"XML-based web-publishing" & Cocoon

Code: xml-ctalk

"Séance des Technologies" de l'Observatoire technologique de l'Etat de Genève

jeudi 21 juin 2001, Palais de Justice de Genève

Thème : "éditique" (ou "paperasseware"), de MS Word à XML

Originaux

url: **http://tecfa.unige.ch/guides/tie/html/xml-ctalk/xml-ctalk.html**

url: **http://tecfa.unige.ch/guides/tie/pdf/files/xml-ctalk.pdf**

Daniel K. Schneider, TECFA, Université de Genève

Version: 1.1 (modifié le 21/6/01)

Résumé

Exposé introduisant le "XML framework" du consortium W3C et le "Cocoon publication framework" qui implémente un partie de ces spécifications.

1. Table des matières

2. Introduction: le phénomène XML

- Introduit en 1997, but: formalisme commun à (presque) tous les langages Internet

Le problème HTML

Le monde HTML

incompatibilités
manque de flexibilité
beaucoup de faiblesses
manque de portée
peu machine-readable
fait pour l'écran

facile .....
beaucoup d'outils

Le monde XML

standardisation
extensibilité
faiblesses (?)
ouvert
portée large
plus facile à parser
indépendant du rendering

difficile .....
peu d'outils

Le problème de la communication entre ordinateurs

- trop de standards
- trop de formalismes différents

L'explosion des nouveaux langages Internet

- Coût élevé pour écrire les outils de A-Z

3. XML

2 façons d'aborder XML

(1) XML comme formalisme

(un formalisme pour créer des grammaires qui définissent des contenus structurés)

```
<!ELEMENT page  (title, content,
comment?)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT content (#PCDATA)>
<!ELEMENT comment (#PCDATA)>
```

```
 <title>Hello Cocoon friend</
title>
 <content>
      Here is some content :)
 </content>
 <comment>
      Written by DKS/Tecfa,
 </comment>
</page>
```

(2) Le "XML framework" du W3C

(un ensemble de langages/grammaires pour créer un web plus puissant. Ces formalismes sont définis avec XML !)

graphisme: SVG
formattage: XSL/FO
hypertexte: Xlink, Xpointer
recherche: XQL
transformations: XSLT
commerce: SOAP
.....
.....

Pour faire vivre ces langages il faut des outils (se mettent peu à peu en place)

3.1 XML comme formalisme

Un document XML doit être bien formé ("well-formed")

- Le document commence par une déclaration XML (version obligatoire),
  `<?xml version="1.0"?>`
  - possibilité de choisir un encodage (le défaut est utf-8):
  `<?xml version="1.0" encoding="ISO-8859-1"?>`
- Structure hiérarchique:
  - balises de début et balises de fin doivent correspondre
  - pas de croisements de type ....
  - Case sensitivity !
- Pas de balises vides
  - Les balises sans contenu utilisent le terminateur "/" (e.g.
)
- Les valeurs d'attributs sont quotés:
  - (e.g. <a href="http://tecfa.unige.ch:8080/xml.html">)
- Un seul élément racine (root):
  - L'élément root ne peut apparaître qu'une fois
  - Le root ne doit pas apparaître dans un autre élément (comme <html>)
- Caractères spéciaux: <, &, >,", '
  - Utilisez `< & > &aquot; &apos;` à la place dans un texte !

Un document XML peut être validé

- Un document "valide" doit être:
  - "well-formed",
  - faire référence à un DTD (ou une autre grammaire)
  - et être conforme au DTD (ou à la grammaire en général)
- Avantages d'une validation:
  - traitement électronique de documents
  - applications de feuilles de style
  - respect de procédures, règles, etc.

DTD (Document Type Definition)

- Un DTD est:
  - Une grammaire (ou schéma ou jeu de règles) qui définit les balises utilisés et leurs attributs, et qui spécifie leur possible imbrication (relation).
  - et il peut être référencé par un URI ou inclus dans le document XML
- Ils existent bien d'autres types grammaires XML comme XSchema (qui peut-être va remplacer les DTD), Relax, etc.

Exemple 3-1: Un simple fichier XML avec un schéma DTD

Données XML

- On a un document avec une racine page `<page>`
- 3 éléments supplémentaires

```
<page>
 <title>Hello le palais du justice</title>
 <content>
      Voici du texte.
 </content>
 <comment>
      Written by DKS/Tecfa.
 </comment>
</page>
```

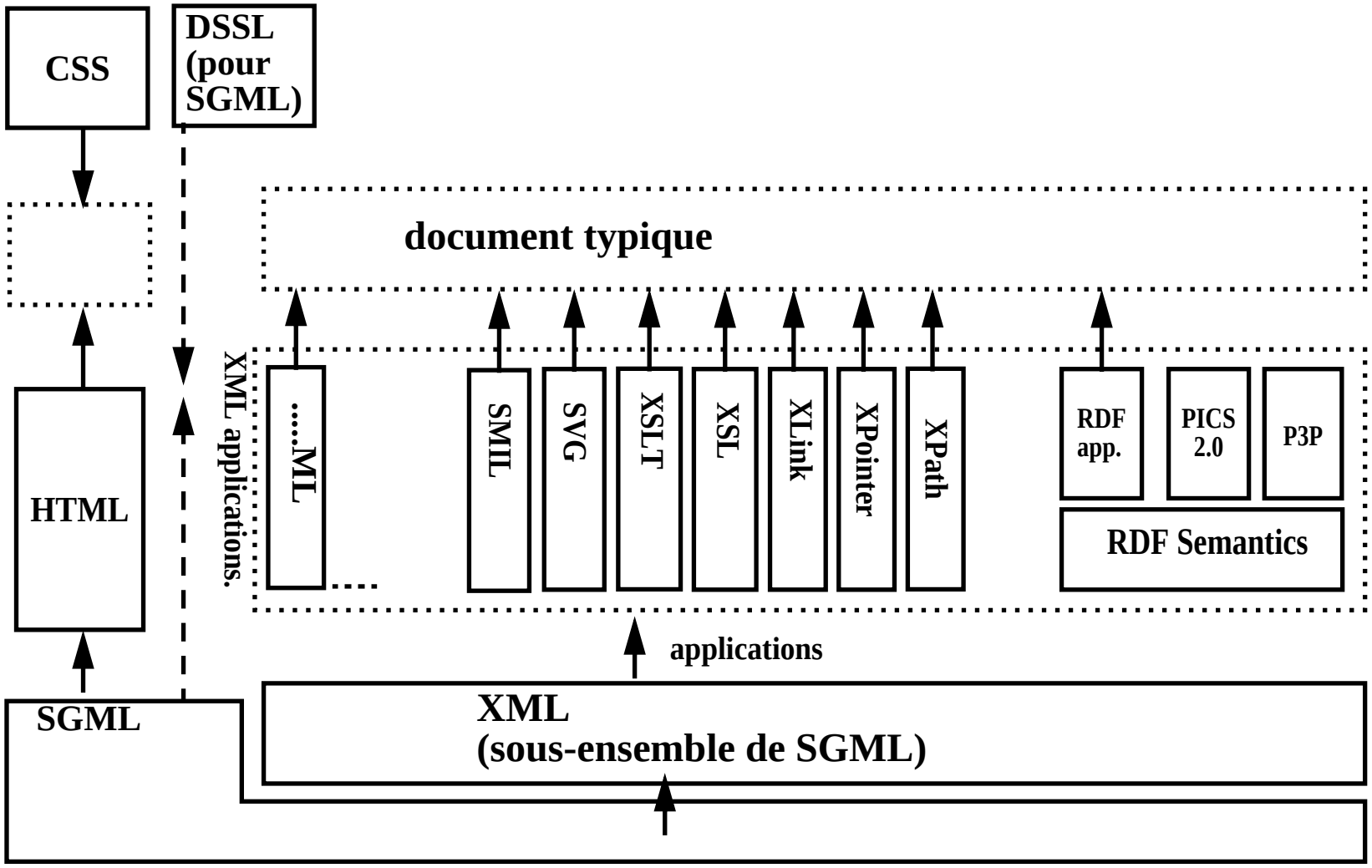
Le DTD (à titre d'indication)

