Cahiers GUTenberg

ÉTAT DES RECOMMANDATIONS XML DANS LE DOMAINE DOCUMENTAIRE

Pierre ATTAR, Bruno CHATEL

Cahiers GUTenberg, nº 37-38 (2000), p. 53-85.

<http://cahiers.gutenberg.eu.org/fitem?id=CG_2000___37-38_53_0>

État des recommandations XML dans le domaine documentaire

Pierre Attar[1] et Bruno Chatel[2]

1 TIRÈME : pattar@tireme.fr
2 bcha@chadocs.com

Note de la rédaction. *Cette étude, réalisée pour le compte de EDF, est parue sur le web*[1] *dans le cadre de Mutu-XML*[2]*, un projet de mutualisation de l'effort de montée en compétences sur XML. Elle est reproduite ici dans sa quasi intégrité*[3]*, avec l'aimable autorisation de Pierre Attar (pour le projet Mutu-XML), que nous tenons à remercier.*

Résumé.

Cette étude, réalisée pour le compte d'EDF, fait le point sur la recommandation XML et l'évolution de ses recommandations associées *du point de vue documentaire* ; ce point de vue si spécifique : où la fonction de l'auteur est primordiale, où le grand nombre d'auteurs nécessite des processus de collaboration importants, où enfin les documents peuvent aller du simple fragment à des arborescences profondes, pour des documents de plusieurs centaines de pages.
L'objectif de ce rapport est de faire le point sur la norme XML, ses forces et ses limites pour répondre aux besoins des applications documentaires. De plus, ce rapport s'intéresse aux *parsers* XML, pour en définir leur qualité et leur efficacité.

Mot-clés : voir le glossaire de l'Annexe 2.

1. http://www.mutu-xml.org/xml-base/articles/PUB-EDFSTD-FR.html dans sa version du 25 septembre 2000 : ce document devrait en toute logique évoluer en fonction de l'actualité ... dès lors que le projet aura réellement démarré, c'est à dire dès lors qu'il aura subventions et adhérents ...

2. « Mutualiser l'effort de montée en compétences sur XML » : http://www.mutu-xml.org/xml-base/annonces/PUB-ANN1-FR.html. Voir ci-après, Annexe 1.

3. Nous avons supprimé les résultats techniques un peu longs pour être imprimés ici (mais ils sont donc accessibles à l'url donnée en note 1) ainsi que diverses adresses et url (voir http://www.mutu-xml.org/xml-base/shared/COLLECTORS-FR.html). Par ailleurs, cette version papier (obtenue grâce à html2latex de Nathan Torkington) posait quelques problèmes dus à l'abondance des liens hypertextuels que nous avons résolus en regroupant toutes les url dans un glossaire unique, en Annexe 2.

1. Méthodologie

Les sources utilisées pour cette étude viennent uniquement du Web et d'Internet. Le W3C[4] est le site de référence pour toute l'information concernant les spécifications. Les listes de discussions et les forums thématiques, d'une part, et les sites consacrés à XML, d'autre part, sont l'information de référence pour comprendre les enjeux et débats.

Lire l'information sur XML n'est pas suffisant ; il faut aussi, en pratique, comprendre ce que cela recouvre. L'ensemble de ce rapport[5], ainsi que le glossaire l'accompagnant, ont par conséquent été réalisés en XML, en utilisant les outils suivants (voir Annexe 2) :

- Epic pour la saisie ;
- Xalan pour la création de l'information HTML et pour les programmes d'assurance qualité ;
- Balise pour tester la validité des liens externes ;
- XEP Rendering Engine pour le formatage papier ;
- XML Spy pour l'écriture des feuilles de styles XSLT ;
- Java API for XML Parsing, Xerces Java Parser, Microsoft XML Parser et Oracle XML Parser pour les tests des parsers.

Comme l'objectif de cette étude était de dégager un ensemble d'enjeux, le rapport lui-même s'intéresse seulement à ces enjeux fondamentaux. Pour le reste, l'explication des recommandations et leur état est donné en annexe, dans un glossaire à forte valeur ajoutée.

