Nicolas Van Wambeke

Digitally signed by Nicolas Van Wambeke
DN: CN = Nicolas Van Wambeke, SN = Van Wambeke, G = Nicolas
Reason: I am the co-author of this document
Location: Toulouse, FRANCE
Date: 2005.07.16 23:21:45 +01'00'

XML
« eXtensible Markup Language »

Kiami Mark
Van Wambeke Nicolas

TABLE DES MATIERES

INTRODUCTION

XML (eXtensible Markup Language) est un méta-langage, c'est à dire un langage permettant de définir d'autres langages. Les langages qu'il définit sont des langages de présentation de documents. A l'aide de XML on peut définir des types de documents. Par exemple modèle de lettres d'embauche, documentation pour automobiles, recettes de cuisine, etc. XML ne donne pas de règles de présentation d'un document. C'est XSL (eXtensible Style Language) et CSS (Cascading Style Sheets) qui s'en chargent.

A la fin des années 60, IBM développe un méta-langage pour documents : GML ("General Markup Language"). En 1978, the American National Standards Institute (ANSI) établit la première version de SGML. Ensuite, en 1986 le premier standard est effectué par Anders Berglund, du CERN. Et c'est cette année que SGML devient un standard ISO. Il faut attendre 1996 pour le début du développement de XML. Enfin, en Février 1998, XML devient un standard W3C.

Ces différents "Markup Language" ne se situent pas au "même niveau". SGML (Standard Generalized Markup Language) comme XML permettent de définir des grammaires de documents (i.e. ensemble de règles que doit vérifier un document pour être "conforme"). Une telle grammaire est appelée une DTD (Document Type Definition).

HTML est une de ces DTD. La grammaire HTML a été définie par SGML. Une "page" HTML est un document vérifiant cette grammaire.

Remerciements à M. MARRE

I. Description de XML

A. Présentation & Structure de XML

XML, acronyme anglais qui signifie « eXtensible Markup Language » (langage de balisage extensible en français) est un méta langage de description. Il permet de hiérarchiser et de structurer le contenu d'un document de façon simple par l'utilisation de balises et d'attributs. XML est un méta langage ce qui signifie qu'il permet de définir la grammaire de nouveaux langages.

Le développement de XML est récent, en effet, il a débuté pendant le courrant de 1996 et la norme XML est reconnue par le W3C (World Wide Web Consortium, organisme de normalisation du web) depuis 1998.
Descendant direct de SGML, langage établi depuis 1986 en tant que norme ISO et utilisé pour de nombreux projets de documentation de taille importante. XML apporte une plus grande extensibilité et permet d'avoir une cible plus large que SGML souvent réservé aux documentations techniques.

De plus, XML est modulaire ce qui rend son utilisation possible dans un grand nombre de domaines. En effet, l'utilisateur de XML peut définir sans limites la structure que prendra un document. Par exemple, un vendeur de voitures pourrait utiliser un document XML tel que celui ci-dessous sans que cela ne pose de difficultés d'implantations

```
<?xml version="1.0" encoding="iso-8859-1" ?>
  <voiture>
  <généralités>
       <marque>Renault</marque>
       <modèle>Twingo Bic</modèle>
       <année>1999</année>
  </généralités>
  …etc..
  </voiture>
```

B. Définition de Grammaires

Il est possible de définir une structure de données et une grammaire propre à un type de document XML. Il est possible de spécifier les éléments d'un type, leurs attributs, ainsi que leurs relations, leurs ordres et leurs fréquences d'apparition dans le document XML

Il existe deux formes de définition de types de documents :
- La DTD (Document Type Definition)
- Le XML-Schema

Ces deux standards ont un même but : permettre la définition d'un type de document en détaillant, pour chaque balise autorisée un certain nombre d'attributs tels le format de données attendu, la taille maximale ou encore le caractère obligatoire ou facultatif de tel ou tel élément.

1. *Document Type Definition*

La Document Type Definition (Définition de Type de document) est l'une des premières méthodes de définitions des modèles associés à XML. En effet, elle existe depuis les débuts du développement de XML.

Une DTD décrit la grammaire d'un type de document. Par l'intermédiaire d'un langage proche de la syntaxe XML, il est possible de définir tous les éléments de la grammaire ainsi que leur caractéristiques.

Il est possible de définir une DTD soit en l'incluant directement dans le fichier XML (DTD interne), soit en créant un fichier séparé et en spécifiant son existence au sein du document XML (DTD externe).

