Introduction au langage XML
3: XSLT et programmation avec XML (modèle DOM et SaX)

XSL : Introduction

- XSLT: e**X**tensible **S**tylesheet **L**anguage
- *Version 1.0*
- W3C Recommandation du 16 novembre 1999
- XSL est l'héritier de DSSSL (scheme (lisp) manipulant du SGML).
- La communauté DSSSL souhaitait un langage permettant le traitement des sources pour imprimer différentes versions:
  - voulait une puissance d'un langage de programmation sans prendre en compte l'interactivité sur la version finale (retour des évènements?)

XSL : caractéristiques

- Un langage de manipulation d'arbres
  - Change un dialecte XML en un autre
  - Langage déclaratif basé sur la recherche de motifs dans un arbre
  - Transformation dirigée par le document
  - Pas vraiment un langage de formatage
- Un langage de réécriture pour XML subdivisé en:
  - XSLT (Transformation) : est un langage permettant de produire un document XML ou texte a partir d'un autre document par application de règles de transformation
  - XSL-FO (Formatting Object): est une DTD XML qui définie la présentation d'un texte sur un document papier (PS, GV, PDF,DPS).
  - Utilise XPath pour la localisation des noeuds

3

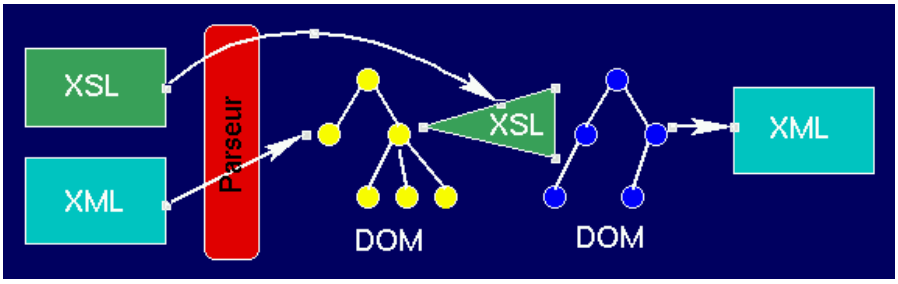
XSLT : Principe

- Un document XML + une feuille de style XSL -> processeur XSLT -> un document XML (ou autre format de sortie, HTML, ...)
  - Analyse un document XML
  - Trouve les noeuds sur l'arbre en mémoire (DOM) correspondant aux règles XSL
  - Génère un nouvel arbre DOM à partir des instructions de ces règles
  - "Imprime" le XML résultant

4

XSLT : Principe (2)

- XSL joue un rôle de pivot entre document original et celui de sortie



5

XSLT : syntaxe de la feuille de style

- Une feuille de style XSL est un document XML

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<xsl:stylesheet
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   version="1.0"  id="identifiant" extension-element-prefixes="liste"
   exclude-result-prefixes="liste" >
... déclarations optionnelles ...
... règles de transformation ...
</xsl:stylesheet>
```

6

XSLT : format de sortie

- Déclaration optionnelle du format de sortie

```
<xsl:output
method="xml|html|text|nom"
version="version"
encoding="type_encodage"
omit-xml-declaration="yes|no"
standalone="yes|no"
doctype-public="identifiant"
doctype-system="identifiant"
cdata-section-elements= "liste de nom"
indent="yes|no"
media-type="type"/>
```

  - A mettre au début du fichier (après xsl:stylesheet)

7

XSLT : format de sortie (2)

Output en HTML:

```
<xsl:output method="html"
encoding="ISO-8859-1"
doctype-public="-//W3C//DTD
HTML 4.01 Transitional//EN"/>
```

Output en XHTML

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transfor
m"  xmlns="http://www.w3.org/TR/xhtml1/strict">
<xsl:output method="xml"  doctype system=
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd"
doctype-public="-//W3C//DTD XHTML 1.0
Transitional//EN"
indent="yes"
encoding="iso-8859-1"
```

Output en SVG

```
<xsl:output
method="xml"
indent="yes"
standalone="no"
doctype-public="-//W3C//DTD SVG 1.0//EN"
Doctype-system="http://www.w3.org/TR/2001/PR-SVG 20010719/DTD/svg10.dtd"
media-type="image/svg"
```

8

XSLT : éléments du premier niveau

- Les éléments du premier niveau d'une feuille XSL

xsl:attribute-set	Définit un groupe d'attributs
xsl:decimal-format	Définit un format d'affichage pour les numériques
xsl:import	Importe d'un programme XSLT
xsl:include	Inclusion d'un programme XSLT
xsl:key	Définit une clé pour un groupe de noeuds
xsl:namespace-alias	Définit des *alias* pour certains espaces de nom
xsl:output	Indique le format de sortie (HTML, XML, texte)
xsl:param	Définit un paramètre
xsl:preserve-space	Conserve les noeuds blancs (noeuds constitué uniquement d'espaces) pour certains éléments
xsl:strip-space	Supprime les noeuds blancs pour certains éléments
xsl:template	Définit une règle XSLT
xsl:variable	Définit une variable XSLT

9

XSLT : Modularité - import et include

- XSLT fournit deux éléments de premier niveau pour intégrer des fichiers afin de constituer des programmes modulaires: xsl:import et xsl:include
- L'assemblage de plusieurs programmes peut créer des *conflits*. Au moment de l'évaluation plusieurs règles peuvent s'appliquer aux mêmes noeuds. Afin de déterminer la règle à appliquer, XSLT utilise un système de *priorités (pour xsl:import principalement)*

10

XSLT : Modularité - import et include (2)

- Importation de feuilles XSL avec gestion de la priorité :

  <xsl:import href="URL_de_la_feuille_XSL" />
  - Les règles importées sont moins prioritaires que les
  - règles définies dans la feuille courante.
  - Cette déclaration doit figurer en tête d'une feuille de style
- Inclusion de feuilles XSL (aucun impact sur la priorité)

  <xsl:include href="URL_de_la_feuille_XSL" />

11

XSLT : Modularité - import et include (3)

- Points différents entre xsl:import et xsl:include
  - dans le cas de xsl:import le processeur affecte aux règles importées une *préséance* inférieure à celle du programme importateur ; dans le cas de xsl:include, les règles importées sont traitées de ma même façon que celle des règles locales
  - xsl:import doit apparaître avant tout *autre* élément de premier niveau dans un programme XSLT; xsl:include peut apparaître au corps du programme

12

XSLT : Modularité - import et include (4)

- Exemple cours.xml

```
<?xml version='1.0' encoding="ISO-8859-1"?>
<COURS UE="TC2.1">
   <SUJET>XML et outils GL</SUJET>
   <ENSEIGNANTS>
   <!-- Enseignant responsable -->
      <NOM>Le Thanh</NOM>
      <NOM>Huet</NOM>
   </ENSEIGNANTS>
   <PROGRAMME>
      <SEANCE ID="1">Documents XML</SEANCE>
      <SEANCE ID="2">Programmation XSLT</SEANCE>
      <SEANCE ID="3">XML SCHEMA</SEANCE>
      <ANNEE>0506</ANNEE>
   </PROGRAMME>
</COURS>
```

13

XSLT : Modularité - import et include (5)

- Exemple : prog1.xsl

```
<?xml version="1.0"
   encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/199
   9/ XSL/Transform">
<xsl:strip-space elements="*"/>
<xsl:template match="/">
<html>
<head>
<title>
<xsl:value-of
   select="COURS/SUJET"/>
</title>
</head>
<body bgcolor="white">
<xsl:apply-templates/>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

14

XSLT : Modularité - import et include (6)

- *Le document HTML produit par prog1.xsl*