- `title` et `content` sont obligatoires
- Il peut avoir plusieurs éléments `content`
- Le `comment` à la fin est à option

```
<!ELEMENT page  (title, content+, comment?)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT content (#PCDATA)>
<!ELEMENT comment (#PCDATA)>
```

3.2 Le "XML framework"

Eléments importants pour le "Web publishing"



Quelques "applications" (schémas) du consortium (W3C):

- XSL / FO (application XML): Langage de style pour XML
- XSLT (application XML): Langage de transformation pour XML
- XLink: Hypertext links
- XPointer (pointeurs vers une ressource) et XPath (chemins dans la structure)
  - (utilisés par XSLT, XInclude, XLink, etc.)
- Applications RDF: (par exemple IMS)
  - voir: **http://tecfa.unige.ch/guides/rdf/pointers.html**
- PICS 2.0: Platform for Internet Content Selection
  - **http://www.w3.org/PICS/**
- SMIL: Synchronized Multimedia Integration Language
  - http://www.w3.org/AudioVideo/
- P3P: Platform for Privacy Preferences
  - **http://www.w3.org/P3P/**
- SVG: Scalable Vector Graphics
- MathML: Mathematical Markup Language
  - **http://www.w3.org/Math/**
- XHMTL: (HTML 4.0 en XML)
  - Tags fermés, pas de croisements, imbrication correcte des éléments

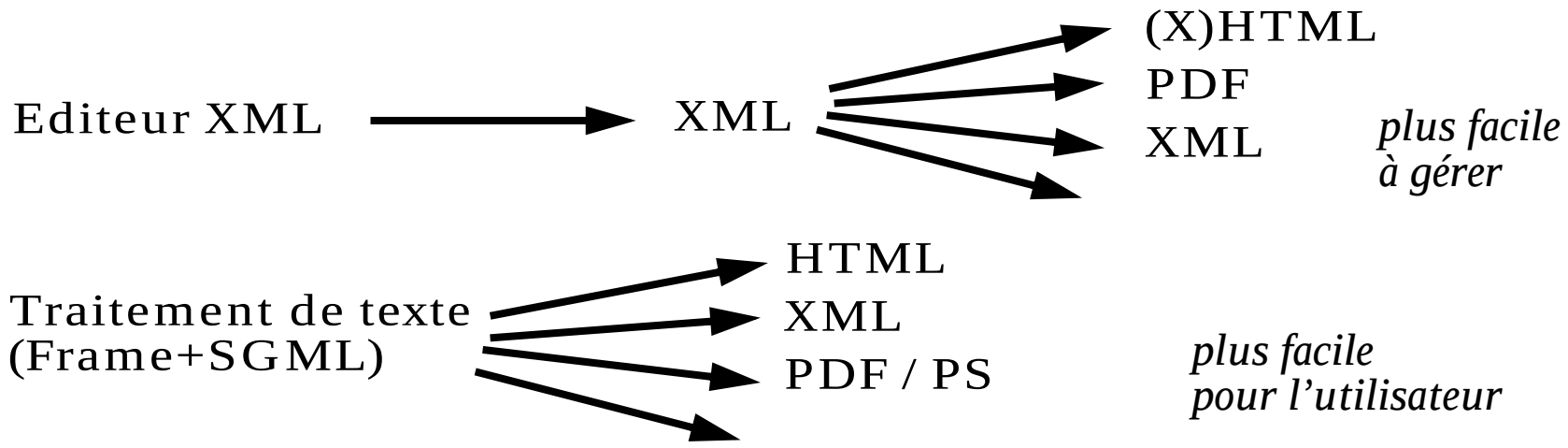
Ce n'est pas tout !

4. Publier avec XML

4.1 Edition "manuelle" de XML

- Outils permettant d'éditer un "arbre" (quelques programmes Java gratuits)
- Outils d'édition de texte structuré (éditeurs de programmation comme Emacs)
- Outils semi-professionnels (comme XMetal ou EpcEdit): assez chers.
- Outils professionnels SGML / XML comme FrameMaker+SGML: chers.)
- Plug-ins pour traitement de texte (médiocres encore)
- Filtres vers XML (HTML, RTF, Latex, etc.): médiocres par nature

2 grandes options:



4.2 Langages de base pour systèmes de "textes" on-line

- Markup: Langage pour caractériser des éléments d'information
- Style: Langage pour définir la mise en page d'une classe d'objets
- Linking: Langage pour représenter des liens entre éléments et objets
- Assemblage: Assembler du texte à partir de fragments d'autres textes
- Scripting: Interface et langages pour créer des applications client-side

