*)




```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<?xml-stylesheet href="sample.css" type="text/css"?>
<commission>...</commission>
```



Fichier CSS

```
<style type="text/css">
commission , personne {}
nom {color: blue;}
prenom {font-size: 10pt ;}
email {color: red ;}
</style>
```

V . Manipulation des documents XML

- Différentes techniques
- Les parseurs XML
- Les parseurs de type *callback*
- Les parseurs de type arbre
- Le *databinding*
- Les langages de requêtes simples ou de type SQL

Différentes techniques

- les API permettant de lire et modifier le contenu des documents XML
- les langages de requête permettant d'extraire des informations de documents XML, à la manière de SQL pour les bases de données

Les parseurs XML

- Il existe principalement trois approches pour accéder à un document XML :
  - Les parseurs de type *callback* (SAX)
  - Les parseurs de type arbre (DOM)
  - Le *databinding* (*Java Specification Request* de Sun)

Les parseurs type *callback*

- Le parseur parcourt le document XML et envoie un événement à chaque élément rencontré.
- Le programmeur définit le traitement à effectuer pour chaque événement, et peut ainsi construire sa propre structure de données.
- **SAX** (*Simple API For XML*) est l'API la plus connue se basant sur cette technique. Elle est indépendante du langage de programmation

Les parseurs type arbre

- Le parseur utilise la structure arborescente d'un document XML.
- Le parseur charge le document XML en mémoire, sous la forme d'un arbre.
- **DOM** (*Document Objet Model*) est l'API la plus connue répondant à cette approche. On parcourt et modifie l'arbre à l'aide des interfaces définies par le W3C. Cette API est aussi indépendante du langage de programmation.

Le *databinding*

- Cette méthode est une *Java Specification Request* de **Sun**, qui permet de charger un document XML en mémoire sous la forme d'un **objet que l'on manipule en utilisant des accesseurs**.
- On sauvegarde ensuite le document en sérializant l'objet en XML. Cette approche permet de manipuler uniquement des objets Java. Les outils permettant de faire du *data binding* sont assez récents : Castor de Exolab, Zeus de Enhydra, etc ...

Les langages de requêtes

- Les langages de requêtes permettent d'extraire des informations de documents XML, à la manière de SQL pour les bases de données.
- Il y a de nombreuses initiatives pour définir un langage qui réponde à tous les besoins de recherche dans un document XML : XQL, XPath, Quilt, XML-QL, XQuery, YATL, Lorel, ...
- Il existe en fait deux types de langages :
  - les langages de requêtes simple (XQL et XPath)
  - les langages de requêtes 'type SQL' (XML-QL, XQuery, ...)

Les langages de requêtes simples

- Ces langages se basent sur la structure du document XML, les plus célèbres sont **XQL** et **XPath**.
- *Exemple en XQL et XPath :*

```
/commission/personne[nom="Thierry Vaira"]
```

- Ces langages ne permettent pas de construire de nouveaux documents XML, mais uniquement de les consulter.

Les langages de requêtes type SQL

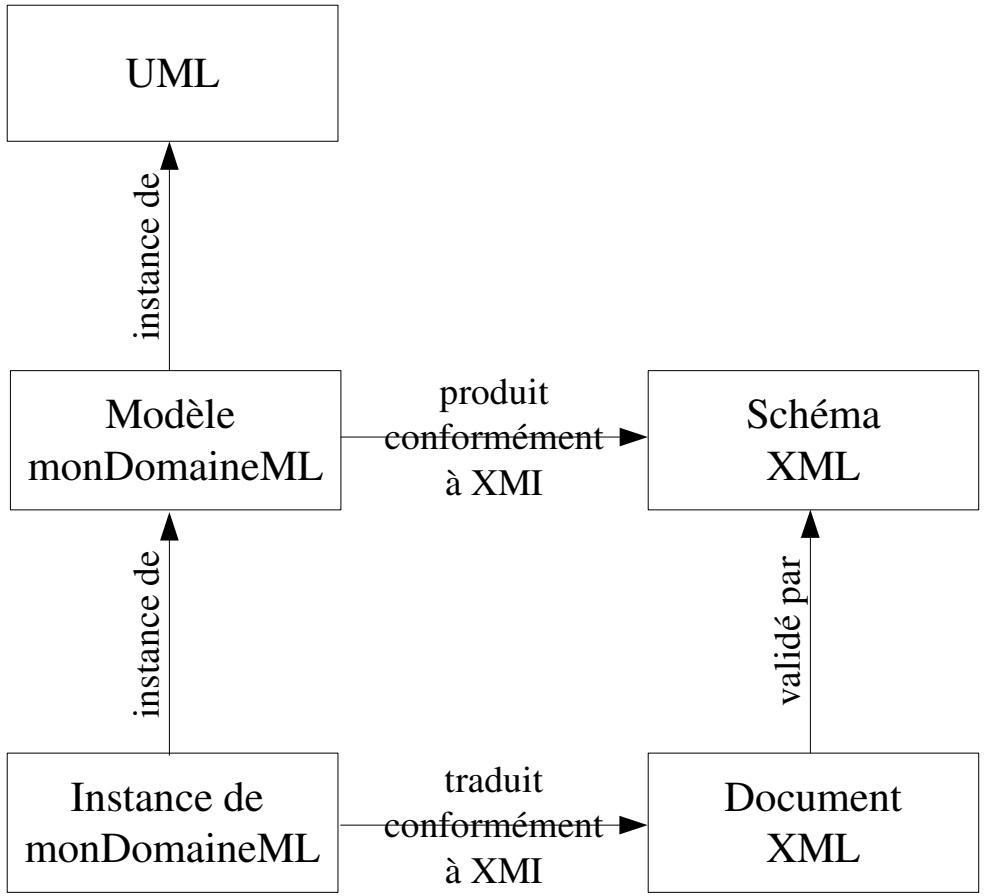
- Les langages 'type SQL' permettent de construire des requêtes similaires aux requêtes SQL (`SELECT ... FROM ... WHERE ...`), adaptées à la structure des documents XML.
- Parmi ces langages on trouve **XML-QL**, **Quilt**, **XQuery**, **Lorel** et **YaTL**.
- Ces langages permettent généralement de construire de nouveaux documents XML en construisant le format de sortie des requêtes.