```
<html>
  <head>
  <META http-equiv="Content-Type"
  content="text/html; charset=UTF-8">
  <title>Publication XSLT</title>
  </head>
  <body bgcolor="white"></body>
</html>
```

15

XSLT : Modularité - import et include (7)

- Exemple principal.xml

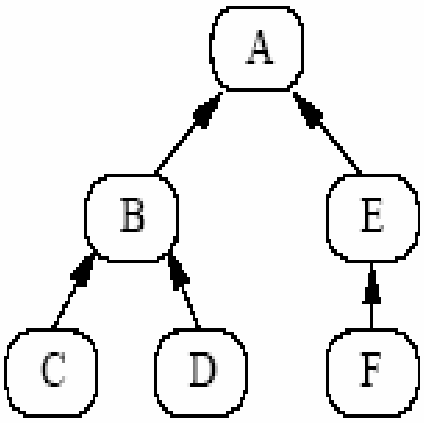
```
<?xml version="1.0" encoding="ISO-8859-
    1"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL
    /Transform">
<xsl:import href=« prog1.xsl"/>
<xsl:template
    match="COURS/ENSEIGNANTS">
<ol><xsl:apply-templates
    select="NOM"/></ol>
</xsl:template>
<xsl:template match="NOM">
<li><xsl:value-of select="."/></li>
</xsl:template>
</xsl:stylesheet>
```

- Ce programme qui affiche la liste des enseignants du cours :
  - Il commence par importer le fichier *prog1.xsl*, ce qui dispense de définir une règle s'appliquant à la racine du document.
  - Ensuite on *redéfinit* les règles s'appliquant au sous-arbre des enseignants pour produire une liste avec les noms.
  - Ces règles ont une préséance supérieure à celles du fichier importé qui ne produisaient aucun résultat.

16

XSLT : Modularité - import et include (8)

- La règle de préséance se généralise assez simplement à l'importation de plusieurs documents de la manière suivante :
  - si un document D1 est importé *avant* un document D2, alors *toutes* les règles de D2 ont une préséance supérieure à celles de D1
  - si une règle r1 a une préséance supérieure à r2, elle-même supérieure à r3, alors r1 a une préséance supérieure à r3 (transitivité)



17

XSLT : Règle de transformation

- Le corps d'une feuille de style (stylesheet) est une succession de règles de réécriture: les *xsl:template*
  - partie gauche: motif qui identifie la partie du document sur laquelle va s'appliquer la règle (XPath)
  - partie droite: description d'une action qui définit la transformation à appliquer sur un noeud
  - une règle "template" permet de sélectionner un noeud de l'arbre XML pour un traitement (transformation)
  - "match = <expression XPath>" définit le noeud à sélectionner
  - Les noeuds peuvent être définis par un chemin complet (pour définir des contextes)

18

XSLT : Règle de transformation (2)

- Exemple :
  - <xsl:stylesheet>
    <xsl:template match="/">
    [action]
    </xsl:template>
    <xsl:template match="livre">
    [action]
    </xsl:template>
    ...
    </xsl:stylesheet>

19

XSLT : Règle de transformation (3)

- Déclaration générale du format de template :

  <xsl:template
  match="pattern_XPATH"
  mode="mode" priority="nombre" >
  ... element(s) de remplacement ...
  </xsl:template>
- Le pattern XPATH est une ou plusieurs expressions XPATH connectées par un ou « | »

20

XSLT : Règle de transformation (4)

- Exemple

Le fichier XML

```
<?xml version="1.0"
encoding="ISO-8859-
1"?>
<?xml-stylesheet
href=« exemple.xsl"
type="text/xsl"?>
<article>
<title>
Introduction à XLM </title>
<resume>
un tutorial pratique de XML
</resume>
<auteur>
Dupont </auteur>
</article>
```

Le fichier XSL exemple.xsl

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="page">
<html> <head> <title> <xsl:value-of select="title"/> </title> </head>
<body bgcolor="#ffffff">
<xsl:apply-templates/>
</body>
</html>
</xsl:template>
<xsl:template match="title">
<h1 align="center"> <xsl:apply-templates/> </h1>
</xsl:template>
<xsl:template match=« resume">
<p align="center"> <xsl:apply-templates/> </p>
</xsl:template>
<xsl:template match=« Auteur">
<hr /> Auteur: <xsl:apply-templates/>
</xsl:template>
</xsl:stylesheet>
```

21

XSLT : Règle de transformation (5)

- Exemple (suite)

Le "fichier" XHTML résultant que l'on désire obtenir

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML
4.0//EN" "http://www.w3.org/TR/REChtml40/strict.dtd">
<html>
<head><title>Introduction à XLM</title></head>
<body bgcolor="#ffffff">
<h1 align="center">Introduction à XLM </h1>
<p align="center"> Un tutorial pratique </p>
<hr> Auteur : Dupont
</body>
</html>
```

22

XSLT : Règle de transformation (6)

- Le déclenchement d'une règle de transformation définie avec xsl:template est fait soit par xsl:apply-templates soit par xsl:call-template
- Une règle peut être déclenchée ou *instanciée* soit par son nom, soit en donnant la **catégorie des noeuds** du document source auxquels elle s'applique
- Cette **catégorie de noeuds** est spécifiée par une sous-classe des expressions XPath désignée par le terme *pattern*

23

XSLT : Règle de transformation (7)

- Les *patterns* : Le programme suivant recherche et affiche les noms des enseignants, les attributs ID et l'intitulé des séances de cours :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" encoding="ISO-8859-1"/>
<xsl:template match="/">   <xsl:apply-templates select="//NOM"/>
<xsl:apply-templates select="//SEANCE/@ID"/>
<xsl:apply-templates select="//SEANCE"/> </xsl:template>
<xsl:template match="NOM"> <NOM> <xsl:value-of select="."/></NOM> </xsl:template>
<xsl:template match="@ID"> <IDSEANCE><xsl:value-of select="."/></IDSEANCE>
</xsl:template>
<xsl:template match="PROGRAMME/SEANCE">
<SEANCE><xsl:value-of select="."/></SEANCE> </xsl:template>
</xsl:stylesheet>
```

24

XSLT : Règle de transformation (8)

- Rappel de l'exemple cours.xml

```
<?xml version='1.0' encoding="ISO-8859-1"?>
<COURS UE="TC2.1">
    <SUJET>XML et outils GL</SUJET>
    <ENSEIGNANTS>
    <!-- Enseignant responsable -->
      <NOM>Le Thanh</NOM>
      <NOM>Huet</NOM>
    </ENSEIGNANTS>
    <PROGRAMME>
      <SEANCE ID="1">Documents XML</SEANCE>
      <SEANCE ID="2">Programmation XSLT</SEANCE>
      <SEANCE ID="3">XML SCHEMA</SEANCE>
      <ANNEE>0506</ANNEE>
    </PROGRAMME>
</COURS>
```

25

XSLT : Règle de transformation (9)