<table>
<tr><th></th><th>monde HTML</th><th>monde XML</th><th>monde SGML</th></tr>
<tr><td>Linking</td><td>(<A> Tag dans HTML)</td><td>Xlink (+ Xpointer & Xpath)</td><td>HyTime & TEI</td></tr>
<tr><td>Assemblage</td><td>"calculs server-side"</td><td>XInclude (+ Xpointer & Xpath) ou entités ou "calculs server-side"</td><td>Entités SGML</td></tr>
<tr><td rowspan="2">Style</td><td>CSS2</td><td rowspan="2">XSL
(CSS)</td><td rowspan="2">DSSL</td></tr>
<tr><td>CSS1</td></tr>
<tr><td>Markup</td><td>HTML</td><td>applications XML (XHTML, Docbook)</td><td>applications SGML (Docbook, TEI, ...)</td></tr>
<tr><td>Multimédia</td><td>formats "exotiques" (Flash, Gif, Jpeg)</td><td>formalismes XML (SVG, SMIL, MathML)</td><td></td></tr>
<tr><td>Interface entre Markup et Scripting</td><td colspan="2">Document Object Model (DOM)</td><td></td></tr>
<tr><td>Scripting</td><td colspan="2">Javascript, JScript, ECMAScript, ...</td><td></td></tr>
</table>

4.3 Vocabulaires pour "publishing"

1. Text markup général: Latex en mieux

   Vocabulaires "neutres" mais très détaillés pour rédiger des textes larges. Ces vocabulaires ont souvent leur origine dans le monde "SGML".

- Exemple Docbook **http://www.docbook.org/xml/**
  - répandu dans le monde technique
  - support au niveau des outils XML (et SGML) haut de gamme
  - très détaillé (> éléments)

2. Text markup décentralisé

   Schémas servant à générer et assembler du texte à partir de multiples sources selon besoin

   - Exemple DITA **http://www-106.ibm.com/developerworks/xml/library/x-dita3/index.html**

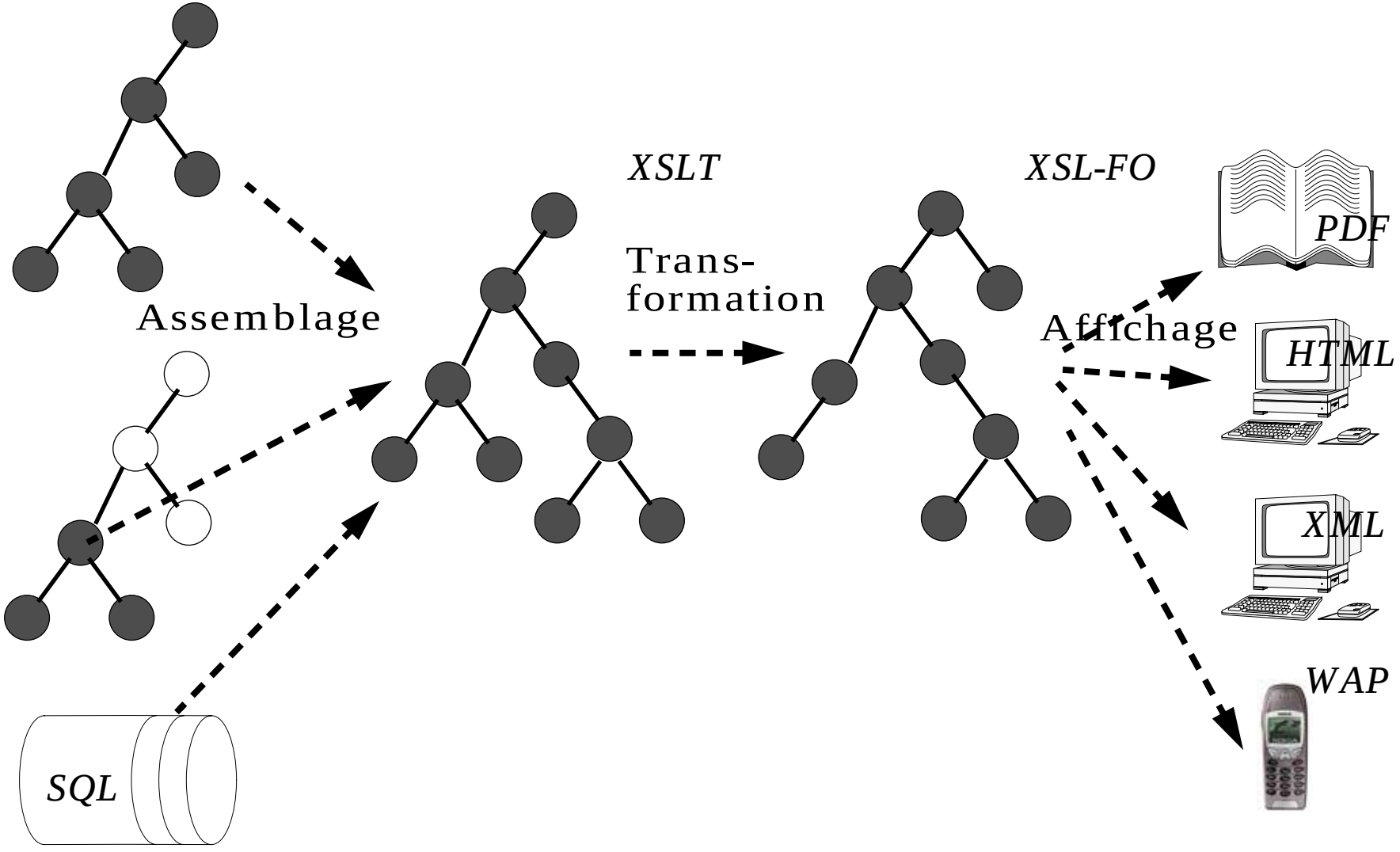
3. Par domaine: SwissDroitML ?

   Vocabulaires pour un domaine précis (souvent juste pour échanger des données)

4. Pour échange de données:
   - par exemple des News
5. Multimédia: SVG, Web3D, MathML, ...

Vers plus d'efficacité, vers un nouveau Babylon, vers une bureaucratisation ???