Exemple :

DTD externe	DTD interne
<? xml version="1.0" standalone="no"?> <!DOCTYPE list SYSTEM "fichierDTD.dtd">	<? xml version="1.0" standalone="yes" ?> <!DOCTYPE list [… detail de la DTD …]>

Une DTD commence toujours par **<!DOCTYPE** et se termine par **]>**.

La DTD est construite à partir d'un ensemble de déclarations permettant de définir le type, la nature et les contraintes liées à chaque nouveau marqueur :

- Déclaration des **éléments** [ELEMENT] : Ils permettent de définir le contenu du fichier XML.
- Déclaration des **attributs** [ATTLIST] : Ils permettent d'enrichir la sémantique des éléments.
- Déclaration d'**entités** [ENTITY] : Elles permettent de définir des alias sur du contenu
- Déclaration de **notations** [NOTATION] : Elles permettent de définir des éléments que doit ignorer le parseur, comme des données binaires par exemple.

- ***DECLARATION DES ELEMENTS***

Les déclarations de type élément permettent de décrire le vocabulaire utilisable dans un document XML. Elle permet également de contraindre les éléments utilisés dans l'application et de spécifier l'ordre dans lequel ces derniers doivent apparaître. Ils correspondent aux marqueurs du document XML. Un élément est désigné par un nom et il peut contenir d'autres éléments, des données ou être sans données.

Notation :

<!ELEMENT **nom type**> où :

- **nom** est le nom de l'élément, il peut être défini à partir de lettres, de chiffres ou des caractères "**:**", "_", "**-**", "**.**", sans ne jamais commencer par un chiffre.
- **type** est le type de l'élément :

EMPTY	Permet de déclarer un élément vide qui n'a pas de contenu. Il peut cependant posséder des attributs. *Ex :* <!ELEMENT fin_de_serie **EMPTY**>, définit un élément *fin_de_serie* sans contenu et implémenté <fin_de_serie/>.
ANY	Permet de déclarer un élément qui contient des chaines de caractères ou n'importe quels autres éléments déclarés dans la DTD, apparaissant dans n'importe quel ordre. *Ex :* <!ELEMENT jeu_d_echec **ANY**>, définit un élément *jeu_d_echec* contenant des chaînes de caractères ou des éléments déclarés dans la DTD.
ELEMENT	Permet de déclarer un type contraint à un contenu défini par uniquement d'autres éléments, exprimés entre parenthèses "**( )**", spécifiés dans un ordre précis, avec le caractère "**,**", ou selon un choix "|". Ex 1 : <!ELEMENT jeu_d_echec **(**auteur**,** date_de_creation**,** atelier**,** composants**)**>, définit un type *jeu_d_echec* contenant un les éléments *auteur*, *date_de_creation*, *atelier*, *composants* dans cet ordre.
MIXED	Permet de déclarer un type contraint à un mélange d'autres éléments et de texte (#PCDATA, Parsed Character DATA) dans un ordre établi. *Ex :* <!ELEMENT ville (#PCDATA, nom) > <!ELEMENT nom (#PCDATA) >, définit un élément *ville* contenant du texte suivi du *nom* de la ville, et implementé <ville> ici on peut mettre du texte <nom>Rennes</nom></ville>

- ***DECLARATION D'ATTRIBUTS***

La définition de listes d'attributs - **ATTRIBUTES** - permet de définir des couples nom-valeur à des éléments de la DTD afin de les compléter ou de les enrichir. Parmi les utilisations possibles des listes d'attributs, l'utilisateur a la possibilité de :

- définir les attributs d'un type d'élément,
- d'établir des contraintes de type pour les attributs qu'ils définissent,
- fournir des valeurs par défaut pour les attributs.

Notation :

<!ATTLIST **nomliste**
nom type [valeur] [#attribut]
...
nom type [valeur] [#attribut]>

où :

- **nomliste** est le nom de l'élément auquel l'attribut est rattaché
- **nom** est le nom de l'attribut
- **type** est le type de l'attribut, il peut être :

CDATA	Character DATA, le contenu de l'attribut doit contenir du texte.
ID	Le contenu de l'attribut est unique parmi l'ensemble des attributs d'un même élément ; la valeur doit commencer obligatoirement par une lettre ou "_" et ne doit contenir aucun espace.
NMTOKEN	Utilise les mêmes conventions de nommage que ID mais les doublons sont autorisés
Liste Enuméré	Elle n'a pas de désignation implicite mais reste utilisée avec "(valeur 1|valeur2|..)"