- *Le résultat du programme XSL sur cours.xml*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<NOM>Le Thanh</NOM>
<NOM>Huet</NOM>
<IDSEANCE>1</IDSEANCE>
<IDSEANCE>2</IDSEANCE>
<IDSEANCE>3</IDSEANCE>
<SEANCE>Documents XML</SEANCE>
<SEANCE>Programmation XSLT</SEANCE>
<SEANCE>XML SCHEMA </SEANCE>
```

26

XSLT : Règle de transformation (10)

- Les expressions Xpath dans l'exemple apparaissent comme
  - valeur de l'attribut ***match*** de l'élément xsl:template,
  - d'autres comme valeur de l'attribut ***select*** de xsl:apply-templates.
- Elles jouent un rôle différent
  - l'expression dans l'élément xsl:apply-templates sert à désigner un ensemble de noeuds, et constitue donc une utilisation classique de XPath ;
  - l'expression dans l'élément xsl:template exprime en revanche une condition sur les noeuds qui vont permettre de déclencher la règle.

27

XSLT : Règle de transformation (11)

- Conclusion :
  - un *pattern* dans une règle définit la « situation » des noeuds de l'arbre qui vont pouvoir déclencher la règle.
  - Cette situation peut être très générale : un *pattern* NOM définit une règle applicable à tous les éléments de type NOM, quel que soit leur position, et quel que soit le type de leurs parents.
  - Un *pattern* comme PROGRAMME/SEANCE est déjà plus restrictif.
  - À l'extrême, un *pattern* peut être un chemin absolu XPath, et la règle ne s'applique plus alors qu'à des sous-arbres très particuliers.

28

XSLT : Règle de transformation (12)

- Conclusion (suite) :
  - Les *patterns* sont des expressions XPath dont l'interprétation consiste à *chercher un noeud contexte* à partir duquel le résultat du *pattern* contient le noeud courant. Ceci a deux conséquences :
    - l'expression *pattern* doit toujours retourner un ensemble de noeuds (node-set) ;
    - la recherche du noeud contexte peut devenir très complexe si on acceptait n'importe quelle expression XPath donnant un ensemble de noeuds : il faudrait, dans certains cas, pour vérifier si un noeud satisfait une expression, chercher un noeud contexte adapté parmi *tous* les noeuds du document source.

29

XSLT : Règle de transformation (13)

- Algorithme d'évaluation d'un parttern P à partir d'un nœud courant N :
  - Cet algorithme fait d'abord une distinction entre le cas d'un *pattern* absolu et le cas d'un *pattern* relatif :
  - Si P est absolu, prendre la racine comme nœud contexte et évaluer P ; si N fait partie du résultat, alors P satisfait
  - Sinon
    - (a) prendre N comme nœud contexte et évaluer P; si N fait partie du résultat, alors P satisfait;
    - (b) sinon prendre le père de N et recommencer l'évaluation ; si N fait partie du résultat, alors P satisfait;
    - (c) sinon recommencer en parcourant les ancêtres de N jusqu'à la racine du document.

30

XSLT : Règle de transformation (14)

- Exemples :
  - le *pattern* absolu /COURS/ENSEIGNANTS sera satisfait par tous les nœuds de type ENSEIGNANTS fils d'un élément racine de type COURS ;
  - //SEANCE[@ID=2] ou SEANCE[@ID=2] seront satisfait par tout nœud de type SEANCE ayant un attribut ID valant 2 (vérifiez l'équivalence des deux expressions)
  - NOM[position()=2] sera satisfait par tout nœud qui est le deuxième fils de type NOM de son père ;
  - *[position()=2][name()="NOM"] sera satisfait par tout nœud de type NOM, second fils de son père ;
  - /COURS/@UE[.="TC2.1"] sera satisfait par l'attribut UE, fils de l'élément racine COURS et dont la valeur est TC2.1

31

XSLT : Règle de transformation (15)

- Une règle est définie par un élément de premier niveau xsl:template. Cet élément comprend :
  - Des attributs qui décrivent les conditions de déclenchement de la règle, et
  - un contenu ou *corps de règle* décrivant le texte à produire quand la règle est déclenchée (ou « instanciée » pour utiliser le vocabulaire XSLT).

32

XSLT : Règle de transformation (16)

- Les quatre attributs d'une règle sont les suivants :
  - *match* est le *pattern* désignant les nœuds de l'arbre XML pour lesquels la règle peut se déclencher ;
  - *name* définit une *règle nommée* qui pourra être appelée directement par son nom ;
  - *mode* permet de définir des *catégories* de règles, à appeler dans des circonstances particulières ;
  - *priority* donne une priorité explicite à la règle.
- Tous les attributs sont optionnels, mais soit *name*, soit *match* doit être défini.

33

XSLT : Règle de transformation (17)

- Règles par défaut : XSLT définit un ensemble de règles par défaut qui sont appliquées quand aucune règle du programme n'est sélectionnée
  - La première règle par défaut s'applique à la racine du document et à tous les éléments.
  - La seconde règle par défaut s'applique aux nœuds de texte et aux attributs. Elle insère le contenu textuel de ces nœuds dans le document résultat

```
1ère règle
<xsl:template match="* | /">
<xsl:apply-templates/>
</xsl:template>
```

```
2e règle
<xsl:template match="text() | @*">
<xsl:value-of select="."/>
</xsl:template>
```

34

XSLT : Règle de transformation (18)

- Règles par défaut (suite) :
  - Enfin la dernière règle par défaut s'applique aux commentaires et aux instructions de traitement. Le comportement par défaut est de les ignorer. La règle ne fait donc rien :
    - <xsl:template match="processing-instruction() | comment()"/>
  - Exemple : programme xsl minimal

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"/>
```

35

XSLT : Règle de transformation (19)

- L'application de ce programme à notre document *CoursXML.xml* donne le résultat suivant :

```
<?xml version="1.0" encoding="UTF-
8"?>
XML et outils GL
Le Thanh
Huet
Documents XML
Programmation XSLT
XLM SCHEMA
0506
```

36

XSLT : Règle de transformation (20)

- Déclenchement de règles avec xsl:apply-templates : Cet élément a deux attributs, tous deux optionnels :
  - select contient l'expression XPath désignant les nœuds à traiter ;
  - mode est la catégorie des règles à considérer
- L'expression de l'attribut select doit toujours ramener un ensemble de nœuds. Sa valeur par défaut est child::node(), autrement dit tous les fils du nœud courant, quel que soit leur type, à l'exception comme d'habitude des attributs

37

XSLT : Règle de transformation (21)

- Exemple :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" encoding="ISO-8859-1"/>
<xsl:template match="/">
<xsl:apply-templates select="//ENSEIGNANTS"/>
</xsl:template>
<xsl:template match="ENSEIGNANTS">
<xsl:comment>
Application de la règle ENSEIGNANTS
</xsl:comment>
```

38

XSLT : Règle de transformation (22)

- Exemple (suite) :

```
<xsl:apply-templates/>
</xsl:template>
<xsl:template match="NOM">
<xsl:value-of select="position()"/> : Noeud NOM
</xsl:template>
<xsl:template match="text()">
<xsl:value-of select="position()"/> : Noeud de texte
</xsl:template>
<xsl:template match="comment()">
<xsl:value-of select="position()"/> : Noeud de commentaire
</xsl:template>
</xsl:stylesheet>
```

39

XSLT : Règle de transformation (23)

- *Résultat du programme précédent*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!--
Application de la règle ENSEIGNANTS
-->
1 : Noeud de texte
2 : Noeud de commentaire
3 : Noeud de texte
4 : Noeud NOM
5 : Noeud de texte
6 : Noeud NOM
7 : Noeud de texte
```

40

XSLT : Instructions

- L'exécution d'un programme XSLT consiste à déclencher (ou *instancier*) des règles définies par des éléments de premier niveau **xsl:template**.
- L'instanciation d'une règle consiste à produire un fragment du document résultat en fonction d'éléments littéraux, de texte et d'instructions XSLT constituant le *corps de règle :*
  - Les éléments littéraux et texte sont insérés directement au résultat
  - les *instructions XSLT* sont interprétées par le processeur XSLT

41

XSLT : Instructions (2)

- Exemple : création d'une liste HTML avec les séances d'une salle de cinéma

```
<h3>Séances</h3>
<ol>
<xsl:for-each
   select="SEANCES/SEANCE">
<li><xsl:value-of select="."/></li>
</xsl:for-each>
</ol>
```

- éléments littéraux : balises <h3>, <ol>,
- nœuds de type **Text :**
  - le noeud fils de <h3> est une chaîne de caractères
- Instructions dont le nom préfixé par xsl:
  - xsl:for-each
  - xsl:value-of

42

XSLT : Instructions (3)

- Liste d'instruction XSL

xsl:apply-imports	Permet d'appliquer une règle importée, tout en la complétant par un nouveau corps
xsl:apply-templates	Déclenche l'application de règles
xsl:attribute	Insère un attribut dans un élément du document résultat
xsl:call-template	Appelle une règle par son nom
xsl:choose	Structure de test équivalente au switch d'un langage comme Java ou C++
xsl:comment	Insère un noeud **Comment dans le document résultat**
xsl:copy	Copie un noeud du document source dans le document résultat
xsl:copy-of	Copie un noeud, ainsi que tous ses descendants
xsl:result-document	Permet de créer plusieurs documents résultats : ajouté dans XSLT 2.0, mais reconnu par la plupart de processeurs XSLT actuels, éventuellement sous un autre nom

43

XSLT : Instructions (4)