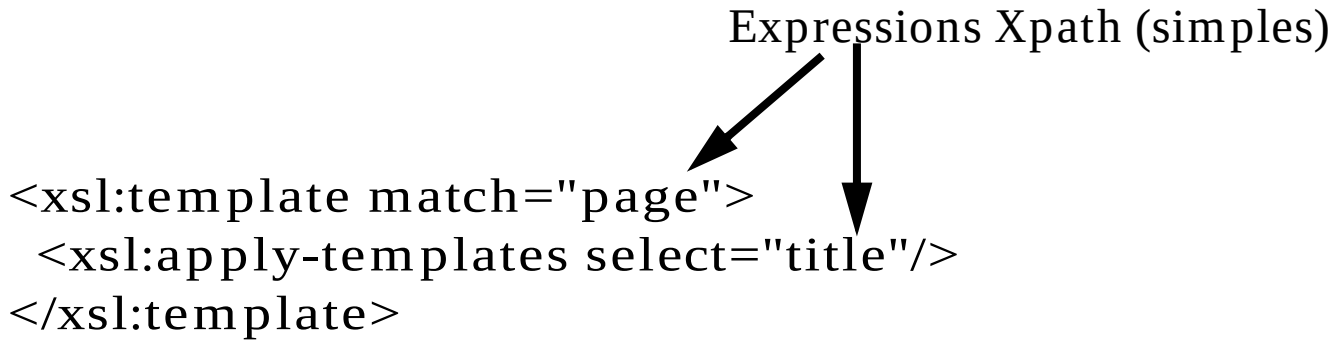
4.4 Exemple d'une chaîne de production "Web publishing"



5. Les outils de base du framework XML

5.1 Xpath: identification de fragments

- XPath est un langage qui permet d'identifier un élément dans un arbre (texte) XML.
  url: **http://www.w3.org/TR/xpath**



- Le principe d'organisation de base ressemble à celui des noms de fichiers (chemins)
  - XSLT utilise XPath pour indiquer à quel élément il faut appliquer une règle
  - Xinclude pour savoir quel fragment "arracher" à un arbre
- Avec XSLT, les débutants peuvent utiliser des simples noms de balises:

```
<xsl:apply-templates select="title"/>
```

- Pour les experts ces expressions peuvent devenir très compliquées:

```
<xsl:apply-templates select="tc-courses/tc-course[position()=1]/course-
module[position()=2]"/>
```

5.2 Assemblage d'un structure XML à partir de plusieurs arbres

- Plusieurs méthodes

A. Avec des entités XML

- Exemples d'entités déclarées:

```
<!ENTITY pm "Patrick Mendelsohn">
<!ENTITY acirc  "&#194;">
<!ENTITY espace " ">
<!ENTITY copyright "&#xA9;">
<!ENTITY explication SYSTEM "citation.xml">
```

Ensuite, on fait une référence du style "&nom_paramètre;":

```
<para> &pm; sort du ch&acirc;teau <para>
<para>Il lit &explication </para>
```

va donner:

```
<para> Patrick Mendelsohn sort du ch&acirc;teau<para>
<para> Il lit:
       <citation> blabla bla </citation>
      <!-- (le contenu du fichier citation.xml -->
</para>
```

B. Xinclude: un langage pour inclure des fragments XML

- utilise Xpath et XPointer

```
<page xmlns:xinclude="http://www.w3.org/1999/XML/xinclude">
 <include xinclude:parse="xml" xinclude:href="proj/proj1/info.xml"/>
<specification>
  <include xinclude:parse="xml" xinclude:href="proj/proj1/
specification.xml#xpointer(//specification/evaluation)"/>
</specification>
.....
   <include xinclude:parse="xml" xinclude:href="proj/proj12/info.xml"/>
......
</page>
```

C. Server-side

- Un programme CGI ou autre peut assembler du XML à partir de n’importe quoi

5.3 XSLT: un langage pour transformations

- La spécification de XSLT est formalisée (W3C Recommendation 16/11/99)
  http://www.w3.org/TR/xslt

Principe de fonctionnement de XSLT

- La transformation du document source (XML) se fait selon des règles XSLT (facteurs conditionnels).
- Une feuille de style XSL est un fichier qui contient un jeu de règles qui déclarent comment traduire des éléments XML (selon leur contexte).

XML vers HTML

- Une règle XSL traduit par exemple un "tag" xml en au moins un "tag" html, mais souvent en plusieurs, par exemple:

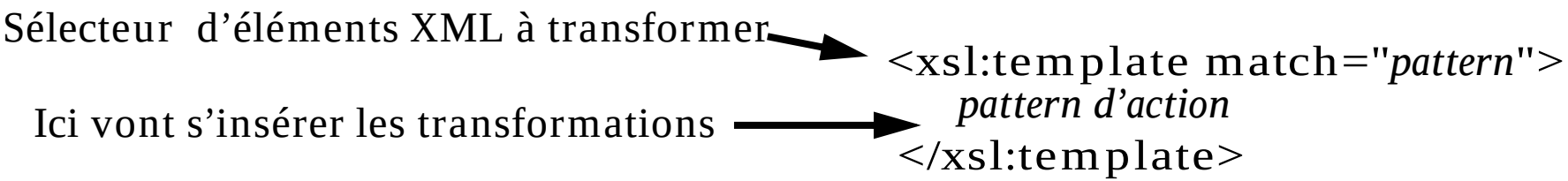
```
<commentaire> xxxx </commentaire>
```

pourrait donner:

```
<DL>
     <DT>Commentaire  </DT>
     <DD> xxxx </DD>
 </DL>
```

- Une feuille de style doit définir une transformation pour chaque tag XML
- L'organisation de la règle qui traite la racine XML est cruciale:
  - Elle définit `<html><head>.....</head><body></body></html>`

Structure d'une règle (template) XSLT:



Définition d'un "template" avec xsl:template

- Une règle "template" est un chablon qui 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 des chemins complets ou absolus

Exemple 5-1: Une simple règle XSLT (XML vers HTML)

- sélectionne <title>, un descendant de <project>
- génère du HTML
- invite le processeur à examiner le contenu de l'élément title

```
<xsl:template match="project//title">
 <h1 align="center"> <xsl:apply-templates/> </h1>
</xsl:template>
```

Anatomie d'un simple style sheet

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

Déclarations XSLT (début de la feuille de style)

```
<xsl:template match="page">
   .......
 <html>
   <head> <title>
        <xsl:value-of select="title"/>
   </title> </head>
   <body bgcolor="#ffffff">
     <xsl:apply-templates/>
   </body>
   </html>
</xsl:template>
```

Règle pour l'élément racine du document

```
<xsl:template match="title">
  <h1 align="center"> <xsl:apply-templates/> </h1>
</xsl:template>
```

Une autre règle

```
.........
```

```
</xsl:stylesheet>
```

Fin de la feuille

5.4 XSLT Patterns de base (A LIRE SOI-MEME)

A. <xsl:apply-templates />

- Un simple apply-templates (sans attributs) examine tous les noeuds enfants
- Dans l'exemple suivant on voit une simple règle pour la racine:

```
<xsl:template match="/">
   <html> <body>
     <xsl:apply-templates/>
   </body> </html>
 </xsl:template>
```

B. L'attribut "select" de apply-templates

- permet de spécifier un élément nommé (au lieu de tous les sous-éléments)
- Dans l'exemple ci-dessous une fois un élément <project> identifié, on traite seulement le sous-élément <title>

```
<xsl:template match="page">
  <xsl:apply-templates select="title"/>
</xsl:template>
```

