



Mehdi Lababidi  
Service Informatique  
Ecole française d'Athènes

# Porphyre 2000

Dossier du module Structure et Stockage des documents XML

Auteur : [mehdi.lababidi@insa-lyon.fr](mailto:mehdi.lababidi@insa-lyon.fr)

<b>Module</b>	Document / documents
<b>Nom</b>	Structure, Stockage et Formatage des documents
<b>version</b>	0.3

# Sommaire

<b>1</b>	<b>INTRODUCTION .....</b>	<b>3</b>
<b>2</b>	<b>ANALYSE DES BESOINS .....</b>	<b>3</b>
2.1	POURQUOI XML.....	3
<b>3</b>	<b>ANALYSE DES NORMES UTILISES POUR XML.....</b>	<b>3</b>
3.1	XLL ( EXSTENSIBLE LINKING LANGUAGE ) .....	3
3.1.1	<i>X-liens</i> .....	3
3.1.2	<i>X-Pointeurs</i> .....	4
3.2	XPATH.....	5
3.3	DTD ( DOCUMENT TYPE DEFINITION ) .....	5
3.4	XSL.....	5
3.5	LE JEU DE CARACTERES UNICODE .....	6
<b>4</b>	<b>STRUCTURE DES DOCUMENTS.....</b>	<b>6</b>
4.1	DOCUMENT XML .....	6
4.1.1	<i>DTD</i> .....	6
4.1.2	<i>Modèle Objet</i> .....	7
4.1.3	<i>Balises</i> .....	7
4.2	DOCUMENT – META-DONNEES .....	9
4.2.1	<i>Modèle Objet</i> .....	9
4.2.2	<i>Classes implémentant ces données au niveau de l'application</i> .....	9
4.2.3	<i>Stockage de ces informations sur BD</i> .....	10
<b>5</b>	<b>FORMATAGE DES DOCUMENTS .....</b>	<b>10</b>
5.1	DOCUMENT XML .....	10

# 1 Introduction

Ce rapport explique surtout le modèle utilisé pour les documents XML mais il ne faut pas oublier que l'application gère plusieurs formats de documents.

L'architecture des documents XML doit être le plus modulaire possible et permettre une forte réutilisation. Ce module contient deux types d'informations : l'information « substantifique moelle » ( chronique des fouilles et autres ) et les identifiants les pointant. Il est préférable de séparer ces informations. En effet, l'information documentaire doit être complètement indépendante de tout type d'objet la traitant ( en cas de réutilisation ), de plus celle-ci est non modifiable alors que les identifiants évoluent sans cesse ( création, modification ... ). C'est pourquoi dans ce document, ces deux structures sont traitées et implémentées indépendamment.

## 2 Analyse des besoins

### 2.1 Pourquoi XML

12 000 pages de texte seront stockées et seront accessibles par les utilisateurs, ce texte est en plus en lecture seule donc a priori aucun conflit d'accès à gérer. L'origine de l'application est de mettre en ligne la chronique des fouilles sur le réseau ; cependant l'outil devra permettre de s'étendre vers d'autres types de documents. Dans un premier temps, revenons aux sources et essayons de caractériser les besoins sur la chronique des fouilles. Celles-ci décrivent l'état des fouilles et les découvertes de chaque année, le type d'information dans une chronique est donc un ensemble de fragments de texte et d'images indépendantes ; il est donc concevable de fragmenter le texte et éclater chaque tome en plusieurs fichiers dont la granularité reste à définir. La syntaxe XML nous permet dans notre application de structurer l'information via des balises et ainsi conserver un aspect brut de l'information ; ce langage décrit la structure et la signification du contenu d'un document et permet ainsi de ne pas se perdre dans le formatage des données. De plus, XML est associé à de nombreuses technologies permettant un traitement efficace et souple : XSL pour le formatage des balises, XLINK pour la navigation inter-documents, XPOINTEUR pour la navigation intra-documents, DTD pour la structure logique des documents, XML supporte aussi le jeu de caractères UNICODE permettant ainsi de supporter plusieurs langues au sein d'un même document ( dans notre cas, il s'agit du français et du grecque ).