- Liste d'instruction XSL (suite)

xsl:element	Insère un noeud **Element dans le document résultat**
xsl:fallback	Règle déclenchée si le processeur ne reconnaît pas une instruction
xsl:for-each	Pour effectuer des itérations
xsl:if	Pour effectuer un branchement conditionnel
xsl:message	Pour produire un message pendant le traitement XSLT
xsl:number	Permet de numéroter les noeuds du document résultat
xsl:variable	Permet de définir un paramètre
xsl:processing-instruction	Insère un noeud **ProcessingInstruction dans le** document résultat
xsl:text Insère un noeud	**Text dans le document résultat**
xsl:value-of	Évalue une expression XPath et insère le résultat
xsl:variable	Permet de définir une variable

44

XSLT : évaluation d'un programme

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" encoding="ISO-8859-1"/>
<xsl:template match="/">
<html>
<head>
<title><xsl:value-of select="COURS/SUJET"/></title>
</head>
<body bgcolor="white">
<xsl:apply-templates/>
</body>
</html>
</xsl:template>
<xsl:template match="SUJET">
<h1><center><xsl:value-of
      select="."/></center></h1>
</xsl:template>
<xsl:template match="ENSEIGNANTS">
<h1>Enseignants</h1>
```

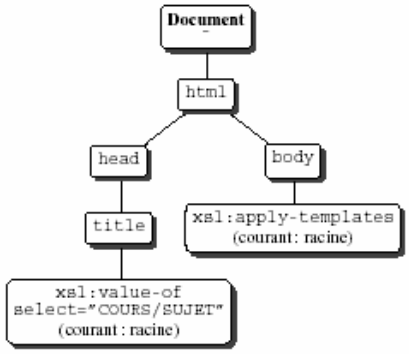
```
<ol>
<xsl:apply-templates select="NOM"/>
</ol>
</xsl:template>
<xsl:template match="PROGRAMME">
<h1>Programme</h1>
<ul>
<xsl:for-each select="SEANCE">
<xsl:call-template name="AfficheSeance"/>
</xsl:for-each>
</ul>
</xsl:template>
<xsl:template match="ENSEIGNANTS/NOM">
<li><xsl:value-of select="." /></li>
</xsl:template>
<xsl:template name="AfficheSeance">
<li> Séance <xsl:value-of select="concat(
position(), '/', last(), ' : ', . )"/>
</li>
</xsl:template>
</xsl:stylesheet>
```

45

XSLT : évaluation d'un programme (2)

- *Début de l'évaluation : le document résultat*

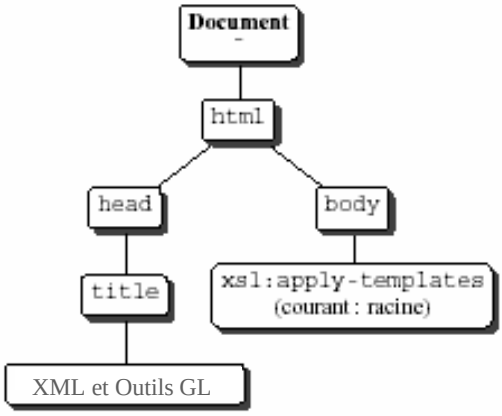
- *Après instanciation de la première règle*



46

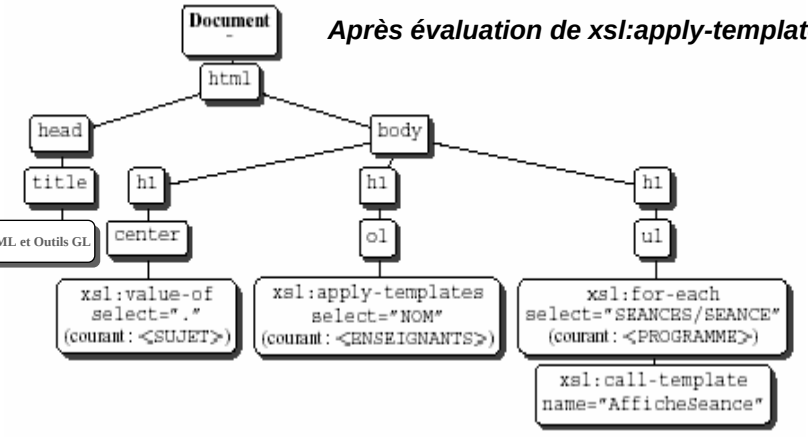
XSLT : évaluation d'un programme (3)

Après évaluation de xsl:value-of



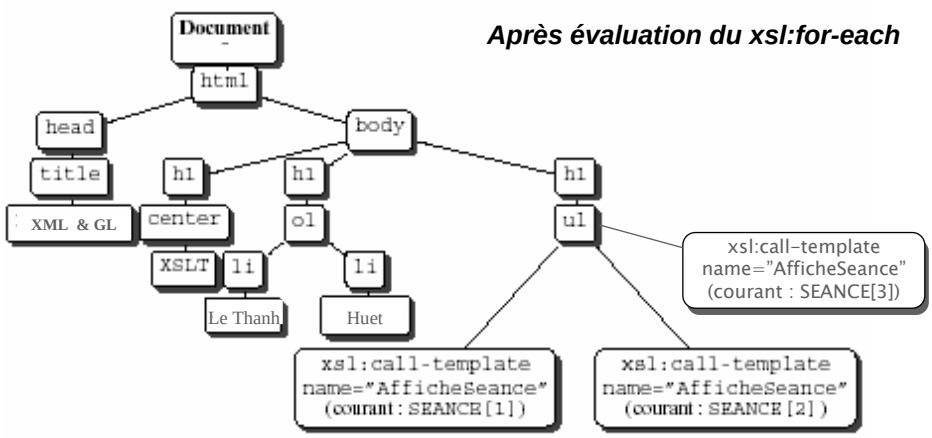
47

XSLT : évaluation d'un programme (4)

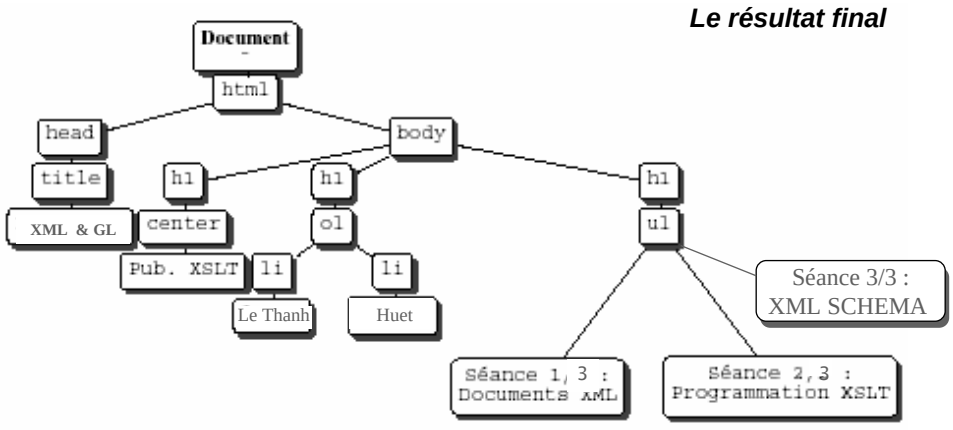


48

XSLT : évaluation d'un programme (5)



XSLT : évaluation d'un programme (6)



XSLT : évaluation d'un programme (7)

- *le document HTML résultat*