- **valeur** est la valeur par défaut de l'attribut, l'utilisateur peut en spécifier une par défaut et en utiliser une autre lors de la saisie du document XML. Ce champ est optionnel, s'il n'est pas renseigné, l'attribut n'a pas de valeur par défaut.
- **#attribut**, cette instruction peut prendre 3 valeurs :

#REQUIRED	Rend obligatoire la saisie de l'attribut.
#FIXED	Définit une valeur qui ne peut pas être changé dans le document XML.
#IMPLIED	Signifie que l'attribut peut contenir une valeur ou être nul.

Exemple : Définit un type *image* qui contient un attribut *largeur* avec une valeur fixée à 100 pixels et une *hauteur* dont la valeur n'est pas précisée mais qui reste obligatoire.

```
<!ELEMENT images (#PCDATA)>
<!ATTLIST images
    largeur #CDATA "100 pixels" #FIXED
    hauteur #CDATA #REQUIRED >
```

Pour une utilisation : <images largeur="100 pixels" hauteur="50 pixels"> toto.gif </images>

- ***DECLARATION D'ENTITES***

Les entités permettent de mettre en place un système de substitutions de contenu (texte ou autre) analysable ou non par le parseur XML. Cela présente l'avantage d'un gain de place et de temps pour le rédacteur de la DTD, puisqu'il peut associer une symbolique à une information récurrente tout au long du document de manière homogène. La déclaration d'une entité consiste juste en la définition d'un nom et du contenu qui s'y rapporte, il est similaire à la déclaration de constantes en programmation classique.

Au niveau XML, il existe 5 entités prédéfinies qui permettent aux rédacteurs de pouvoir utiliser des caractères utiles à la création de documents sans poser de problèmes dans la constitution du document XML lui-même. En effet, & est codé & , < est codé < , > est codé > , « est codé " et enfin l'apostrophe est codée ' .

Notation :

<!ENTITY nom valeur> où :
- nom est le nom a employer pour l'entité.
- valeur est la valeur de l'entité.

L'utilisation d'une telle entité dans le document XML se fera par l'utilisation combinée des caractères '&' et ';' avec une notation de la forme &nom; où nom est le nom de l'entité.

Par exemple: <!ENTITY pe "philippe ensarguet">.
Son utilisation dans les documents XML sera de la forme &pe; et reviendra à chaque fois à écrire : « philippe ensarguet ».

Le contenu d'une entité peut être un simple bout de texte, mais il peut également être beaucoup plus long et plus complexe, alors l'utilisateur peut faire appel à un fichier :

<!ENTITY nom SYSTEM "http://mon.url.verslefichier/fichier">

On peut également utiliser des entités pour construire les éléments ou leurs attributs dans une DTD. On les appelle alors **entités paramètres** (parameter entities).
L'exemple suivant permet de définir une entité information pouvant être un élément nom, numéro ou référence. Cette entité pourra être utilisée pour contribuer à définir un élément information, comme suit :

<! ENTITY % information "(nom|numero|reference)">

avec comme utilisation :

<!ELEMENT auteur (%information;|trigramme)>

- ***DECLARATION DE NOTATIONS***

La déclaration de notation permet à des documents XML de se référer à des sources d'informations externes, "non XML". On peut par exemple se servir de la déclaration de notation pour faire référence à des « helpers » ou « plug-ins » utilisé par des données du document XML.

Notation :

<!NOTATION nom SYSTEM valeur>

Exemple : <!NOTATION PRG SYSTEM "http://mon.url.duprogramme/prg.exe">

Il s'agit ici de la déclaration de la notation PRG associée au programme prg.exe situé à l'URL indiqué pour voir des données de la notation PRG. Un document XML qui veut utiliser des données dans cette notation peut les référencer par une référence externe ou il peut l'inclure directement dans le fichier XML en tant que données d'un élément de type NOTATION.

Les handlers déclarés avec les NOTATIONS peuvent être rattachés à des ENTITES pour traiter les fichiers externes au document XML.

Exemple :

<!NOTATION **gif** SYSTEM "gif_viewer.exe">
<!ENTITY mongraph "http://mon.url.com/graph.gif" **NDATA gif**>

Cette notation permet de définir un viewer de fichier graphique au format GIF, et de le rattacher à chaque référence déclarée dans une entité.