2. Spécificité d'XML pour le monde documentaire

XML est un langage de codage de données dont l'objectif est, dans un échange entre systèmes informatiques, de transférer, en même temps, des données et leurs structures. Ayant le pouvoir de coder n'importe quel type de donnée, depuis l'échange EDI* aux documents les plus complexes en passant par les échanges de données inter-applications, le potentiel d'XML est de devenir *le* standard universel et multilingue d'échange d'informations.

De façon historique, XML vient du monde documentaire, au travers de la norme SGML et, antérieurement, de GML. Ce monde documentaire se divise entre, d'une part, ceux qui voient un document au travers de son *enveloppe*, qui contient un ensemble d'index permettant de définir un contenu par ailleurs opaque, et, d'autre part, ceux qui s'intéressent à la *structure des contenus* eux-mêmes : les anglicistes parlent de *content management*.

4. Les sigles, acronymes et noms de produits sont définis à la fin de cet article, Annexe 2 page 83.

5. Il s'agit du rapport et des glossaires qui se trouvent sur le web à l'url donnée en note 1.

Si un document *n'est qu'*un ensemble d'index sur un document par ailleurs stocké, il existe peu de spécificité pour le monde documentaire. Ceci est tellement vrai que l'on voit apparaître des simplifications de tous les langages de descriptions de documents de base, pour les confondre avec ceux, plus génériques, concernant les métadonnées.

En revanche, les spécificités des documents structurés selon leurs contenus sont importantes. Tout d'abord, elles concernent la taille des documents eux-mêmes. À titre d'exemple, on se rappellera qu'un manuel de maintenance aéronautique « pèse » quelques centaines de méga-octets ... de quoi dépasser la capacité mémoire de beaucoup d'ordinateurs, dès lors que des traitements doivent être réalisés sur tout le document pour, par exemple, générer une table d'index.

Une autre spécificité notoire tient aux modes de gestion de cette information. Que ce soit un éditeur juridique ou un industriel de l'Après-Vente, à partir du moment où des efforts de structuration logique de l'information documentaire sont réalisés, il devient utile de faire porter à ces éléments de structuration des notions de configuration d'information, de gestion de processus éditorial, voire n'importe quel autre processus de *knowledge management*.

Faire porter de l'information de gestion au document permet de le contrôler au mieux et *au plus près de* l'information à gérer. Par exemple, il n'est pas rare de voir, directement dans un document, un appel à une image ayant par ailleurs un statut de type « image commandée ». Un processus automatique se chargera alors de réaliser la commande à un « service image » et d'en gérer la production.

La spécificité des documents structurés selon leur contenu est alors que le document lui-même devient réservoir d'informations. Cette information permet, certes, d'en assurer son formatage multisupports, mais aussi de déclencher n'importe quel autre processus de gestion, de qualité, etc.

Enfin, la spécificité des documents structurés selon leurs contenus tient surtout à l'*étroite inter-relation existant entre le texte et sa structure*. Dans une documentation technique automobile, on peut voir des phrases de type :

> « ... utiliser l'outil <referenceOutil num="234"> pour démonter le joint. Au remontage, appliquer la graisse <ingredient nom="molykote"> sur le joint <piece part="PR456"> et serrer au couple de <couple ref="JOINT-REMONT">... »

L'intérêt ? Dans une interface de consultation dynamique, sur le Web, et en fonction de la configuration de la voiture que l'on souhaite réparer, on pourra présenter à l'utilisateur la graisse, la pièce de rechange et le couple de serrage appropriés. La conséquence ? Le travail du rédacteur devient plus conceptuel : il passe d'une écriture explicite et linéaire d'un document à la création de fragments autonomes d'informations documentaires, où la notion de document linéaire est construite *a fortiori*, lors de la création d'une publication, sur un support donné. Les environnements d'édition des rédacteurs et concernant les processus de gestion des contenus documentaires deviennent alors le facteur clé du succès des applications XML ... toute chose qui n'existe pas, par ailleurs, dans d'autres utilisations de cette même recommandation.