```
<html>
<head>
<META http-equiv="Content-
    Type" content="text/html;
    charset=ISO-8859-1">
<title>XML et Outils GL</title>
</head>
<body bgcolor="white">
<h1>
<center> title>XML et Outils GL
    </center>
</h1>
<h1>Enseignants</h1>
<ol>
<li>Le Thanh</li>
<li>Huet</li>
</ol>
<h1>Programme</h1>
<ul>
<li> S&eacute;ance 1/3 :
    Documents XML</li>
<li> S&eacute;ance 2/3 :
    Programmation XSLT</li>
<li> S&eacute;ance 3/3 : XML
    SCHEMA</li>
</ul>
</body> </html>
```

51

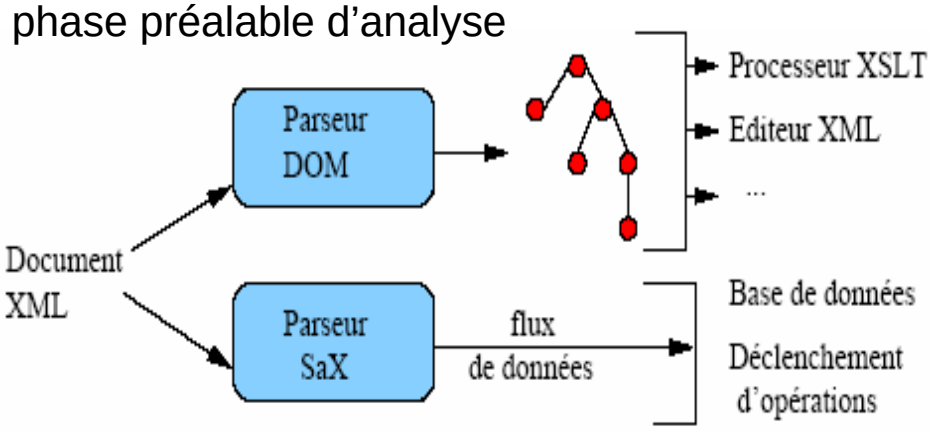
Programmation avec XML : introduction

- Les deux principales interfaces de programmation (API) XML depuis différents langages :
  - DOM (*Document Object Model*), basé sur une représentation arborescente
  - SaX (*Simple API for XML*), basé sur des *déclencheurs* "événement/action"
- On a pces API pour les différents langages : Java, Javascript, PhP, Perl, Python, C++, C#, …

52

Programmation avec XML : déroulement

Toutes les applications XML passent par une phase préalable d'analyse



53

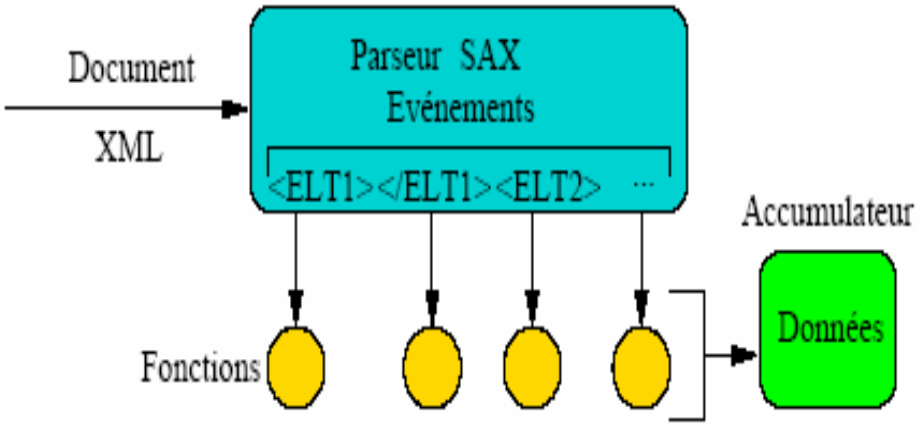
Programmation avec SaX : résumé

- SAX 1.0 apparaît en mai 1998 comme une tentative d'unifier ces interfaces.
- Originellement basé sur Java. Aujourd'hui : Perl, Python, C, C++, etc...
  - Modèle à évènements
    - Découpe en lexèmes
    - Émet des évènements
  - Processus rapide et léger
    - Les documents ne sont pas entièrement chargés en mémoire
    - Permet de traiter des gigas de données avec peu de mémoire
  - Lecture séquentielle du document
  - Bas niveau:
    - convient à tous usages
    - faible temps de traitement
  - Niveau lexical: l'application doit construire la structure
- Plusieurs implémentation de SAX dans les différents langages

54

Programmation avec SaX : architecture

Associer des événements aux balises :



Programmation avec SaX : architecture (2)

- Exemple

XML

```
<foo size="1">
  <bar> Hello ! </bar>
</foo>
```

SAX

```
begin
        doc
        begin : foo (size --> 1)
                begin : bar text : "Hello !"
                end : bar
        end : foo
end doc
```

Chaque élément lexical du fichier XML est traduit en un évènement SAX

Programmation avec SaX : évènements

- Un parseur SAX génère un événement à chaque fois qu'il rencontre
  - le début et la fin de document,
  - le début et la fin d'un élément,
  - une instruction de traitements
  - un commentaire, ...
- Les fonctions s'exécutent indépendamment =>

il faut leur faire partager une zone mémoire (« accumulateur »)

57

Programmation avec SaX : API Java

- Collection d'interfaces et de classes Java
  - Package *org.xml.sax*
- Interfaces:
  - Analyseur (Parser)
    - XMLReader
  - Event handlers
    - ContentHandler

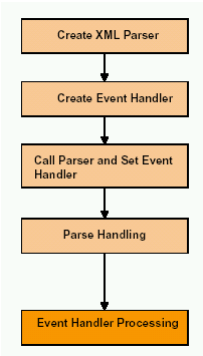
58

Programmation avec SaX : API Java (2)

- Défini les méthodes
- Séparation du *"comment"* et du *"quoi"*
  - Défini ce que les méthodes font
  - Mais ne dit pas comment les méthodes sont implémentées
- Les interface Java sont implémentées par des classes concrètes
- Les classes sont associées à l'interface à l'exécution
- La plupart des APIs Java sont définies comme des types d'interface Java:
  - Implémentations multiples
  - Les implémentations peuvent évoluer sans impacte sur les applications

Programmation avec SaX : API Java (3)

Procédure Sax




```
XMLReader parser =
XMLReaderFactory.createXMLReader();

myHandler handler = new Handler();

parser.parse(args[0]);

// SAX parser calls methodes on the event handler

public void startDocument() {
  System.out.println("XML Document Start");
}
```

Programmation avec SaX : API Java (4)

SAX1 versus SAX2 :

SAX 1:
La notion de source ou de filtre est implicite.
Ne prend pas en compte les *namespaces*.
Ne prend pas en compte les entités externes.

SAX 2:
Redéfinir certaines interface (ex: Parser - XMLReader)
Améliorer et enrichi les interfaces fournies par SAX1.
Gèrer les *namespaces*

Interface Parser (SAX 1)

- C'est une source d'évènements SAX : en général lecture d'un stream

```
public interface Parser {
  // Lance le parsing du document XML
  //     throws IOException, SAXException;
  void parse(String);
  void parse(InputSource);

  // Connexion des handlers
  void setDTDHandler(DTDHandler);
  void setDocumentHandler(DocumentHandler);
  void setEntityResolver(EntityResolver);
  void setErrorHandler(ErrorHandler);
  void setLocale(java.util.Locale);
}
```

Interface XMLReader (SAX 2)

- Source généralisée pour SAX2
- Les options (*features*) permettent de gérer finement le parsing

```
public interface XMLReader {
  // ... On retrouve les méthodes citées dans Parser
  // Gestion des property et feature
  //     throws SAXNotRecognizedException,
  //            SAXNotSupportedException;
  boolean getFeature(String);
  void setFeature(String, boolean);
  Object getProperty(String);
  void setProperty(String, Object);
  // Plus de locale
  // Récupération deshandlers
  ContentHandler getContentHandler();
  DTDHandler getDTDHandler();
  EntityResolver getEntityResolver();
  ErrorHandler getErrorHandler();
}
```

63

Instanciation de XMLReader

L'implémentation concrète est associée à l'interface XMLReader doit être créé avant l'analyse. Cette instance est créée en utilisant la méthode statique *createXMLReader()* issue de la classe *XMLReaderFactory*

```
XMLReader parser = null;
try {
  // Creation d'une instance d'analyseur SAX
 parser = XMLReaderFactory.createXMLReader()
  // Analyse du document XML  ...
}
catch (SAXException e) {
  // Traite les exceptions:
  //    Problème de création de l'analyseur, lors de
  //    l'analyse, etc...
}
```

64

Programmation avec SaX : exemple java (1)

```
import java.io.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;
import
    org.apache.xerces.parsers.SAXParser;
public class Trace extends DefaultHandler
    {
// début de document
public void startDocument() {
    System.out.println("start document");
    }
// fin de document
public void endDocument() {
    System.out.println("end document");
    }
```