## 3 Analyse des normes utilisés pour XML

### 3.1 XLL ( eXtensible Linking Language )

#### 3.1.1 X-liens

Ce langage permet d'implémenter des liens entre les documents XML. Il est beaucoup plus puissant que le lien HTML. Les X-liens permettent des liens multidirectionnels. Un document peut être uniquement une association de X-liens ; chaque lien réfère à un document XML défini par son URL. Le document est ensuite automatiquement généré par ces liens.

Les attributs :

➤ **xlink: type** : définit le type de lien

*valeur* :

- **simple** : définit un lien unidirectionnel
- **extended** : un lien étendu est un graphe orienté dans lequel les emplacements correspondent à des vertices et les liens entre les nœuds à des arêtes. Chaque vertice est associé à une adresse URI.
- **locator** : fonctionnent avec les liens étendus. Les liens étendus définissent généralement plusieurs cibles. Pour y parvenir, ils stockent les définitions des sources et des cibles dans des sous-éléments localisateurs et non dans un seul attribut **href**.
- **arc**: les attributs **show** et **actuate** des liens simples contrôlent les modalités d'exploitation d'un lien. Les liens étendus sont un peu plus complexes parce que de nombreuses directions sont possibles. Ces différentes possibilités correspondent à des arcs. Chaque traversée d'un emplacement à une autre peut être associée à des règles différentes déterminant quand le lien peut être traversé et ce qui se produit une fois la traversée effectuée.

➤ **xlink : role** : décrit le rôle de la ressource cible

➤ **xlink : title** : décrit le titre de la ressource cible

➤ **xlink : show** : suggère une manière de présenter le contenu lors de l'activation du lien

*valeur* :

- **replace** : remplace le document existant
- **new** : lance une nouvelle fenêtre
- **parsed** : s'insère dans le document

➤ **xlink : actuate** : définit le mode de chargement du lien

*valeur* :

- **user** : activation du lien par l'utilisateur
- **auto** : lien activé automatiquement dès le chargement du lien

### 3.1.2 X-Pointeurs

Les X-pointeurs permettent à un lien de faire une référence à une portion de document sans avoir à modifier ce document ( vs l'insertion d'une ancre dans HTML ). Les X-pointeurs ne se contentent pas de pointer en un seul point d'un document puisqu'ils permettent de cibler des plages ou des étendues. Un X-pointeur permet ainsi de sélectionner un bloc, ce qui permet ensuite de le copier ou de le charger dans un programme.

Exemple de X-pointeurs :

```
xptr (id('ebnf'))
recherche l'élément dont l'identificateur ID est 'ebnf'
```

xptr (descendant ::language[position( )=2]) recherche le premier élément nommé <b>language</b> qui est le deuxième sous-élément de son parent.
---

Le prédicat de recherche suit la **syntaxe XPATH** défini par la suite.

Cette technologie étend les X-liens ; en effet le document n'est pas spécifié dans le X-pointeur ; c'est le X-lien qui le spécifie. En fait, un X-pointeur s'ajoute à la suite de l'adresse URI avec un séparateur #.

Exemple de X-liens avec X-pointeur :

<a href="http://www.w3.org/TR/1998/REC-xml-199.xml#xptr(id('enbf'))">http://www.w3.org/TR/1998/REC-xml-199.xml#xptr(id('enbf'))</a>
---

### 3.2 XPath

Xpath est un langage pour définir les prédicats de sélection dans un document XML. Elle reste très puissante. Je ne m'étendrai pas sur ce sujet. Pour plus d'information, voir les normes définies par le w3c.

### 3.3 DTD ( Document Type Definition )

Le contenu d'un document XML peut être modélisé ou défini via une DTD ; une telle définition fournit la liste de éléments, des attributs, des notations et des entités que contient le document XML ainsi que les règles de relations qui les régissent : en bref, la DTD correspond au modèle structurel des données. Cette DTD est référencé dans le document XML.

### 3.4 XSL

Le langage de style extensible XSL est en fait constitué d'un langage de transformation et d'un langage de formatage. Chacun est une application XML. Le langage de transformation réunit des éléments permettant de définir des règles pour transformer un document XML en un autre document XML. Le résultat peut utiliser le balisage et la DTD du document original ou opter pour un jeu de balises totalement différent. Il peut notamment utiliser les balises qui sont définies par l'autre partie de XSL, constituée d'objets formants.

Le premier langage permet en pratique de transformer un document XML en un document HTML bien formulé ; les capacités de ce langage permettent de convertir des données d'une représentation XML vers une autre.