Cette règle pourrait s'appliquer au texte XML suivant:

url: **http://tecfa.unige.ch/guides/xml/cocoon/simple/hello-page.sxml.text**

C. Extraction d'une valeur

xsl:value-of

- Sélectionne le contenu d'un élément et le copie vers le document "sortie"
- Autrement dit: extraction d'un seul sous-élément (ou sous-attribut)
- Utilisé souvent pour remplir des tables de toutes sortes
- La règle suivante se déclenche dès qu'une balise <projet> est retrouvée et insère dans le document sortie le contenu de l'élément <title> qui se trouve à l'intérieur d'un sous-élément <problem>

```
<xsl:template match="project">
    <P>
      <xsl:value-of select="problem/title"/>
    </P>
  </xsl:template>
```

Syntaxe spéciale pour insérer la valeur d'un objet dans un string d'attribut utilisé dans l'output d'un template: `{....}`

```
<xsl:template match="contact-info">
....
  <a href="mailto:{@email}"><xsl:value-of select="@email"/></a>
...
```

D. xsl:copy

- Copie également les "matched tags" (pas juste le contenu)
  - Utile pour reproduire l'original (ici un tag ...)

```
<xsl:template match="p">
 <xsl:copy> <xsl:apply-templates/> </xsl:copy>
</xsl:template>
```

  - Utile pour récupérer tout ce qui n'a pas été défini, mais attention si le tag ne correspond pas à un tag HTML il faut regarder le source HTML produit !

```
<xsl:template match="*">
   <xsl:copy>Garbage: <i> <xsl:apply-templates/> </i> </xsl:copy>
</xsl:template>
```

E. <xsl:if>

- <xsl:if> Permet de changer l'output en fonction d'un test
- Attention:
  - il n'existe pas de "else" (utilisez "choose" à la place)
  - le mot clé "test" doit apparaître tel quel dans une clause if (valable aussi pour choose)

Exemple 5-2: xsl:if exemple pour insérer des virgules dans une liste

```
<xsl:template match="animal">
  Nom:<xsl:value-or select="@name"/>
  <br> Couleurs:
   <xsl:value-of select="couleur"/>
   <xsl:if test="position()!=last()">, </xsl:if>
</xsl:template>
```

XML:

```
<animal name="zebre"> <couleur>noir</couleur> <couleur>blanc</couleur>
<couleur>bleu</couleur> </animal>
```

Résultat:

```
Nom: zebre
Couleur: noir, blanc, bleu
```

F. <xsl:choose>

- exemple (à élaborer)

```
<xsl:template match="animal">
  <xsl:choose>
    <xsl:when test="@couleur='noir'">
      <P style="color:black">
        <xsl:value-of select="."/>
      </P>
    </xsl:when>
    <xsl:when test="@couleur='bleu'">
      <P style="color:blue">
        <xsl:value-of select="."/>
      </P>
    </xsl:when>
    <xsl:when test="pattern">
       ...
    </xsl:when>
    <xsl:otherwise>
      <P style="color:green">
        <xsl:value-of select="."/>
      </P>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

G. <xsl:for-each>

- sert à faire des tables par exemple

Exemple 5-3: Présentation du résultat XML d'une requête SQL avec XSLT

```
<xsl:template match="ROWSET">
    <table border="2" cellspacing="1" cellpadding="6">
     <xsl:for-each select="ROW">  <tr>
      <td><xsl:value-of select="id"/></td>
      <td><xsl:value-of select="login"/></td>
      <td><xsl:value-of select="fullname"/></td>
       <td bgcolor="tan"><a href="{url}"><xsl:value-of select="url"/></a></td>
      <td><xsl:value-of select="food"/></td>
      <td><xsl:value-of select="work"/></td>
      <td><xsl:value-of select="love"/></td>
      <td><xsl:value-of select="leisure"/></td>
      </tr> </xsl:for-each>
    </table>
</xsl:template>
```

5.5 XSL-FO: un langage pour la mise en page

Spécification de XSL

url: http://www.w3.org/TR/xsl/

But

- Langage de mise en page sophistiqué
- Qualité d'affichage de haut niveau (équivalent à celle d'un bon traitement de texte)
- Adaptation aux média (browser, imprimante, ...)
- Multi-culturel

Usage

- XML + XSLT + XSLFO (-> FO) -> format imprimable / affichable
- FO -> format imprimable / affichable

  (FO est du XML avec XSL-FO mélange)

Statut

- XSL/FO n'est pas encore une recommendation du W3C (mais presque)

A. XSLFO minimaliste

Exemple 5-4: Un petit exemple XML + XSLT + XSLFO complet

Source XML

```
<page>
 <title>Hello Apache/FOP and Apache/Cocoon friends</title>
<content>
Here is some content. It should work with FOP 0.18 (and hopefully above).
It is totally uninteresting. It is totally uninteresting. ... </content>
<content>
   Here is some more content.
   It is slightly uninteresting.  .... :)
</content>
<comment>Written by DKS/Tecfa 6/01</comment>
</page>
```

Feuille de style XSL/FO

```
<?xml version="1.0"?>

<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:fo="http://www.w3.org/1999/XSL/Format" version="1.0" >
```

```
<!-- rule for the whole document: root element is page -->
<xsl:template match="page">
 <fo:root>
  <fo:layout-master-set>
  <!-- Definition of a single master page. It is simple (no headers etc.) -->
  <fo:simple-page-master
          master-name="first"
          margin-left="2cm"  margin-right="2cm"
          margin-bottom="0.5cm" margin-top="0.75cm"
          page-width="21cm" page-height="29.7cm"
          >

   <!-- required element body -->
   <fo:region-body/>
  </fo:simple-page-master>
 </fo:layout-master-set>

 <!-- Definition of a page sequence -->
 <fo:page-sequence master-name="first">
  <fo:flow flow-name="xsl-region-body" font-size="14pt" line-height="14pt">
    <xsl:apply-templates/>
  </fo:flow>
 </fo:page-sequence>
</fo:root>
</xsl:template>
```

```
<!-- A series of XSLT rules that produce fo:blocks to be inserted above -->

 <xsl:template match="page/title">
    <fo:block font-size="36pt" text-align="center" line-height="40pt" space-
before="0.5cm" space-after="1.0cm">
    <xsl:apply-templates/></fo:block>
  </xsl:template>

 <xsl:template match="content">
    <fo:block text-align="justify"  space-before="0.5cm">
    <xsl:apply-templates/></fo:block>
  </xsl:template>