La complexité du poste de rédaction et du travail d'auteur fait encore peur aujourd'hui ! On a pensé que XML simplifierait la complexité de SGML ; on a simplement oublié que, ici, la complexité n'est pas technique, elle est liée à une évolution de métier : de la rédaction à la mise en place de fonds documentaires de connaissance. On a aussi oublié que le travail d'un auteur se complexifie, dès lors qu'il doit, en même temps, penser à ce qu'il rédige et penser à comment structurer et organiser ce qu'il rédige... Car il existe, d'un point de vue opératoire, une différence fondamentale entre, d'une part, écrire « serrer la vis à un couple de 4 DaN » et, d'autre part, écrire « serrer la vis » puis changer d'outil d'*édition,* pour aller chercher dans une base de données de faits techniques le bon couple de serrage approprié à la configuration de l'information en cours rédaction et le coller dans le document en cours d'écriture.

Si la complexité est avérée, c'est cependant dans ce type d'applications que beaucoup voient la richesse du monde documentaire structuré selon ses contenus : elle permet d'adapter les usages, pour ne présenter que ce qui est pertinent et efficace, en fonction de configurations ou de profils. Elle permet aussi d'atteindre des objectifs d'assurance qualité impossible à atteindre avec des méthodes de rédaction traditionnelles ; elle permet, enfin, la parallélisation des tâches, à un moment où l'on parle de plus en plus d'outils de travail en groupe.

Si le travail de l'auteur est aussi important, la spécificité du monde documentaire va obliger à s'intéresser aux notions de modèles et de conformité des documents aux modèles. En effet, comment proposer à un rédacteur un environnement de saisie sans l'aider et contrôler comment il saisit. Comment, par ailleurs, pouvoir assurer des rédactions collégiales, entre différents partenaires, sans être capable de contrôler et valider que ce qui est échangé correspond à ce qui est attendu.

3. XML

La spécification XML est maintenant votée depuis février 1998, soit bientôt 3 ans. Depuis, peu de demandes de modifications ont été nécessaires et les seuls changements en cours, pour la nouvelle version sont éditoriaux. XML est donc une recommandation stable, avec un intérêt croissant de la part de la communauté du Web, mais aussi de beaucoup d'autres secteurs informatiques s'intéressant aux notions d'échanges d'informations entre applications. Elle est aussi candidate au remplacement de SGML, la norme qui structure d'ores et déjà des fonds documentaires très importants.

3.1. Apports de XML

L'objectif de cette étude n'est pas de justifier l'intérêt de XML en tant que tel, mais beaucoup plus de comprendre ce qu'il apporte aux documents structurés, entre autres, par rapport à SGML, depuis longtemps utilisé dans ce domaine.

D'un point de vue très technique, XML met en place la différence entre les notions de modèles de données et celles de langage de codage de données. Autant, avec SGML,

le modèle (la DTD*) définissait *aussi* le système d'encodage, autant, avec XML, ces deux notions sont bien séparées. Elles sont tellement bien séparées qu'il est alors possible d'avoir des langages de définition de modèles alternatifs, comme le sont les schema*. Cette avancée est fondamentale, car elle permet à tous les outils de ne s'intéresser aux modèles qu'au moment où cela est réellement nécessaire, pour des activités de validation.

Pour aller plus loin dans cette démarche, si le W3C définit XML (le langage de codage d'un *échange* entre deux applications), il définit aussi ce que l'on doit *reconnaître* dans un document XML, d'un point de vue applicatif : c'est l'objectif des *Infoset*. Du coup, on est, certes, capable d'échanger, selon un format neutre et indépendant des systèmes et des applications, mais on est, de plus, capable d'obtenir une vision commune de ce qui est échangé : ce qui est important et ce qui ne l'est pas.