```
// balise d'ouverture
public void startElement(String uri,
String localName, String qName,
Attributes attributes) {
System.out.println("starting element: " +
qName);
    }
// balise de fermeture
public void endElement(String uri,
String localName, String qName) {
System.out.println("end element: " +
    qName);
}
```

65

Programmation avec SaX : exemple java (2)

```
// espace blanc
public void ignorableWhitespace(char[] ch,
int start, int length) {
System.out.println("whitespace, length " +
length);
}
// instruction de traitement
public void processingInstruction(String
    target,
String data) {
System.out.println("processing instruction: "
    + target);
}
```

```
// texte
public void characters(char[] ch, int start,
int length){
System.out.println("character data, length "
    + length);
}
// procédure principale
public static void main(String[] args) {
Trace t = new Trace();
SAXParser p = new SAXParser();
p.setContentHandler(t);
try { p.parse(args[0]); }
catch (Exception e) {e.printStackTrace();}
}
```

66

Programmation avec SaX : exemple BD

À partir d'un flux XML contenant des films :

<FILMS>
...
<FILM>
<TITRE>Alien</TITRE>
<ANNEE>1979</ANNEE>
<AUTEUR>Ridley Scott</AUTEUR>
<GENRE>Science-fiction</GENRE
<PAYS>USA</PAYS>
</FILM>
...
</FILMS>

Insertion des tuples

Programmation avec SaX : exemple BD

Association des évènements

- Début de document : connexion à la base
- Balise < FILM> : création d'un enregistrement *Film*
- Balise < TITRE> : on affecte *Film.titre*
- Balise < ANNEE> :on affecte *Film.annee*
- etc.
- Balise < FILM> :insertion de l'enregistrement dans la base
- Balise </FILM> :destruction de l'enregistrement *Film*

Programmation avec DOM : introduction

- Javascript dans les navigateurs
- Langage doit pouvoir accéder au **document** HTML dans le navigateur:

  <a href="" onMouseOver="this.fillcolor='red'">
- Problème: Netscape et IE avaient des **API**s différentes pour manipuler le document, le DHTML
- Besoin d'une API commune, le **DOM** (1998).

69

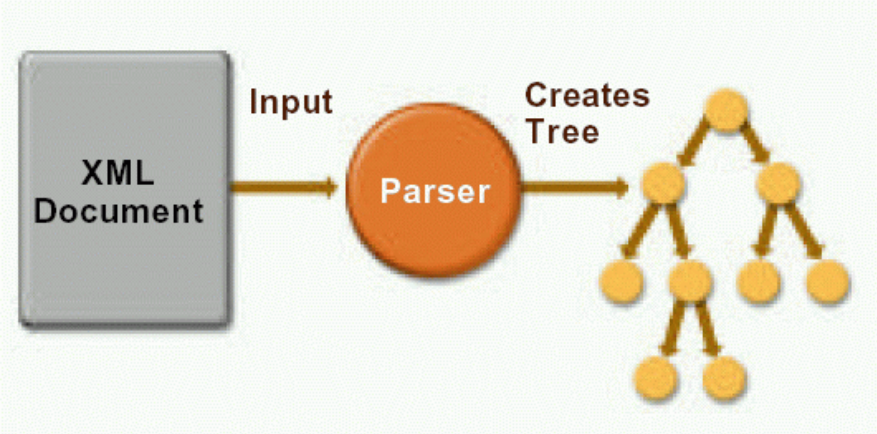
Programmation avec DOM : modèle

- Un parseur DOM prend en entrée un document XML et construit un **arbre** formé **d'objets** comparable à un arbre de syntaxe abstraite :
  - chaque objet appartient à une sous-classe de Node
  - des opérations sur ces objets permettent de **créer** de nouveaux noeuds, ou de **naviguer** dans le document
  - Possibilité de parcourir l'arbre "n'importe comment"
  - Nécessite de stocker le document intégralement en mémoire

=> éditeurs XML, processeurs XSLT

70

Programmation avec DOM : modèle (2)



71

Programmation avec DOM : modèle (3)

Types de nœuds:

Document node	Processing Instruction node
Document Fragment node	Document type node
Element node	Entity node
Attribute node	Entity Reference node
Text node	CDATA Section node
Comment node	Notation node

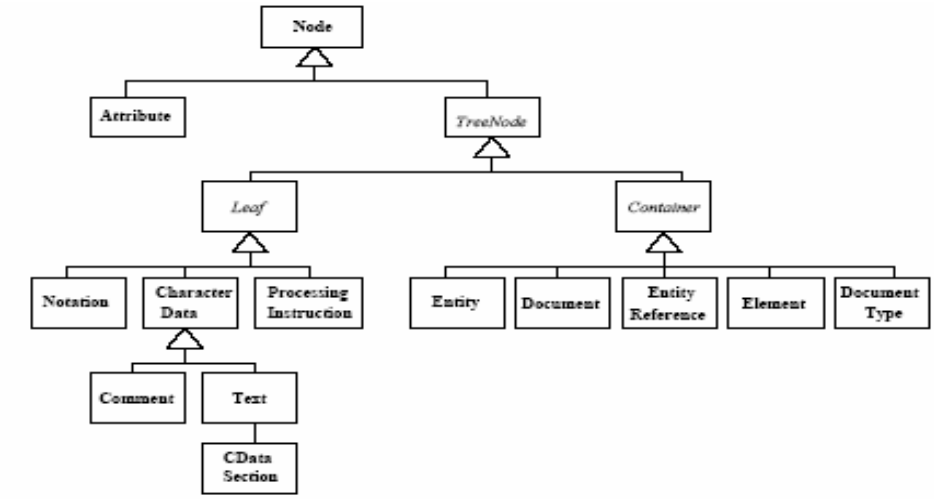
72

Programmation avec DOM : modèle (4)

- Règles de construction
  - Un noeud Document contient:
    - un noeud Element (root element node)
    - un ou plusieurs noeuds Processing Instruction
  - Un noeud Element peut contenir:
    - d'autres noeud Element
    - un ou plusieurs noeuds Text
    - un ou plusieurs noeuds Attribut
  - Un noeud Attribut contient:
    - Un noeud Text

73

Programmation avec DOM : modèle (5)



74

Programmation avec DOM : modèle (6)

- Document XML

```
<gens>
<personne naissance="1982">
   <nom>
     <prenom>Alain</prenom>
     <nomDeFamille>Turing
     </nomDeFamille>
   </nom>
   <profession>Ingénieur en Infor
   matique</profession>
   </personne>
</gens>
```

- Noeud XML Document
- Noeud Element "gens"
  - Noeud Element "personne"
    - Noeud Element "nom"
      - Noeud Element "prenom"
        - Noeud Text "Alain"
      - Noeud Element "nomDeFamille"
        - TNoeud Text " Dupont"
    - Noeud Element "Profession"
      - Noeud Text "Ingénieur en Informatique"
    - Noeud Attribut "naissance"
      - Noeud Text "1982"

75

Programmation avec DOM: propriété d'un noeud

Propriété	Type
nodeType	**unsigned short**
nodeName	**DOMString**
nodeValue	**DOMString**
attributes	**NamedNodeMap**

76

Programmation avec DOM: propriété d'un noeud

Propriété	Type
parentNode	**Node**
firstChild	**Node**
lastChild	**Node**
childNodes	**NodeList**
previousSibling	**Node**
nextSibling	**Node**

77

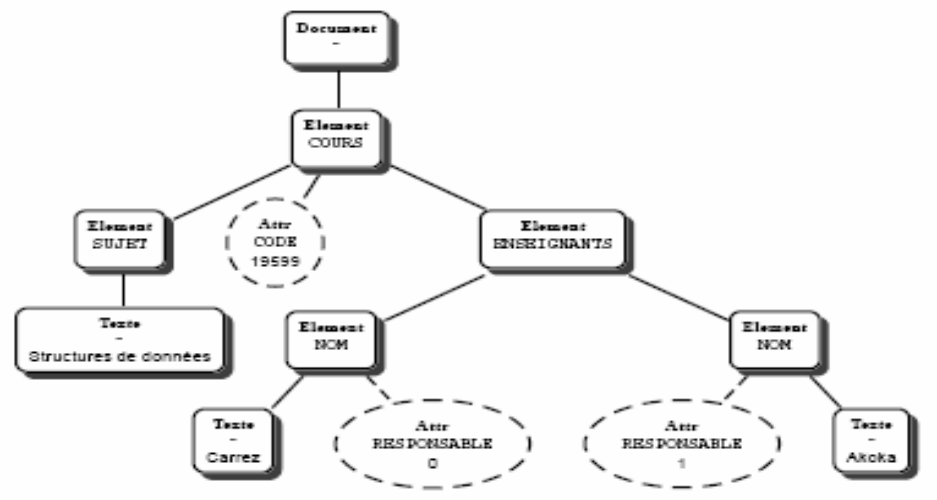
Programmation avec DOM: propriété d'un noeud

- Exemple