Pour le deuxième langage ( de formatage ), il s'agit d'une application XML permettant de décrire comment doivent être mise en page les données en sortie de traitement. En général, une feuille de style exploite le langage de transformation de XSL pour transformer un document XML en un autre document XML qui incorpore le vocabulaire des objets formants XSL. Les objets formants ( *formatting objects* ) de XSL constituent un modèle de mise en page visuelle bien plus sophistiqué que HTML avec CSS. Les objets formants XSL offrent des choses uniques, inaccessibles au CSS, comme la mise en page pour des langues non

occidentales, les notes de bas de pages, les notes de marges, les références aux numéros de page dans les références croisées...

### 3.5 Le jeu de caractères UNICODE

XML supporte totalement le jeu de caractères UNICODE sur deux octets ainsi que ses représentations compactes ; UNICODE permet de gérer l'essentiel des caractères romains et non romains actuellement utilisés dans différents pays.

Comment écrire du XML en UNICODE :

UNICODE est le jeu de caractères natif du XML. Mais il n'existe pour l'instant pas d'éditeur de textes supportant la totalité du jeu UNICODE. Pour gérer ce problème, il existe deux solutions :

- Rédiger le texte avec un jeu de caractères spécifiques tel que Latin-3 puis le convertir en UNICODE grâce à une application spécifique tel que native2ascii du kit Java JDK de Sun.
- Insérer des références à des caractères UNICODE dans le texte pour identifier de manière numérique des caractères spéciaux. ( ex : au sein d'un document XML &#960 sera le code UNICODE de la lettre grecque pi ).

## 4 Structure des documents

Dans ce rapport, on décrit uniquement la structure des documents XML or il ne faut pas oublier que l'application permet de charger d'autres type de documents. A cet instant, les formats de documents supportés sont le XML, HTML, RTF pour le texte ( donc auxquels on peut appliquer le stabylo ) et pour les images le GIF, JPEG (JPG). Mais à l'origine, la chronique des fouilles sera stocké sur des documents XML.

### 4.1 Document XML

La structure XML des documents doit être le plus simple possible pour ne pas surcharger les données d'informations non spécifiques. Les quelques balises qui agrémenteront les documents XML définiront uniquement la structure logique ( de titrage ).