 <xsl:template match="comment">
    <fo:block font-size="12pt" text-align="start" space-before="0.7cm" font-
style="italic">
    <xsl:apply-templates/></fo:block>
  </xsl:template>

</xsl:stylesheet>
```

Sequelette XSL-T/FO de base

```
<xsl:template match="page">
 <fo:root>
  <fo:layout-master-set>

  <!-- Definition of a single master page. It is simple (no headers etc.) -->
  <fo:simple-page-master
          master-name="first"  >
   <!-- required element body -->
   <fo:region-body/>
  </fo:simple-page-master>
 </fo:layout-master-set>

 <!-- Definition of a page sequence -->
 <fo:page-sequence master-name="first">
  <fo:flow flow-name="xsl-region-body" font-size="14pt" line-height="14pt">
    <xsl:apply-templates/>
  </fo:flow>
 </fo:page-sequence>
</fo:root>
</xsl:template>
```

- Une règle XSLT qui définit la racine FO pour la racine XML

Architecture de la racine FO

Sous la racine fo:root il y a toujours:

1. un **`fo:layout-master-set`**
   - qui définit un ou plusieurs page layouts définis avec **`fo:simple-page-master`**
   - qui peut définir des combinaisons de page layouts (page sequence masters)
2. des déclarations à option **`fo:declarations`**
3. une sequénce plusieurs séquences de flux de texte
   - **`fo:page-sequences`** qui contiennent du texte (litéralement ou fournis par XSLT) et des instructions de formattage.

B. On peut faire plus sophistiqué

Exemple d'un layout de type "livre"

```
</fo:layout-master-set>
.......
     <fo:page-sequence-master master-name="run">
         <fo:repeatable-page-master-alternatives maximum-repeats="no-limit" >
             <fo:conditional-page-master-reference
                 master-name="left"
                 odd-or-even="even" />
             <fo:conditional-page-master-reference
                 master-name="right"
                 odd-or-even="odd" />
             <fo:conditional-page-master-reference
                 master-name="title"
                  />
         </fo:repeatable-page-master-alternatives>
   </fo:page-sequence-master>
```

Contenus statiques

```
<fo:static-content flow-name="xsl-region-before">
     <fo:block text-align="end" font-size="10pt" font-family="serif" line-
height="14pt" color="red" > Atelier Webmaster <fo:page-number/> </fo:block>
   </fo:static-content>
```

C. Formattage des blocs

- Le principe ressemble assez fortement à CSS (sauf qu'on utilise une syntaxe XML)
- Note: certains attributs CSS deviennent des éléments XSL ! (listes par exemple)

Exemple 5-5: Formattage CSS vs. XSL

CSS:

```
page > title {
     display: block;
     text-align: center; line-height: 40pt; ....
}
```

XSLT/FO:

```
 <xsl:template match="page/title">
    <fo:block font-size="36pt" text-align="center"
              line-height="40pt" space-before="0.5cm"
              space-after="1.0cm">
    <xsl:apply-templates/></fo:block>
  </xsl:template>
```

FO simple:

```
 <fo:block font-size="36pt" text-align="center"
              line-height="40pt" space-before="0.5cm"
              space-after="1.0cm">
Hello Apache/FOP and Apache/Cocoon friends
</fo:block>
```

6. Cocoon 1

Cocoon est un "publishing framework" basé XML et écrit en Java

- support pour les éléments importants du XML framework: XML, XSLT, XSL-FO, XInclude, Xpath, Xpointer, SVG, XForms.
- "Interfaces" (SQL, LDAP, Xforms, etc.) et XML-embedded scripting
- Certaines fonctionalités (comme XSLT) sont très stables, d'autres sont plus expérimentaux.

Philosophie de base = séparation des tâches:

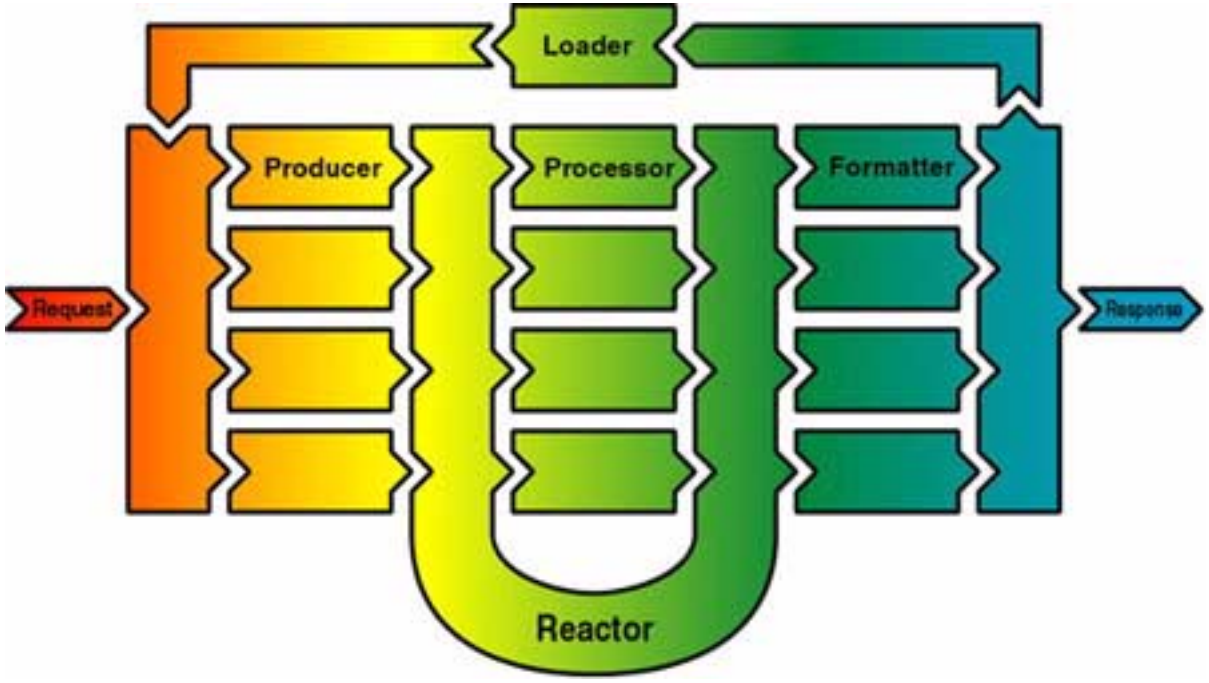
1. Création de XML: Typiquement ces fichiers sont produits par des auteurs / spécialistes de contenu avec un éditeur XML.
2. Traitement de XML: Certains vocabulaires ou tags nécessitent un traitement spécial par un "logicsheet". Imaginez un tag <publications name="dill"> qui fait de sorte à ce que toutes les publications de "dill" soient sorties d'une base de données.
3. Mise en forme du résultat par une feuille de style. Rendering en HTML, PDF, XML, WML, XHTML, VRML, etc. en fonction du client et / ou des besoins

Résumé: Cocoon / XSP permet de séparer contenu, "logique" et style.

(quasi-impossible avec Php, Jsp, Asp etc.)

"Reactor Design" de Cocoon

url: **http://xml.apache.org/cocoon/guide.html (explications)**



Installation

- Un "servlet container" (serveur Java) pas trop viellot
- A TECFA on utilise Tomcat qui tourne comme "esclave" d'un Apache httpd
- Tout fichier *.sxml est envoyé à Tomcat, puis à Cocoon pour traitement.

6.1 XML vers HTML avec XSLT

A. Usages à TECFA

- Cocoon est "en production" à TECFA depuis 1999

Exemple 6-1: Le plan de cours d'une formation continue à TECFA

- DTD fait maison
- Information dans un seul fichier XML
- Plusieurs vues (résumé de la formation, résumé des séances, modules en détail)
- version XSLT -> HTML
- version XSLT + XSLFO -> PDF

Exemple 6-2: Portefeuilles des étudiants

- Ils doivent indexer leurs travaux dans un fichier XML (DTD imposé)
- Ils peuvent adapter la feuille de XSLT à leur goût
- On a des servlets qui vont extraire de l'information de ces fichiers

B. Instructions de traitement Cocoon

Il faut faire un fichier *.xml (sxml à TECFA) et un fichier *.xsl (style)

Vous devez indiquer à Cocoon (selon l'exemple ci-dessous):

- comment traiter le fichier xml (avec des "processing instructions")
- comment utiliser la feuille de style

Entêtes à mettre dans les fichiers (S)XML et XSL:

Fichier XML (extension = *.sxml):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet href="VOTRE_FICHER_XSL.xsl" type="text/xsl"?>
<?cocoon-process type="xslt"?>
```