2. XML Schéma

Les schémas XML se présentent comme une alternative aux DTDs permettant d'apporter des réponses aux limites de ces dernières. Comme une DTD, un schéma permet de définir un ensemble de règles visant à définir un document XML, et notamment les marqueurs autorisés, leurs attributs et les relations des uns par rapport aux autres. Mais contrairement à une DTD, un schéma permet de définir des types pour les données.

De plus, un Schema XML est un document XML à part entière, il peut donc être édité et manipulé à partir de n'importe quel outil d'édition ou de traitement XML.

Un exemple d'amélioration notable est qu'avec une DTD on peut définir un marqueur <NOM>, mais son contenu peut être n'importe quel type de données. Toute la puissance des schémas réside dans sa possibilité à typer le marqueur. On pourra ainsi facilement obliger un utilisateur à ne saisir que des caractères pour le marqueur <NOM>.

Cette nouvelle approche est intéressante, entre autres lors d'échanges entre systèmes hétérogènes, par exemple deux applications communiquant entre elles sur un même ordinateur mais aussi via un réseau ou encore sur internet (WebServices). En effet, cela

permettra d'échanger des données au travers d'un format unique indépendant des applications qui communiquent entre elles : XML, mais surtout cela permettra de vérifier le contenu transféré à l'aide du typage de données.

Pour l'instant, le développement des Schémas est encore en cours. Le consortium W3C qui s'occupe de mener à bien le projet de définition des normes XML Schéma étudie actuellement la définition de la version 1.1 de ceux-ci.

Les principaux désavantages des DTD devant les Schémas sont les suivants :

- Une DTD est difficile à lire
- Une DTD est non extensible, car ce n'est pas un document XML
- Une DTD ne permet pas de typer les données
- Une DTD ne supporte pas les espaces de nommages (Namespace)
- Une DTD est plus concise...mais moins riche qu'un Schema XML

C. Validité de la structure & Parseurs

Pour assurer la compatibilité des documents XML produits par différentes sources, il est important de s'assurer de la validité de la structure. Il existent pour cela divers outils permettant de vérifier qu'un document XML respecte bien les contraintes de forme et syntaxe présentes dans la définition de la grammaire.

Cette fonction de vérification des documents est une des composantes principales de toute application utilisant XML et elle est généralement réalisée par le Parseur.

- ***PARSEURS***

Un parseur XML est la couche logicielle indispensable dans toutes les applications informatiques qui représentent des données XML. Il permet le passage d'un flux balisé de données qui est le XML à une représentation interne propre à l'application.

Un parseur ne peut traiter qu'un document XML bien écrit, et annonce toutes les erreurs de malformation dans la structure.

Il existe aussi des parseurs validants. Ils comparent le document XML à sa DTD et annoncent les erreurs dites de validation. Avec l'émergence des XML-Schéma, ces parseurs sont développés pour comparer un document XML à son Schéma.

Deux grandes catégories de Parseurs : SAX (Simple API[1] for Xml)
DOM (Document Objet Model)

[1] API : Application Programming Interfaces

- **SAX (*Simple API for XML*)**

Effectue son traitement par “événements” de la transmission des données, c'est-à-dire, il associe des événements à la rencontre de certaines structures syntaxiques dans le flux des balises. Chaque événement est associé à un certain nombre d’items d’informations, liés par exemple aux noms des valeurs d’attributs associés à une balise ouvrante. L’API SAX n’est pas une norme W3C, mais ce parseur est largement utilisé dans les communautés de développeurs d’outils XML.

- **DOM (*Document Object Model)***

Un parseur DOM modélise un document XML par un arbre d’objets typés et propose des possibilités de navigation et de manipulation d’arbre, comme la création et modification de l’arbre donc du document XML. L’API DOM fait partie des normes W3C.

Cette technique permet de se ramener à des structures de données algorithmiques qui sont connues de tous les programmeurs et qui possèdent l’avantage d’être présentes depuis longtemps.

- ***Les parseurs aujourd’hui***

Les parseurs SAX sont généralement plus économes en mémoire que les parseurs DOM, car ces derniers utilisent souvent de grands espaces mémoire pour représenter la totalité de l’arbre XML.