#### 4.1.1 DTD

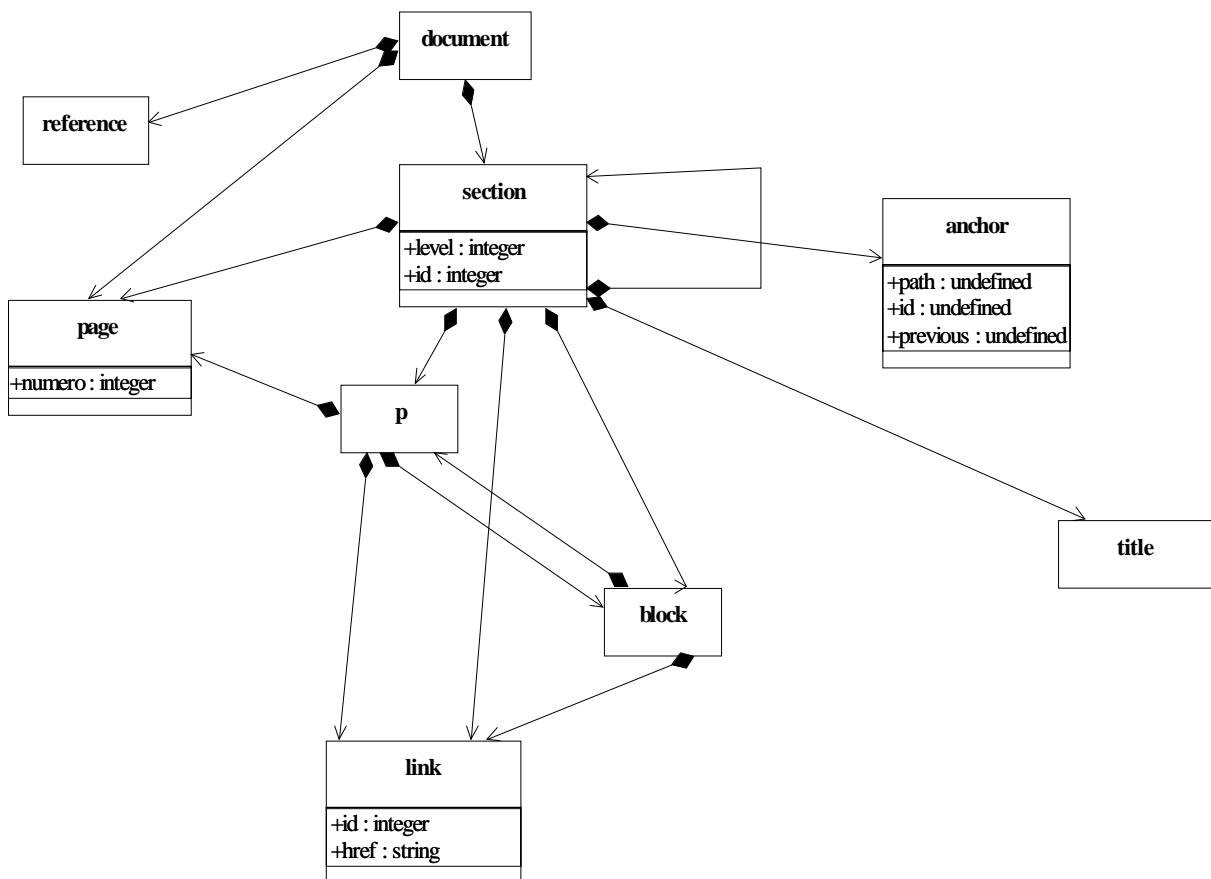
```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT document ((page* | reference+ | section+))+>
<!ELEMENT section (anchor+, (title+ | block* | link* | p* | page* | section*))+>
<!ATTLIST section
  id CDATA #IMPLIED
  level CDATA #REQUIRED
>
<!ELEMENT title ANY>
<!ELEMENT block (link,p)+>
<!ELEMENT p ANY>
```

```

<!ELEMENT page EMPTY>
<!ATTLIST page
  number CDATA #REQUIRED
>
<!ELEMENT link (#PCDATA)>
<!ATTLIST link
  href CDATA #REQUIRED
  id CDATA #IMPLIED
>
<!ELEMENT anchor EMPTY>
<!ATTLIST anchor
  path CDATA #REQUIRED
  id CDATA #REQUIRED
  previous CDATA #IMPLIED
>
<!ELEMENT reference (#PCDATA)>
<!ELEMENT b (#PCDATA)>
<!ELEMENT i (#PCDATA)>
<!ELEMENT u (#PCDATA)>

```

#### 4.1.2 Modèle Objet



#### 4.1.3 Balises

**document** : définit le nœud racine de l'arbre

**reference** : définit la référence du document

**page** : indique uniquement un changement de page et donne son numéro.

**section** : une section définit une vue, un contexte autosuffisant. L'attribut **id** permet de l'identifier dans le document, utile dans la syntaxe Xpath. L'attribut **level** donne le niveau de titre de la section. Une section doit contenir une ou plusieurs ancres et un titre. L'attribut **ID** est attribué lorsque l'utilisateur charge le document dans la base. Celui-ci prend la valeur de l'identifiant généré qui l'identifiera à l'avenir dans toute l'application.

**anchor** : Utile uniquement lors du chargement du document dans la base, ces ancres déterminent le chemin de descripteurs associés à la section. Ex : toto/tutu/titi ; l'identifiant de la section se verra affecté au niveau du graphe sur le descripteur titi. L'attribut **path** définit le chemin, l'attribut **previous** indique l'ancre père auquel le premier descripteur doit se rattacher, la valeur **previous** peut prendre plusieurs valeurs. Ex :

```
<section id= « 1 »>
  <anchor path= "titi/tutu" id="1"/>
  <anchor path="toto/tyty" id="2"/>
  <titre>
    blabla
  </titre>
</section id="2">
  <anchor path="truc/muche" id="3" previous="1/2" />
  ...
```

Au niveau du graphe, on aura donc :



**title** : son contenu définit uniquement le nom du titre.

**p** : définit un paragraphe, il peut contenir des pages, d'autres paragraphes ou block, des links et diverses balises de formatages de textes tels que `<i>` pour italique...