Fichier XSL (extension = .xsl):

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

// doit aller DANS le template pour la racine XML !!!
<xsl:template match="VOTRE_RACINE">
<xsl:processing-instruction name="cocoon-format">type="text/html"
     </xsl:processing-instruction>
......
```

6.2 LDAP Interface

- Même principe que pour SQL (voir le logicsheet XSP)

```
<?xml version="1.0"?>
<?xml-stylesheet href="ldap.xsl" type="text/xsl"?>
<?cocoon-process type="ldap"?>
<?cocoon-process type="xslt"?>

<page>
  <ldap-defs>
    <ldap-server name="tecfa">
      <initializer>com.sun.jndi.ldap.LdapCtxFactory</initializer>
      <ldap-serverurl>ldap://tecfa2.unige.ch:389</ldap-serverurl>
    </ldap-server>
    <ldap-querydefs name="standard" default="yes"/>
  </ldap-defs>
    <ldap-query  server="tecfa"  ldap-searchbase="o=tecfa.unige.ch"
   defs="standard">
     givenname=Daniel
  </ldap-query>
</page>
```

6.3 Xinclude Processor

Template:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet href="style.xsl" type="text/xsl"?>
<?cocoon-process type="xinclude"?>
<?cocoon-process type="xslt"?>
<page xmlns:xinclude="http://www.w3.org/1999/XML/xinclude">

 <include xinclude:parse="xml" xinclude:href="inclure.xml"/>

</page>
```

6.4 SQL Processor

- démodé (en faveur d'une XSP logicsheet)

6.5 FOP avec Cocoon1

- Le processeur FOP s'utilise avec le processeur XSLT
- Le fichier XSL contient une processing instruction pour l'élément racine et une définition du name-space

Exemple 6-3: Un simple stylesheet XSLT + FO

Fichier XML

```
<?xml version="1.0"?>
<?xml-stylesheet href="hello-page-xslfo.xsl" type="text/xsl"?>
<?cocoon-process type="xslt"?>
```

Fichier XSL / Cocoon

```
<?xml version="1.0"?>
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:fo="http://www.w3.org/1999/XSL/Format"
  version="1.0" >
<xsl:template match="page">
    <xsl:processing-instruction name="cocoon-format">type="text/xslfo"
    </xsl:processing-instruction>

 <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <fo:layout-master-set>
.......
```

6.6 Cocoon - XSP: pages dynamiques à la sauce XML

- XSP = eXtensible Server Pages
  - Une alternative à Php / JSP / ASP / etc.
- Principe plus élégant: on construit un arbre XML au lieu de faire des "print"
- Fonctionne un peu près comme JSP (page compilation en servlets)
- Contient des processeurs spécialisés (XSLT, LDAP, SQL, ...)
- Fournit quelques "tag libraries" appelés "logicsheets" (SQL, Xforms, etc)
- permet d'en ajouter (une feuille de style XSLT qui traduit en Java)

Exemple 6-4: Good Morning ou good Afternoon

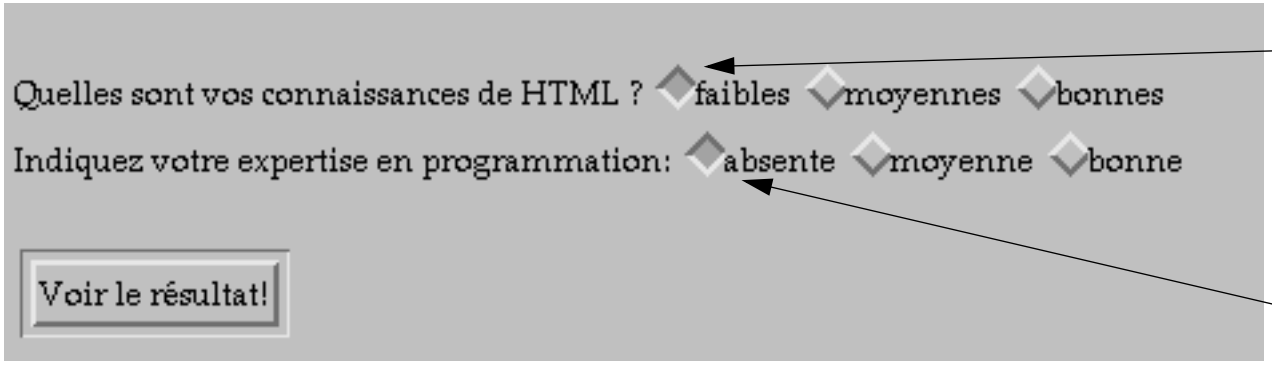
```
<?xml version="1.0"?>
<?cocoon-process type="xsp"?>
<?cocoon-process type="xslt"?>
<?xml-stylesheet href="simple-page-html.xsl" type="text/xsl"?>
<xsp:page language="java"
    xmlns:xsp="http://www.apache.org/1999/XSP/Core"
    xmlns:util="http://www.apache.org/1999/XSP/Util"
  <page>
    <title>Good morning or good afternoon</title>
  <p> It is <util:time format="HH:mm, dd-MM-yyyy"/> </p>
  <p> .... or <util:time format=""/> if you prefer. </p>
 </page>
</xsp:page>
```