D'un point de vue plus francophone ou multinational, l'apport principal d'XML est le fait que le langage repose sur *Unicode XML*. Il devient alors possible d'échanger de l'information sans se soucier de la façon dont chaque plateforme matérielle et logicielle interprétera les caractères. *Unicode XML* définissant un jeu de caractères extrêmement large, il ne nécessite plus de se reposer sur les jeux de polices pour modifier des significations de caractères, comme cela pouvait être réalisé avec l'ASCII.

D'un point de vue opérationnel, enfin, c'est la kyrielle de recommandations associées qui font prendre à XML toute sa valeur, surtout les recommandations liées à l'exploitation de documents XML. Même si peu de concepts nouveaux sont introduits par rapport à SGML, la force d'XML est liée au fait que ses recommandations associées sont implémentées ou en cours d'implémentation. Du coup, utiliser de l'information XML devient aisé, vu la qualité et la diversité de l'offre logicielle existante.

3.2. Limites

La première limite, rapidement identifiable, est que toutes les recommandations du W3C sont liées à l'utilisation de documents XML sur *Internet*. Certes, c'est là le champ d'action du W3C, mais on ne peut que regretter le fait que d'autres regroupements ou organismes de normalisation plus ouverts (comme l'ISO) ne s'approprient pas plus rapidement ces textes, afin de les rendre utilisables dans n'importe quel environnement applicatif.

En effet, même si Internet prend de plus en plus de place dans les environnements informatiques, il n'est pas raisonnable, à ce jour, de considérer que c'est la seule plateforme réseau utilisée et, *a fortiori*, pas non plus la seule plateforme système. Du coup, la question est : est-ce possible d'utiliser XML comme étant le format sur lequel se reposent les bases de données internes et leurs applicatifs, dans le but, certes d'alimenter un site Web, mais aussi de réaliser beaucoup d'autres traitements, qui ne soient pas spécifiquement liés à un environnement de réseau ? Pour XML lui-même, cela ne semble poser que peu de problèmes, car c'est, par définition, un format d'échange neutre. Pour les recommandations associées, il faut bien trier ce qui relève de la seule utilisation Internet de ce qui peut être généralisé à n'importe quelle architecture informatique.

La seconde limite est liée au champ couvert par la recommandation, qui propose *un langage de codage de documents*, un langage de *définition de modèles documentaires*, un langage d'expression d'*inclusions de fichiers* et des *fonctionnalités applicatives* comme, par exemple, la notion de langue de rédaction (xml:lang).

D'une part, la double fonction des DTD (définir des modèles et permettre d'exprimer des inclusions) soulève beaucoup de questions quant à l'avenir du mécanisme des inclusions par entités générales définies dans des DTD, à un moment où beaucoup pensent au remplacement de l'expression des modèles de document selon des DTD par des schema. Il serait alors dommageable que ce remplacement puisse faire « perdre » les d'entités générales servant à l'inclusion.

D'autre part, même si l'intention, consistant, par exemple, à intégrer au plus bas niveau d'XML des notions de langues est bonne, XML perd alors sa grande force, qui est d'être un langage d'échange *neutre* au regard des applications : la langue est une information hautement applicative et des traitements sur cette valeur sont directement définis dans la recommandation. Que faire alors quand une entreprise ou un groupement de professionnels définissent eux-mêmes leurs propres notions de langage ?

De façon plus générale, il serait important de différencier les différentes parties de cette recommandation pour bien comprendre de quoi relève chaque partie. Cela nécessiterait, certes, de créer des recommandations différentes, mais surtout de commencer à construire des architectures d'utilisation de recommandations, toute chose que le W3C ne semble pas vouloir faire.

D'un point de vue technique, le grand reproche que l'on pourrait aujourd'hui faire à la recommandation XML est lié à la localisation de fichiers. En effet, toutes les entités externes définies dans des DTDXML nécessitent de différencier, d'une part, et si nécessaire, un nom public d'entité, mais, d'autre part, une localisation dans un *système de fichier*. La localisation est alors codée définitivement dans les modèles et il devient impossible de réaliser des implémentations différentes, sur des sites différents.