**link** : Bien que les images ne soient pas stockées au même endroit, pour garder l'intégrité du document sur les références du texte sur les images. Il est possible d'intégrer cette balise dans n'importe quelle autre balise, et de ce fait l'image lui sera associée. Cette balise n'est pas spécifique à un format de document, l'attribut **href** définit le nom du document. Lorsque l'utilisateur charge le document dans la base, l'application teste si le nom de document est bien enregistré dans la base de stabylo et retourne son numéro d'identifiant respectif, des lors l'attribut **ID** du lien prendra cette valeur.

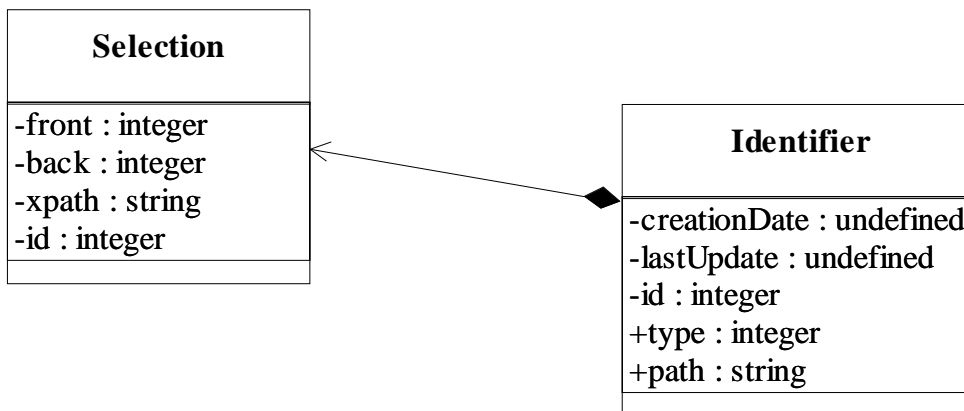
**block** : lorsque l'on veut associé une image à un paragraphe, cette balise construit un tableau dans lequel la première case prend la valeur du lien et la deuxième le paragraphe.

## 4.2 Document – Meta-Données

Ce paragraphe décrit toutes les données stockées pour chaque document, ne s'applique donc pas exclusivement aux documents XML.

Meta-Données quoi ? pour l'instant pas grand chose, alors essayons d'éclaircir la signification de ce « meta-données ». Comme nous l'avons vu précédemment, le document ne renseigne en rien sur les fragments. Rappelons que lorsque l'utilisateur demande un document, il renseigne au serveur document l'identifiant du fragment qu'il veut manipuler. A cet identifiant doit correspondre toutes les données complémentaires sur le fragment ; ces données étant modifiables, il fallait donc a priori les détacher du fragment cible. Les identifiants sont des données évolutives il faut donc gérer les conflits d'accès, et il est beaucoup plus facile de le gérer sur une base de données. Pour chaque document référencé par le graphe ( identifiant ), il existe un n-uplet dans la table Stabylo qui donne toutes les informations nécessaires pour extraire le document.

### 4.2.1 Modèle Objet



### 4.2.2 Classes implémentant ces données au niveau de l'application

**Identifier** : regroupe toutes les informations sur le fragment cible. Il est identifié par un **ID**. Le **path** donne le nom du document. Cas particulier du fichier XML ou le **path** peut avoir un xpoiteur en suffixe ; celui-ci délimité par un caractère spécifique. Ex : doc.xml#xptr(id(2)) ce chemin pointe sur la section identifié par le numéro 2 dans le document doc.xml. Un identifiant peut posséder une sélection qui détermine à la suite la sélection fluorescente dans l'éditeur de document.

**Selection** : définit la zone de sélection du fragment. Cette sélection se définit uniquement dans la vue. Les attributs **front** et **back** correspondent au numéro de caractères de début et de fin de la sélection à partir de la zone pointé par le **xpath**. Le **xpath** définit le chemin du nœud

contenant le texte à sélectionner. Celui-ci est nul lorsque le document est du type autre que XML ; il est utile uniquement pour que la sélection reste indépendante de la feuille de style XSL.

### 4.2.3 Stockage de ces informations sur BD

Ces informations sont donc stockées sur une BD, les deux tables les représentant se nomment **Stabylos** pour **Identifier** ( pas logique mais pas eu le temps de le changer ) et **Selections** au cas ou le fragment de document possède un stabylo.

ATTENTION à cette confusion !!!!