A. Anatomie d'une simple page XSP

```
<?cocoon-process type="xsp"?>
<?cocoon-process type="xslt"?>
<?xml-stylesheet href="VOTRE_SHEET.xsl" type="text/xsl"?>

<xsp:page language="java" xmlns:xsp="http://www.apache.org/1999/XSP/
   Core">

<racine_XML>

 <xsp:logic>
 du code Java (comme les scriptlets <% %> en JSP
 </xsp:logic>

<xsp:expr>
  une simple expression Java  (comme les <%= %> en JSP )
</xsp:expr>

</racine_XML>

</xsp:page>
```

B. Traitement de formulaires (GET/POST)

Exemple 6-5: Simple calcul




```
<xsp:page language="java" xmlns:xsp="http://www.apache.org/1999/XSP/Core">
<page>
 <xsp:logic>
    String choice = request.getParameter("choice");
    String choice2 = request.getParameter("choice2");
    int score = 0;
    if ((choice == null) || (choice2 == null)) {
       <xsp:content>Please use the <a href="form.html">form</a> </xsp:content>
      return;
    } else score = Integer.parseInt(choice) + Integer.parseInt(choice2);
 </xsp:logic>
 <title>Form - XSP Demo </title>
 <content>You entered <xsp:expr>choice</xsp:expr> and <xsp:expr>choice2</
xsp:expr>. That makes <xsp:expr>score</xsp:expr>.</content>
</page>
</xsp:page>
```

6.7 SQL avec le logicsheet "ESQL"

Exemple 6-6: XSP/ESQL simple 1

```
<?xml version="1.0"?>
<?cocoon-process type="xsp"?>
<?cocoon-process type="xslt"?>
<?xml-stylesheet href="simple-taglib-html.xsl" type="text/xsl"?>

<xsp:page
  language="java"
  xmlns:xsp="http://www.apache.org/1999/XSP/Core"
  xmlns:esql="http://apache.org/cocoon/SQL/v2" >
<page>
 <title>Cocoon XSP ESQL TagLibs demo</title>
 <author> <name>Daniel Schneider</name> </author>
<p> Shows some of Cocoon’s SQL/XSP tag library. Also shows how to build a simple
table with xsl </p>

  <esql:connection>
   <esql:driver>org.gjt.mm.mysql.Driver</esql:driver>
   <esql:dburl>jdbc:mysql://tecfa.unige.ch/demo</esql:dburl>
   <esql:username>nobody</esql:username>
   <esql:password></esql:password>

 <esql:execute-query>
    <esql:query>SELECT * FROM demo1 order by id;</esql:query>
```

```
      <esql:results>
       <ROWSET>
        <esql:row-results>
         <ROW>
         <esql:get-columns/>
         </ROW>
        </esql:row-results>
       </ROWSET>
      </esql:results>

     </esql:execute-query>
    </esql:connection>
<p> See <a href="./">directory and appended README.html</a> for more
information. </p>
</page>
</xsp:page>
```

- Les balises <ROWSET> et <ROW> sont des balises arbitraires qu'on insère pour "entourer" les résultats d'une balise (sinon on obtient une liste à plat)
- <esql:results> : le "result tree" selon les spécifications
- <esql:row-results> retourne chaque ligne selon les spécifications
- La balise <esql:get-columns> retourne les colonnes d'une ligne entourés des balises portant le nom du label de la colonne

Exemple 6-7: XSP/ESQL simple 2

- Choix selectif de certaines colonnes
- Chaque résultat est inséré dans des balises
  - choisis au "hasard", mais qui doivent évidémment respecter le DTD (réel ou virtuel) de la feuille de style

```
<esql:results>
  <ROWSET>
   <esql:row-results>
    <ROW>
     <id><esql:get-int column="id"/></id>
     <login><esql:get-string column="login"/></login>
     <fullname><esql:get-string column="fullname"/></fullname>
     <url><esql:get-string column="url"/></url>
     <food><esql:get-string column="food"/></food>
    </ROW>
   </esql:row-results>
  </ROWSET>
 </esql:results>
```

7. Remarques finales et outlook

7.1 Evaluation de Cocoon1

- marche à notre satisfaction
  - (mains on utilise peu des éléments avancés, pas de propres tag libs)
- L'utilisation de Processing Instructions (PIs) est génant sur le plan conceptuel (mais tout à fait légale, un processeur ne digère que celles qu'il comprendra)

7.2 Cocoon 2

- En Beta 1 depuis Juin 2001 (sans documentation !)
- Abolit le principe des processing instructions
- Introduit des site-maps centraux par "servlet contexte" et qui indiquent comment traiter des URNs

7.3 Alternatives

- Servlets XSLT, FOP etc. pour des usages plus ponctuels
- Utilitaires "off-line" pour traitements en batch
- Autres produits (Oracle, Microsoft etc.): ok pour XSLT et intégration SQL, mais il leur manque l'élégance de Cocoon.

7.4 Pour plus de détails:

Voir les transparents utilisés dans mes enseignements (suivre le XML trail)

url: **http://tecfa.unige.ch/guides/tie/tie.html**

surtout:

url: **http://tecfa.unige.ch/guides/tie/html/xml-tech/xml-tech.html**

url: **http://tecfa.unige.ch/guides/tie/html/xml-xslfo/xml-xslfo.html**

url: **http://tecfa.unige.ch/guides/tie/html/xml-xslt/xml-xslt.html**

Elliotte Rusty Harold, XML in a Nutshell, O'Reilly, ISBN number is 0-596-00058-8

Portails et Indexes

- Café con Leche: **http://www.ibiblio.org/xml/**
- XmlHack: **http://xmlhack.com/**
- <Xml>fr: **http://xmlfr.org/**
- XML@Tecfa: **http://tecfa.unige.ch/guides/xml/pointers.html** (simple page)

Standards

url: **http://.w3.org/xml**

Le projet Apache/XML et Cocoon

url: **http://xml.apache.org/ (processeurs XSLT, XSLFO, SVG, etc.)**

url: **http://xml.apache.org/cocoon/**