Par exemple, si une DTD référence un fichier qui doit être inclus dans un document, le mécanisme est connu et s'écrit comme suit :

```
<!DOCTYPE essai [
    ...
     <!ENTITY inclusion PUBLIC "-//MUTU-XML//Fichier Appelable//EN"
     "t:\mutu-xml\tmp\fichier.xml">
      ...
]>
 <essais>
  &inclusion;
 </essais>
```

Toute la question est alors liée à l'adresse physique t:\mutu-xml\tmp\fichier.xml. En effet, c'est un fichier sur un disque qui est ici proposé. Sur un ordinateur portable, peut être que ce disque n'existera pas... Dans tous les cas, cette adresse ne fonctionnera pas sur

Internet. Certes, on peut se rattacher à des adresses relatives, mais ce mécanisme nécessite alors des organisations de sites et *de systèmes de fichier,* qui soient toutes identiques. Pourtant, ce que l'on veut dire est relativement simple : il existe *quelque part* un objet dont le nom logique est -//MUTU-XML//FichierAppelable//EN ; charge aux applicatifs de le trouver physiquement.

Le problème n'est pas nouveau ! Après bien des errements, la communauté SGML s'était finalement mise d'accord, sous l'égide du consortium *OASIS Open* sur une résolution *Entity Management, OASIS Technical Resolution TR9401* qui propose de créer un *catalogue*, partageable par un ensemble d'applications permettant la résolution de noms publics en des chemins d'accès à des objets physiques.

Avec ce système, un catalogue par site de production ou d'utilisation est suffisant, toutes les applications partageant cette table de traduction. Les catalogues eux-mêmes sont de la forme :

```
PUBLIC "-//MUTU-XML//Fichier Appelable//EN" "t:\mutu-xml\tmp\fichier.xml"
```

Dans le domaine XML, ce type de recommandation n'existe pas (voir *Standard Deviations from Norm - If You Can Name It, You Can Claim It!*). Pourtant, beaucoup ressentent ceci comme une difficulté réelle, dès lors que l'on s'intéresse à la portabilité de documents et d'applications sur des sites distincts.

C'est ainsi qu'*Arbortext* propose, sur son propre site, un ensemble de classes Java, implémentant les recommandations de *OASIS Open* quant à la gestion de catalogues... C'est une avancée certaine, en attendant une prise en compte de cette recommandation par le W3C. Par ailleurs, la question est enfin formellement identifiée par *OASIS Open* qui vient de créer un comité technique sur le sujet http://www.oasis-open.org/committees/entity/index.shtml.

Pour conclure sur ce point, il faut noter que, si les DTD doivent disparaître, la question soulevée serait encore plus gênante, car les schema, supposés être les remplaçants des DTD, ne comportent pas de notion de ce type : les schema reposent sur les *Infoset*, qui s'appuient sur la représentation de l'information disponible dans un document XML, *une fois les entités et espaces de noms résolus*.

Enfin, la dernière limite, non négligeable, d'XML, du point de vue des intégrateurs de systèmes et des applications développées en environnement ouvert utilisant différents outils logiciels, est liée à la kyrielle de recommandations associées et à la difficulté à comprendre comment tout cela coopère. Ce point est plus particulièrement développé dans la section suivante.

4. Architecture des recommandations liées à XML

Recenser toutes les recommandations liées à XML ou utilisant XML, issues du W3C ou d'autres associations, donne parfois des vertiges ! Elles sont toutes proposées, sous

forme de liste extrêmement linéaire où il faut ensuite explorer soi-même, pour comprendre de quoi relève une recommandation particulière et quel est son contexte d'utilisation.

Pourtant, si XML est un standard de description de données, il faut, pour l'utiliser, lui adjoindre ses spécifications d'utilisation, qui lui donnent toute sa valeur ; c'est ce dont est en train de se rendre compte la communauté de développeurs XML. En effet, une série de messages (*"The failure to communicate XML - and its costs to e-business"*) est apparue sur les listes de diffusion XML dont l'intérêt est finalement d'expliquer que certes XML est simple mais qu'il n'apporte pas grand chose, d'un point de vue implémentation, en tant que tel : ce sont les standardisations de méthodes de traitement et surtout les vocabulaires qui lui font prendre toute sa valeur ... forcément au prix d'une complexité accrue.