Il existe à l’heure actuelle de nombreux parseurs aux normes SAX ou DOM pour les langages les plus communs. On peut citer par exemple, AdaXML qui est une implémentation SAX pour Ada ou encore Xerces, développé par le projet « Apache » qui propose des parseurs aux normes SAX et DOM pour C/C++/Java.

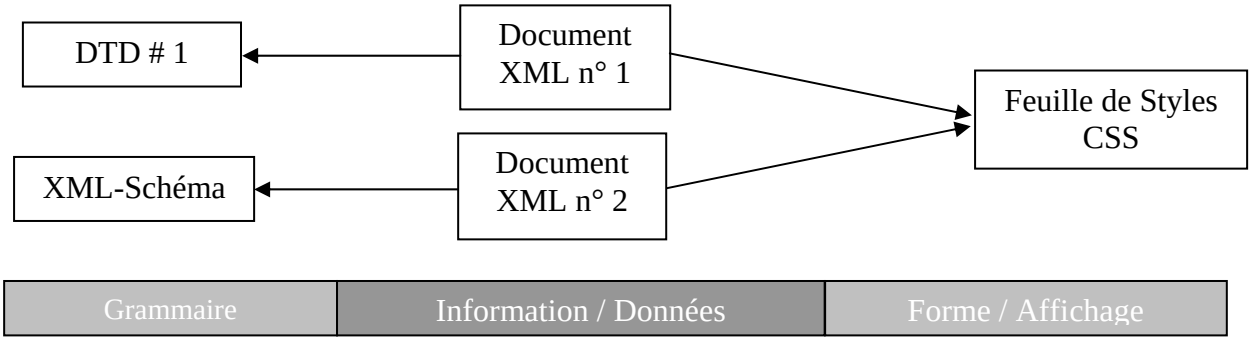
II. Languages Outils autours de XML

A. LE RENDU : CSS

1. Présentation de CSS

CSS veut dire « Cascading Style Sheets ». Les styles possibles avec CSS définissent comment l’information peut être représentée visuellement, donc ce langage permet de définir la forme que peut avoir une information.

Normalement, les styles sont stockés dans des « Style Sheets » ou feuilles de styles en français. Des fichiers externes aux fichiers de données contiennent de l’information comme du HTML ou bien du XML, dans notre cas. Ainsi, les Style Sheets Externes peuvent améliorer considérablement le travail en séparant l’information de la forme et du rendu visuel. De plus, les styles externes permettent une réutilisation pour plusieurs autres fichiers XML comme il est indiqué sur le schéma suivant.



Dans le cas où nous avons plusieurs feuilles de style, elles cascadent dans une seule suivant des règles et un ordre précis :

1- Styles par défaut du navigateur
2- Feuilles de Styles Externes
3- Feuilles de Styles Externes incluses dans la balise <head> pour HTML
4- Styles incluses dans des balises du corps du document HTML

(Voir quels styles présenter dans cette partie)

2. Structure de CSS et exemples.

Un fichier CSS est un fichier contenant les styles attribués aux différentes balises du document XML associé. Afin de lier une feuille de styles à une document XML, il est nécessaire d'ajouter la directive suivante au fichier XML :

< ?xml-stylesheet type = " text /css" href = "fichier.css">

Le fichier CSS se présente sous la forme d'une énumération d'étiquettes. Chacune de ces étiquettes est associée à une balise XML (on donne à l'étiquette le même nom qu'a la balise). Cette étiquette est suivie d'une série d'attributs décrivant le style.

Il est possible de choisir parmi plusieurs attributs ainsi que les différentes valeurs associées. Le détail des attributs et valeurs possibles est décrites dans les normes des différentes versions de CSS. On trouve par exemple, parmi ces attributs : la taille du texte, la justification, la fonte etc..

Exemple :

```
voiture {
        text-align : center;
        color: grey;
        font-family:courrier;
        font-size:25;
        font-weight:900}
```

Appliqué à note exemple de document XML représentant une voiture, cette feuille de styles nous donne le rendu suivant :

Renault Twingo Bic 1999

B. XSLT : Transformation de documents XML

XSL (« eXtensible Markup StyleSheets ») est un langage de présentation qui permet de transformer un fichier XML en une autre structure. On peut ainsi par exemple, à partir d'un fichier XML, d'une feuille de styles et d'une DTD, grâce à XSL, produire un document au format PDF ou encore un autre document XML.