```
<?xml version='1.0' encoding="ISO-8859-1"?>
<COURS CODE="19599">
  <SUJET>Structures de données</SUJET>
  <ENSEIGNANTS>
      <NOM RESPONSABLE="0">Carrez</NOM>
      <NOM RESPONSABLE="1">Akoka</NOM>
  </ENSEIGNANTS>
</COURS>
```

78

Programmation avec DOM: arbre généré



79

Programmation avec DOM: arbre généré

Résultat	Méthode	Paramètres
Node	*insertBefore()*	**Node** *nouv*, **Node** *fils*
Node	*replaceChild()*	**Node** *nouv*, **Node** *anc*
Node	*removeChild()*	**Node** *fils*
Node	*appendChild()*	**Node** *nouv*
boolean	*hasChildNodes()*	
Node	*cloneNode()*	**boolean** *prof*

80

Programmation avec DOM: Interfaces

- Il s'agit d'un jeu d'interfaces normalisées par le W3C (package org.w3c.dom)
- Il existe plusieurs niveaux :
  - DOM Level 1 : XML + HTML (org.w3c.dom.html)
  - DOM Level 2 : org.w3c.dom.(events|ranges|traversal) + namespaces
  - DOM Level 3 : en préparation... (load / save)

Programmation avec DOM: noeud

```
public interface Node {
  // nom du noeud
  String getNodeName();
  String getLocalName();
  String getNamespaceURI();
  short getNodeType(); // ELEMENT_NODE, TEXT_NODE, etc...

  // valeur du noeud (dépend de la classe concrète)
  String getNodeValue();
  void setNodeValue(String);

  boolean hasAttributes();
  NamedNodeMap getAttributes();

  void setPrefix(String);
  String getPrefix();

  ...
}
```

Programmation avec DOM: arbre

```
public interface Node {
  ... // gestion d'un arbre père-fils
  // modification des fils
  Node appendChild(Node);
  Node removeChild(Node);    Node replaceChild(Node, Node);
  Node insertBefore(Node, Node);
  // parcours
  boolean hasChildNodes();
  NodeList getChildNodes();
  Node getFirstChild();
  Node getLastChild();
  Node getPreviousSibling();
  Node getNextSibling();
  // retour au père
  Node getParentNode();
  // retour à la racine
  Document getOwnerDocument();
  ...
}
```


Programmation avec DOM: document

```
public interface Document extends Node {

  DocumentType getDoctype(); // DOCTYPE du document
  Element getDocumentElement(); // Elément racine du document

  // recherche d'élements
  Element getElementById(String);
  NodeList getElementsByTagName(String);
  NodeList getElementsByTagNameNS(String, String);
  // retourne l'implémentation
  DOMImplementation getImplementation();
  ...
}
```

Programmation avec DOM: document (2)

- *création des noeuds* :
- public interface Document extends Node {

```
// partie "factory" (création des noeuds)
Element createElement(String);
Element createElementNS(String, String);
Attr createAttribute(String);
Attr createAttributeNS(String, String);
Text createTextNode(String);
CDATASection createCDATASection(String);
Comment createComment(String);
DocumentFragment createDocumentFragment();
EntityReference createEntityReference(String);
ProcessingInstruction createProcessingInstruction(String, String);

// importation d'un noeud/sous-arbre issu d'un autre document
Node importNode(Node, boolean);
```


Programmation avec DOM: élément

- public interface Element extends Node {

```
// recherche d'éléments du sous-arbre
NodeList getElementsByTagName(String);
NodeList getElementsByTagNameNS(String, String);
// accès aux attributs
String getAttribute(String);          String getAttributeNS(String, String);
Attr getAttributeNode(String);        Attr getAttributeNodeNS(String, String);
boolean hasAttribute(String);         boolean hasAttributeNS(String, String);
void setAttribute(String, String);    void setAttributeNS(String, String, String);
Attr setAttributeNode(Attr);          Attr setAttributeNodeNS(Attr);
void removeAttribute(String);         void removeAttributeNS(String, String);
Attr removeAttributeNode(Attr);
// accès au nom de l'élément
String getTagName();
}
```

Programmation avec DOM: attribut

```
public interface Attr extends Node {
    String getName(); // nom de l'attribut
    String getValue(); // valeur de l'attribut
    void setValue(String); // modification de la vale
ur
    Element getOwnerElement(); // élément père
    boolean getSpecified(); // valeur implicite/explic
ite
}
```

87

Programmation avec DOM: CharacterData

```
public interface CharacterData extends Node {
    // récupération des données
    int getLength();
    String getData();
    // modification des données
    void setData(String);
    void insertData(int, String);
    void replaceData(int, int, String);
    void appendData(String);
    void deleteData(int, int);
    String substringData(int, int); // extraction d'une sous-chaîne
}
```

88

Programmation avec DOM: Text/Comment

- public interface Text extends CharacterData {
  Text splitText(int); *// sépare en deux noeuds*
  }
- public interface Comment extends CharacterData
  {
  }

Programmation avec DOM : Entity/EntityReference

- public interface Entity extends Node {
  // définition d'une entité
  String getNotationName();
  String getPublicId();
  String getSystemId();
  }
- public interface EntityReference extends Node {
  // entité référence
  }

Programmation avec DOM :DocumentType

```
public interface DocumentType extends Node {
    // nom du document
    String getName();
    String getPublicId();
    String getSystemId();
    // liste des entités
    NamedNodeMap getEntities();
    // partie locale
    String getInternalSubset();
    // liste des notations
    NamedNodeMap getNotations();
}
```


Programmation avec DOM: exemple de code

Ajoute un élément <adventure type="epic">&adventure1;</adventure>
avant le 4e élément du document :

```
var racine = myDocument.documentElement;
var enfants = racine.childNodes;

var el_nouv = createElement("adventure") ;
var ent_ref = createEntityReference("adventure1");

el_nouv.setAttribute("type", "epic");
el_nouv.appendChild(ent_ref)

insertBefore(el_nouv, enfants.item(3));
```

Programmation avec DOM: JavaScript

- Insertion de code :
  - **Un code JavaScript, en effet, peut contenir un certain nombre de caractères < ou >, qui risquent d'être interprétés par un *parser* XSL. Le code doit donc être inséré dans un élément XML non interprété:**
  - **<script type="**text/javascript**"><![CDATA[**(Emplacement du code)**]]></script>**
  - **Il est alors possible d'écrire le code JavaScript (ou autre...) sans se soucier d'éventuelles incompatibilités de syntaxe**

93

Programmation avec DOM: JavaScript (2)

- **Accéder aux fichiers XML et XSL**
  - Pour cela, on utilise deux collections :
    - **document.XMLDocument** et
    - **document.XSLDocument**