4.1. Généralités

Plusieurs solutions sont utilisées pour mettre en place les spécifications léies à XML :

- *Définitions de valeurs noms d'attributs ou d'éléments : les vocabulaires*
  Avec cette méthode, utiliser des recommandations revient, dans le codage des documents XML eux-mêmes, à ajouter des éléments et/ou des attributs dont le nom et la signification sont par ailleurs définis. L'utilisation des recommandations revient à introduire des modèles de modules d'informations (éléments et attributs) définis dans un espace particulier.
  Les *NameSpace* sont alors utilisés, pour être sûr de ne pas provoquer de « collisions » entre les éléments ajoutés et ceux du modèle propriétaire. Les logiciels applicatifs traitant les documents peuvent alors reconnaître ces objets typés et les utiliser pour ce qu'ils représentent. Sont définis dans cette catégorie :
  - *des uniformisations de base*, communes à toutes les applications XMLBase, XMLLanguage, XHTML, XPointer, XLink, etc.),
  - *des uniformisations horizontales*, permettant d'envisager des objets contenus dans des documents composites (SVG, OASISTables, MathML, etc.),
  - *toutes les uniformisations sectorielles*, qu'elles soient horizontales (CML, RDF, etc.) ou verticales (EAD, DocBook, NewsML, XHTML, ebXML, BizTalk, etc.),
  - etc. ;
- *Uniformisation de méthodes de représentation d'un fichier* XML (Canonical, Infoset, etc., mais aussi XSLT et XPath, qui ont leur propre modèle de données) ;
- *Uniformisation des méthodes d'accès à un objet contenu dans un document* XML (XPath, XMLQuery, etc.) ;
- *Uniformisation des langages de spécification de programmes manipulant des données* XML (DOM, XSLT, SAX, XSL, etc.).

Un si grand nombre de spécifications nécessitent de se définir une typologie permettant de classer celles-ci. Il est intéressant de regarder, sur les sites parlant d'XML, les propositions réalisées.

Par exemple : le site du W3C, organisé par domaines, proposait les domaines suivants en début d'année 2000 : interface utilisateur (*User Interface domain*), architecture (*Architecture domain*), technologie et société (*Technologies and Society domain*)... une typologie somme toute relativement pauvre. Aujourd'hui, plutôt que d'étoffer leurs nombres de domaines, ils se restreignent, sur leur page d'accueil, à ne plus mettre en avant cette organisation, pour ne plus proposer qu'une simple liste de recommandations, sous forme de liste ordonnée d'acronymes.

Par ailleurs, Jeanne El Andaloussi, lors de la conférence XML Europe 2000, proposait une classification distinguant les standards génériques des standards industriels. Elle différenciait aussi les standards génériques en fonction de leur phase d'utilisation dans un projet. Les catégories étaient alors les suivantes : modélisation, implémentation, publication, plus une catégorie liée à la seule syntaxe d'XML. Un effort était enfin réalisé ! Cette typologie a un avantage opérationnel certain : elle repose sur un acquis commun qui est celui de l'organisation de projet.

La première critique, à y regarder *de plus près*, est liée au recouvrement des catégories de la typologie. Par exemple, la catégorie syntaxe recouvre souvent toutes les autres catégories. De même, et comme l'explique l'auteur, une recommandation particulière peut apparaître dans différentes phases d'un projet : ce sont, par exemple, les schema, qui servent aussi bien à la modélisation qu'à l'implémentation. Enfin, cette vision catégorise trop les recommandations en fonction de leur utilisation dans un projet.

4.2. Proposition de typologie

Avec une typologie, il devient possible de comprendre à quoi sert une recommandation et de quel type elle relève. Il devient aussi possible de comprendre quelles sont les recommandations concourantes sur un même sujet et comment les positionner entre elles.