VI . UML et XML

- XMI et MOF
- Postulats de XMI
- Principe de génération des schémas XML
- Les packages UML
- UML/XML/DTD par l'exemple

XMI (*XML Metadata Interchange*) et **MOF** (*Meta Object Facility*)

- La liaison entre UML et XML se fera par la spécification **XMI** (*XML Metadata Interchange*), elle même reposant sur l'utilisation de la spécification **MOF** (*Meta Object Facility*) qui permet de définir la langage UML.
- XMI décrit une approche rigoureuse pour la génération de DTD à partir de définition de métamodèles et pour la génération de documents XML à partir de modèles.
- XMI a été rédigé pour permettre l'échange de modèles UML entre AGL de différents éditeurs. Evidemment, XMI a un champ d'application bien plus vaste ...



Principaux postulats de XMI

- La traduction est automatisée
- La traduction est reproductible
- La multiplicité et l'odre des éléments ne sont pas appliquées
- Echange de données avant tout
- La compacité n'est pas une priorité

Principe de génération

Un modèle UML entier est représenté par une seule DTD

- **Espaces de noms** : sans ou un préfixe dans tous les noms d'éléments
- **Package** : un élément XML simple est généré pour chaque package et il peut inclure les classes et les sous-packages
- **Unicité des noms d'éléments** : on préfixe chaque attribut et chaque rôle d'association UML par le nom de la classe suivi d'un point : Classe1.role2 ou PackageA.Classe1.role2
- **Eléments ou attributs** : on génére un élement XML pour chaque attribut UML et pour chaque rôle d'association ou on génére un attribut XML pour chaque attribut UML ayant un type de données primitif avec une multiplicté maximale de 1
- **Contraintes de multiplicité** : toute multiplité des éléments est non bornée (*unbounded*)
- **Héritage** : il n'y a aucune prise en charge dans les DTD, on reproduit donc dans chaque sous-classe tous les attributs et rôles des super-classes (= copie descendante)
- **Orde des éléments** : tous les contenus sont non ordonnés (DTD souple) ou les éléments apparaissent dans l'ordre spécifié dans UML : d'abord les attributs puis les rôles (DTD stricte)
- **Type de données** : il n'y a aucune prise en charge dans les DTD (on utilise donc #PCDATA pour les éléments et CDATA pour les attributs)

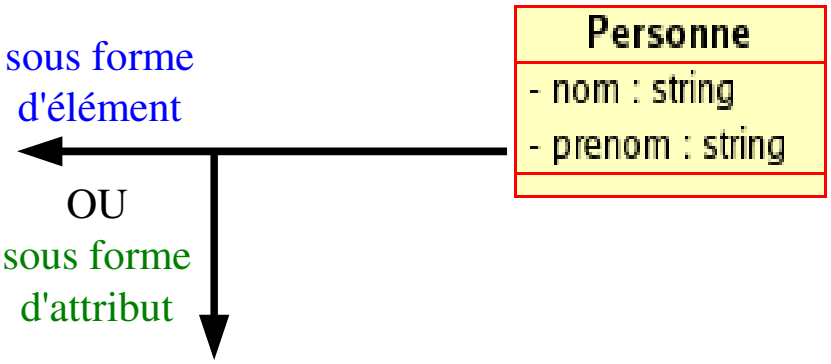
Les packages UML

- UML est caable de définir une hiérarchie de packages spécifiant des espaces de noms pour les éléments d'un modèle.
- A la différence d'UML, les espaces de noms ne gèrent pas la hiérarchie des packages. Un espace de noms XML se résume à l'apposition d'un préfixe.
- Le préfixe permet de regrouper des termes issus de différentes terminologies dans un même document : unique objectif du standard *Namespace* actuel.