```
var XMLsource = new Object ;
var XSLsource = new Object ;
XMLsource = document.XMLDocument ;
XSLsource = document.XSLDocument ;
```

94

Programmation avec DOM: JavaScript (3)

- **Sélection des éléments dans un fichier XML**
  - Cela est réalisé par un appel à la méthode
    - **selectNodes()**, qui prend comme argument une expression XPath, appliquée à la collection **documentElement** de l'objet précédemment défini

      XMLsource.documentElement.**selectNodes(livre/auteur/@nom) ;**
    - On peut également modifier un élément de la feuille XSL

collection_element=XSLsource.documentElement.**selectNodes("xsl:for-each [@select='livre']") ;**
element=collection_element[0] ;
element.**setAttribute("select", auteur/@nom)**

95

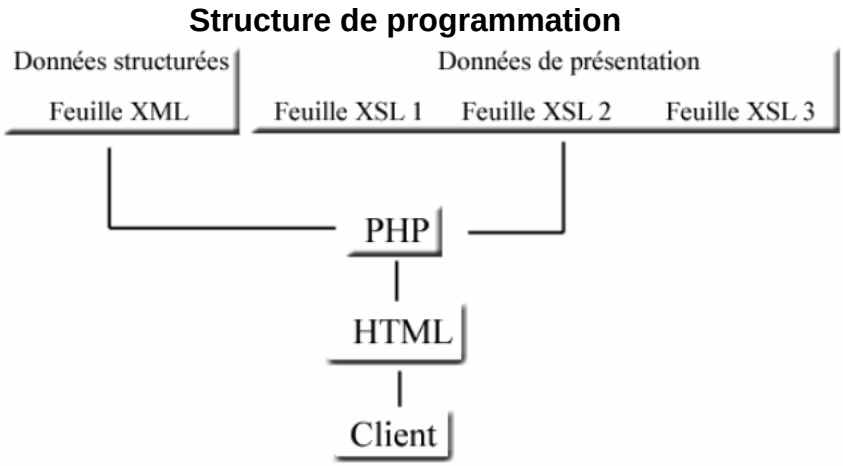
Programmation avec DOM: JavaScript (4)

- **Mise à jour de l'affichage**
  - Il est nécessaire, pour terminer, de mettre à jour l'afichage avec les modifications désirées. Cela est réalisé en utilisant la propriété **innerHTML** de l'objet **document.body**, et lui affectant le résultat de la transformation de l'objet XML par l'objet XSL modifié :

document.body.innerHTML =
XMLsource.**transformNode(***XSLsource*) ;

96

Programmation avec DOM: php



97

Programmation avec DOM: php (2)

Problème :

- désirons réaliser un cours, divisé en chapitres puis en parties. Toutes les numérotations de chapitres et de parties seront réalisées automatiquement, de manière à éviter des modifications pénibles dans le cas d'ajout d'une partie ou d'un chapitre. Un index sera généré automatiquement. Le cours sera consultable partie par partie ou en une version imprimable. La version imprimable reprendra sur une seule page l'index et l'ensemble du cours. La version partie par partie disposera d'en-tête et de pied de page permettant de revenir au sommaire ou de passer à la page précédente ou suivante. L'entête et le pied de page seront générés automatiquement, de manière là encore à éviter les modifications dans le cas d'un ajout. Les fichiers XML et XSL qui sont présentés ciaprès ne seront jamais accessibles au client. Le client appellera une page PHP qui retournera la page HTML mise en forme.

98

Programmation avec DOM: php (3)

Exemple

Document	Description
Cours.xml	Cours contenant des chapitres
Index.xsl et index.php	Générer des index d'accès au cours
Article.xsl et article.php	Afficher une partie de cours
full.xsl et full.php	Afficher le cours sous forme imprimable
Balise.xsl	Inclus dans les autres pour afficher les textes

Programmation avec DOM: php (4)

- Fichier index.php

```
<?php
$xh = xslt_create();
$file=fopen("cours.xml","r");
$xml=fread($file,16384);
fclose($file);
$file=fopen("index.xsl","r");
$xsl=fread($file,16384);
fclose($file);
```

```
$arguments = array('/_xml' =>
    $xml, '/_xsl' => $xsl);
$result = xslt_process($xh,
    'arg:/_xml', 'arg:/_xsl',
    NULL, $arguments);
xslt_free($xh);
print "$result";
?>
```

Programmation avec DOM: php (5)

$xh = xslt_create();	créer ici une instance du parseur XML.
$file=fopen("cours.xml","r"); $xml=fread($file,16384); fclose($file);	lire le contenu du fichier XML (jusqu'à 16384 octets)
$file=fopen("index.xsl","r"); $xsl=fread($file,16384); fclose($file);	lire le fichier XSL.
$arguments = array('/_xml' => $xml, '/_xsl' => $xsl); $result = xslt_process($xh,'arg:/_xml', 'arg:/_xsl', NULL, $arguments);	Le fichier XML et le fichier XSL sont passés au parseur instancié plus tôt. Le fichier HTML résultant sera récupéré dans la variable $result.
print "$result"; xslt_free($xh);	Détruire le parseur XML Imprimer le fichier html généré

101

Programmation avec DOM: php (6)

- Full.php

```
<?php
$xh = xslt_create();
$file=fopen("cours.xml","r");
$xml=fread($file,16384);
fclose($file);
$file=fopen("full.xsl","r");
$xsl=fread($file,16384);
fclose($file);
$arguments = array(
'/_xml' => $xml,'/_xsl' => $xsl);
$result = xslt_process($xh,
    'arg:/_xml', 'arg:/_xsl',
    NULL, $arguments);
xslt_free($xh);
print "$result";
?>
```

102

Programmation avec DOM: php (7)

- article.php

```
<?php
$xh = xslt_create();
$file=fopen("cours.xml","r");
$xml=fread($file,16384);
fclose($file);
$file=fopen("full.xsl","r");
$xsl=fread($file,16384);
fclose($file);
```

```
$arguments = array(
'/_xml' => $xml,'/_xsl' => $xsl);
if(isset($_GET['id']))
$xslt_params["id"] = $_GET['id'];
else
$xslt_params["id"] = "1";
$result = xslt_process($xh,
    'arg:/_xml', 'arg:/_xsl', NULL,
    $arguments,
    $xslt_params["id"]);
xslt_free($xh);
print "$result";
?>
```

103

Programmation avec DOM: php (8)

- SimpleXML dans PHP5 :
  - L'introduction de SimpleXML est certainement une belle avancée dans le domaine d'interaction entre PHP et XML.
  - Là où l'utilisation des objets existant était parfois complexe, SimpleXML, qui porte bien son nom, améliore le confort du développeur pour interpréter des fichiers XML

104

Programmation avec DOM: php (9)

- Exemple : nouvelles.xml

```
<?xml version="1.0"?>
<nouvelles>
<nouvelle>
<contenu>Monsieur et Madame X viennent de se marier!
    Félicitations</contenu>
<date>10/10/2004</date>
</nouvelle>
<nouvelle>
<contenu>Confirmant les tendances pessimistes, Monsieur et Madame X
    viennent de
<date>10/10/2005</date>
</nouvelle>
</nouvelles>
```


Programmation avec DOM: php (10)

- Code php pour parcourir l'arbre xml

```
< ?php
$nouvelles = simplexml_load_file('nouvelles.xml');
foreach($nouvelles->nouvelle as $nouvelle) {
echo 'Contenu : ' ,utf8_decode($nouvelle-
    >contenu).'<br>';
echo 'Date : ' ,$nouvelle->date.'<br>';}
```

  - ?>

Programmation avec DOM: php (11)

Résultat généré par le fichier php depuis du fichier xml

Affichage généré par le script
Contenu : Monsieur et Madame X viennent de se marier! Félicitations Date : 10/10/2004
Contenu : Confirmant les tendances pessimistes, Monsieur et Madame X viennent de divorcer Date : 10/10/2005

Programmation avec DOM: php (12)

- Il est possible de transférer à SimpleXML un objet DOM préalablement instancié. Cette manipulation se fait comme ceci

```
$xml = new domDocument ;
$xml->load('nouvelles.xml') ;
$simpleXml = simplexml_import_dom($xml);
```

Programmation avec DOM: php (13)

- La méthode xpath permet de pointer directement un noeud spécifique

```
<?
$nouvelles = simplexml_load_file('nouvelles.xml');
$xpath = '/nouvelles/nouvelle/contenu';
$nouvelle = $nouvelles->xpath($xpath) ;
foreach( $nouvelle as $news ) { echo
   utf8_decode($news);}
?>
```

109

Programmation avec DOM: php (14)

- Affichage généré par le script

Monsieur et Madame X viennent de se marier! Félicitations
Confirmant les tendances pessimistes, Monsieur et Madame X viennent de divorcer

110