## 5 **Formatage des documents**

Chaque type de document à a priori son propre formatage implémenté dans les bibliothèques java :

- RTF / HTML avec le JeditorPane
- JPEG / GIF avec l'objet ImageIcon

### 5.1 *Document XML*

Seul le document XML doit être formaté en un format de présentation standard. C'est pour quoi nous utilisons une feuille de style XSL qui formate le document XML en sortie HTML.

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >
  <xsl:template match="/">
    <html>
      <head>
        <style type="text/css">
          .1 {margin-left:5pt;font-family : serif; font-size:30pt;font-style:italic;}
          .2 {margin-left:20pt;font-family : arial; font-size:25pt;color:red;}
          .3 {margin-left:30pt;font-family : serif; font-size:20pt;color:blue;}
          .4 {margin-left:40pt;font-family : serif; font-size:15pt;color:teal;}
          .5 {margin-left:45pt;font-family : serif; font-size:12pt;color:black;}
          .para {font:10pt Verdana;margin-left:45pt;color:black}
          .para2 {font:10pt Verdana;margin-left:130pt;color:black}
          .link {color:teal;font-family : courier,times;font-size:12pt}
          .ref {font-size:20pt;font-weight:bolder;margin-left:20pt;color:teal;font-
family : courier,times;}
          .page {font-size:20pt;font-weight:bolder;margin-left:20pt;color:teal;font-
family : courier,times;}
          u {text-decoration:underline}
        </style>
      </head>
      <body>
        <hr/>
        <xsl:apply-templates select="document"/>
        <hr/>
      </body>
    </html>
  </xsl:template>
  <xsl:template match="document">
    <xsl:apply-templates select="reference"/>
    <xsl:apply-templates select="page"/>
    <xsl:apply-templates select="section"/>
  </xsl:template>
  <xsl:template match="section">
    <div>
      <xsl:attribute name="ref"><xsl:value-of select="@ref"/></xsl:attribute>
      <xsl:apply-templates select="title"/>
      <xsl:apply-templates select="block | p | img"/>
      <xsl:apply-templates select="section"/>
    </div>
  </xsl:template>
</xsl:stylesheet>
```

```

</xsl:template>
<xsl:template match="title">
  <div>
    <xsl:attribute name="class"><xsl:value-of select="../@level"/></xsl:attribute>
    <xsl:value-of select="."/>
  </div>
</xsl:template>
<xsl:template match="block">
  <table>
    <tr>
      <td WIDTH="80" ALIGN="CENTER">
        <xsl:for-each select="link">
          <xsl:apply-templates select="."/><br/>
        </xsl:for-each>
      </td>
      <td class="para">
        <xsl:for-each select="p">
          <!--<xsl:attribute name="ref"><xsl:value-of select="@ref"/></xsl:attribute-->
          <xsl:apply-templates/><br/>
        </xsl:for-each>
      </td>
    </tr>
  </table>
</xsl:template>
<xsl:template match="p">
  <div class="para2">
    <!--<xsl:attribute name="ref"><xsl:value-of select="@ref"/></xsl:attribute-->
    <xsl:apply-templates/>
  </div>
</xsl:template>
<xsl:template match="b">
  <b><xsl:apply-templates/></b>
</xsl:template>
<xsl:template match="i">
  <i><xsl:apply-templates/></i>
</xsl:template>
<xsl:template match="u">
  <u><xsl:apply-templates/></u>
</xsl:template>
<xsl:template match="font">
  <font>
    <xsl:attribute name="size">
      <xsl:value-of select="attribute::size"/>
    </xsl:attribute>
    <xsl:apply-templates/>
  </font>
</xsl:template>
<xsl:template match="reference">
  <div class="ref">
    Reference : <xsl:value-of select="@doc"/>
  </div>
</xsl:template>
<xsl:template match="page">
  <div class="page">
    Pages :<xsl:value-of select="@debut"/>-<xsl:value-of select="@fin"/>
  </div>
</xsl:template>
<xsl:template match="link">
  <a class="link">
    <!--<xsl:attribute name="ref"><xsl:value-of select="@ref"/></xsl:attribute-->
    <xsl:attribute name="href">
      <xsl:value-of select="attribute::id"/>
    </xsl:attribute>
    <xsl:value-of select="."/>
  </a>
</xsl:template>
</xsl:stylesheet>

```

RECOMMENDATIONS :

Toutes les informations contenues dans la page HTML finale doivent être contenues dans les nœuds textes du document XML et non dans les attributs. Cf. implémentation.