UML/XML par l'exemple
Classes

- **Chaque instance d'une classe UML produit un élément XML.**
- En XML, un élément sert de conteneur aux attributs et aux sous-éléments.
- Comme en UML, l'élément XML (l'instance de la classe) sert de conteneur aux attributs et aux rôles d'association.
- L'élément XML d'une instance de **Personne** sera représenté :

```
<Personne>
    <nom>VAIRA</nom>
    <prenom>Thierry</prenom>
</Personne>
```



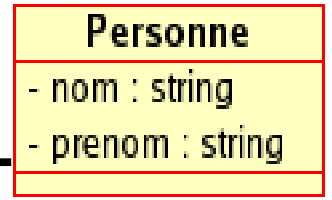
ou `<Personne nom="VAIRA" prenom="Thierry"/>`

UML/DTD par l'exemple
Classes

- **Chaque classe UML définie dans le modèle produit un ensemble standard de déclarations de types d'éléments et d'attributs XML.**
- *Remarque:* les attributs et rôles d'association UML peuvent devenir soit seulement des éléments XML (DTD stricte) soit à la fois des éléments et des attributs XML (DTD souple).
- La DTD produite à partir de la classe **Personne** sera :

```
<!ELEMENT Personne (Personne.nom | Personne.prenom)*>
<!ELEMENT Personne.nom (#PCDATA)>
<!ELEMENT Personne.prenom (#PCDATA)>
```

sous forme d'élément



OU sous forme d'attribut

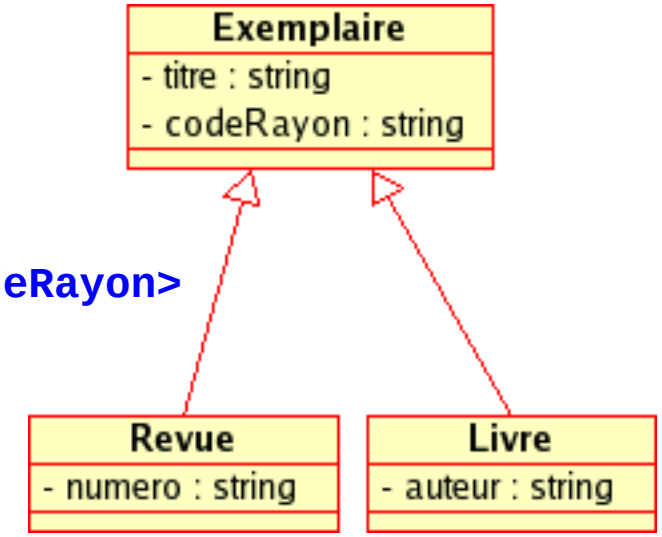
ou `<!ATTLIST Personne prenom CDATA #IMPLIED>`

UML/XML par l'exemple
Héritage

- Les standards XML actuels ne proposent pas de mécanisme intégré de représentation de l'héritage.
- Une DTD ne peut incoporer l'héritage aux définitions des éléments mais elle peut représenter la strutcure d'**agrégation** des éléments inclus dans d'autres éléments.
- Par conséquent, on spécifie l'héritage par **copie descendante** pour les attributs, les associations et les compositions.
- L'élément XML d'une instance de **Livre** sera représenté :

```
<Livre>
    <Exemplaire.titre>XML</Exemplaire.titre>
    <Exemplaire.codeRayon>L-112</Exemplaire.codeRayon>
    <Livre.auteur>Y.L.</Livre.auteur>
</Livre>
```