Nous avons vu comment XML propose une abstraction entre les données et leur définition d'une part et leur forme de traitement d'une autre (par exemple leur mise en page). XSL est donc un élément indispensable à la chaîne de traitement des documents XML, assurant la dernière étape de médiation et de distribution des données.

XSLT est un des sous langages de XSL, il a pour but d'assurer la transformation des données de type XML et XPath. D'autres sous langages existent permettant de transformer des données d'autres types de XML.

1. XSLT en action

XSLT permet de transformer un document XML à partir d'une feuille de style qui contient les informations relatives aux modifications à apporter nommées « Template rules ». C'est plus précisément le processeur XSL qui va se charger d'effectuer la conversion.

Le processeur XSL manipule les données en les représentant sous la forme d'arbres, ainsi il en manipule principalement trois :

- Arbre correspondant au document XML source
- Arbre correspondant au document XML resultat
- Arbre qui contient les règles de transformation : les template rules

L'arbre résultat est construit par l'association des « patterns » issus de l'arbre source (correspondant aux nœuds et feuilles) avec les « templates » définis dans l'arbre de transformation. Chaque « template » se rapporte à un ou plusieurs « pattern » et définit les règles de transformation qui lui sont associées.

Un processeur XSL n'interagit pas avec les données. Il travaille sur la structure de document. Afin de pouvoir modifier le contenu du document, il fonctionne en coordination avec un parseur événementiel permettant d'altérer le contenu.

2. Structure des XSL Stylesheets

Une feuille de style XSL commence toujours par l'instruction ***<xsl:stylesheet>*** et se termine par ***</xsl:stylesheet>***. Cette balise dispose de plusieurs attributs comme :

- le numéro de version XSLT employé : *version ="1.0"*
- la référence vers le « namespace » qui régit le vocabulaire XSL : *xmlns:xsl="URI"*

Exemple :

```
<xsl:stylesheet
   [version="1.0"]
   [xmlns:xsl="http://www.w3.org/1999/XSL/Transform"]
   [xmlns:fo="http://www.w3.org/TR/WD-xsl/FO" result-ns="fo"]
>
...
</xsl:stylesheet>
```

A l'intérieur de l'élément <xsl:stylesheet>, on peut retrouver plusieurs balises dont certaines sont :

- xsl:import	Permet d'importer une feuille de style externe et modifie la structure arborescente de la feuille de style XSL appellante.
- xsl:include	Permet d'inclure une feuille de style externe sans modifier la structure de la feuille de style XSL appelante.
- *xsl:template*	Permet de définir des règles de présentation/dérivation pour chaque noeud ou élément du document XML original
- xsl:output	Permet de spécifier le format de sortie de l'arbre XML résultant de la transformation
- xsl:strip-space	Permet d'enlever les éléments d'un ou de plusieurs noeuds concernés pour lesquels il n'y a pas de valeurs.
- xsl:preserve-space	Permet de conserver les éléments d'un ou de plusieurs noeuds concernés pour lesquels il n'y a pas de valeurs.

On peut également retrouver des balises permettant d'effectuer des instructions algorithmiques:

- xsl:for-each	Permet de boucler sur un élément du document XML afin, par exemple d'en extraire toutes les valeurs sous-jacentes.
- xsl:if	Permet de construire des parties de l'arbre résultat de manière conditionnelle.
- xsl:choose	Permet de construire des parties de l'arbre résultat de manière conditionnelle.
- xsl:sort	Permet d'appliquer un tri sur les éléments de l'arbre original pour construire le résultat du traitement.

CONCLUSION

Nous avons présenté une introduction au langage XML en essayant de bien mettre en évidence la dissociation qui est faite entre structure et information. En effet, l'informatique a pour but le traitement de l'information et XML est une alternative puissante aux formes de stockage et de traitement qui existaient au préalable.

Les outils associés à XML sont nombreux. Nous nous sommes contentés ici d'en présenter quelques uns qui sont eux mêmes à la base d'outils plus complexes qui contribuent à la généralisation de l'usage de XML pour des applications nouvelles. Ainsi, l'émergence des WebServices a donné lieu à l'apparition de nombreux projets de développement afin de fournir une solution XML adaptée. On peut citer le projet « Jakarta » de « Apache » par exemple.

Le monde des réseaux est sans aucun doute le domaine où les applications de XML sont les plus nombreuses. Il est donc indispensable de le connaître afin d'appréhender les problèmes complexes de communication entre diverses applications qui de plus en plus s'effectuent à l'aide des outils associés à XML.