ou `<Livre codeRayon="L-112"/>...</Livre>`



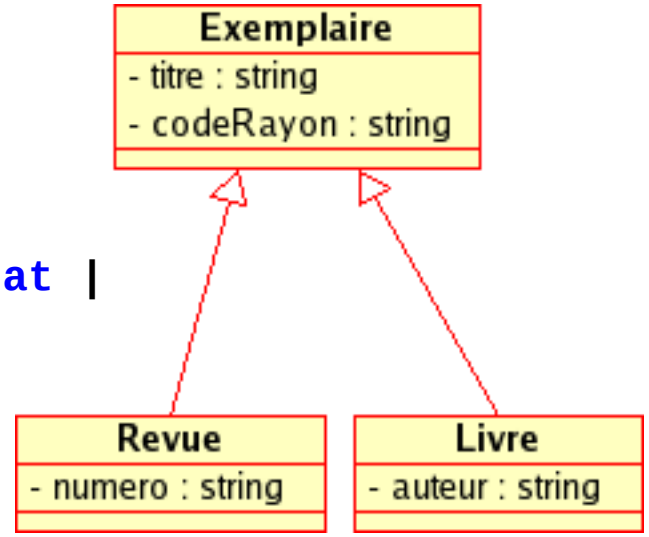
UML/DTD par l'exemple
Héritage

- Comme les DTD ne gèrent pas l'héritage entre types d'éléments, la spécification XML exige que tous les attributs et les rôles d'association de toutes les superclasses soient inclus dans les déclarations de sous-classes.
- Dans le cas de **Livre**, le modèle de contenu inclut par conséquence tous les **éléménts** et les **attributs** de contenu de la super-classe **Exemplaire** :

```
<!ELEMENT Exemplaire (Exemplaire.titre | Exemplaire.etat)*>
<!ELEMENT Exemplaire.titre (#PCDATA)>
<!ATTLIST Exemplaire codeRayon CDATA #REQUIRED>
<!ELEMENT Exemplaire.etat (#PCDATA)>


<!ELEMENT Livre (Exemplaire.titre | Exemplaire.etat |
  Livre.auteur)*>
<!ELEMENT Livre.auteur (#PCDATA)>
<!ATTLIST Livre codeRayon CDATA #REQUIRED>
```

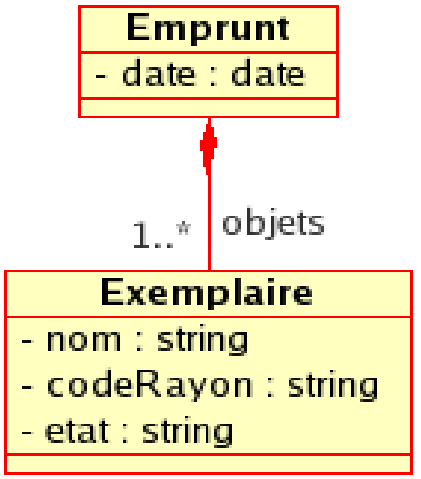
...



UML/XML/DTD par l'exemple
Association

- Les rôles d'association, composition et agrégation UML peuvent devenir soit seulement des éléments XML (DTD stricte) soit à la fois des éléments et des attributs XML (DTD souple).
- La **DTD** produite à partir de la classe **Personne** sera :

```
<!ELEMENT Emprunt (Emprunt.objets)*>
<!ATTLIST Emprunt date CDATA #REQUIRED>
<!ELEMENT Emprunt.objets (Exemplaire)*>
```



- Et un exemple de fichier **XML** valide :

```
<Emprunt date="10102003">
        <Emprunt.objets>
            <Exemplaire codeRayon="E-666">
                <Exemplaire.nom>Un truc</Exemplaire.nom>
                <Exemplaire.etat>neuf</Exemplaire.etat>
            </Exemplaire>
        </Emprunt.objets>
</Emprunt>
```

Liens

- Portail d'information sur XML: http://www.xml.org/
- Portail francophone: http://xmlfr.org/
- Pour les développeurs XML: http://xmlhack.com/
- Consortium de développement de technologies autour de XML: http://www.oasis-open.org/
- PerfectXML: http://www.perfectxml.com/
- Présentation de XML: http://lsoron.free.fr/dossiers/xml/xml.html
- Apache et XML: http://xml.apache.org/
- Le magazine de XML: http://www.xmlmag.com/
- Dr Dobbs et XML: http://www.ddj.com/topics/xml/
- Tutoriaux et conseils: http://www.xml-zone.com/
- IBM et XML: http://www-106.ibm.com/developerworks/xml/
- XML Francophone: http://www.chez.com/xml/
- Regroupement de news: http://www.mutu-xml.org/
- Comment ça marche: http://www.commentcamarche.net/xml/
- Site de David Carlson (auteur du livre "Modelisation d'applications XML avec UML) : http://www.xmlmodeling.com/
- Autres liens: http://www.chez.com/xml/liens/index.htm
- Cours XML:
- http://www.laltruiste.com/coursxml/
- http://magali.contensin.free.fr/html/XML/
- http://xml.ladoc.net/
- Liste d'outils XML:
- http://mapageweb.umontreal.ca/marcoux/grds/outils-xml/
- http://www.garshol.priv.no/download/xmltools/
- http://WDVL.com/Software/XML/
- http://www.xmlsoftware.com/
- http://www.oasis-open.org/cover/xml.html#xmlSoftware