La typologie ici utilisée est basée sur des objectifs partagés par un ensemble de spécifications. Par ailleurs, elle différencie tout ce qui relève des vocabulaires applicatifs de ce qui est lié à des notions de traitement, voire à une boîte à outils de base.

Elle nécessite parfois, pour atteindre ses objectifs, de « découper » différentes notions, de nature différentes, coexistant dans une même recommandation, : par exemple, dans la recommandation XML, cette typologie différencie le format de codage XML, les DTD et les notions de XMLLanguage.

4.2.1. Standards de base

Il s'agit surtout des recommandations XML et UnicodeXML ; SGML en fait partie, comme norme de référence au sein de l'ISO.

4.2.2. Définition de modèles

Il s'agit des recommandations permettant de définir des modèles électroniques d'informations ; elles sont définies sous forme de métalangage.

On y trouve surtout les DTD d'XML et les schema. Les *NameSpace* font partie de cette classe, en ce sens qu'ils assurent la coopération de modèles. Il faudrait certainement, en toute logique, y adjoindre RDF, en tant que spécification de modélisation de Métadonnées.

4.2.3. Représentation et fragmentation

canonical et Infoset s'intéressent à la représentation de l'information XML : l'un, sous forme de *fichier* canonique, l'autre, sous forme d'*arbre d'objets typés*.

XInclude et FragmentInterchange s'intéressent à la fragmentation logique de documents XML : l'un propose de définir une sémantique d'inclusion différente de celle proposée par le mécanisme d'entités, l'autre s'intéresse à l'information de contexte qui doit être échangée avec un fragment de document XML.

4.2.4. Utiliser des documents XML

Une série importante de recommandations adresse les besoins génériques d'utilisation de documents XML. Les impératifs d'utilisation peuvent être définis lorsque l'on construit un document XML ou lorsque l'on exploite un document XML.

- *Construire* Plus la construction utilisera des sémantiques « universellement » reconnues (par exemple, la sémantique de *position* définie par XPointer), plus l'exploitation en sera aisée, en ce sens qu'elle se reposera sur des classes d'objets de plus haut niveau que les classes de documents conformes à un modèle.
  Il faut distinguer, dans cette catégorie, les recommandations permettant d'identifier des contenus de documents XML (XPath, XPointer) et celles permettant d'accéder aux contenus des documents XML (XMLQuery, XLink).
  Par souci de cohérence, il serait certainement intéressant d'ajouter à cette catégorie la recommandation sur les types de données, issue des schema.
- *Exploiter* L'exploitation de documents XML différencie les recommandations liées aux traitements (XSLT, DOM, SAX) de celles basées sur des méthodes génériques de présentation (XSL).

4.2.5. Domaines d'utilisation

La catégorie des *Domaines d'utilisation*, des vocabulaires, est celle qui va le plus évoluer : c'est celle qui permet de passer d'un langage universel et neutre à un langage applicatif, en définissant des types d'objets (qu'il s'agisse d'attributs ou d'éléments, avec leurs modèles de contenus).

Trier cette catégorie relève d'une gageure et cette partie de la typologie devra souvent être réactualisée.

Plus précisément, il est intéressant de dissocier les vocabulaires applicatifs horizontaux de ceux s'adressant à un marché beaucoup plus vertical. Les vocabulaires applicatifs horizontaux liés au Web appartiennent à cette catégorie. Ceci permet de comprendre pourquoi le champ couvert par les spécifications du W3C est beaucoup plus large que son seul objectif, qui est de recommander des standards d'utilisations du Web.

- *Vocabulaires horizontaux* Il s'agit des vocabulaires que l'on peut espérer voir partagés par toutes les implémentations d'outils utilisant XML. On y distingue les graphiques vectoriels du Web (SVG) et le modèle de tableaux d'OASISOpen, etc.
  Est-ce que ces vocabulaires de base doivent aussi prendre en compte les mathématiques, avec MathML, voire les formules chimiques, avec CML ? En toute logique, oui pour le premier cas, du fait de la communauté importante de techniciens, chercheurs et autres scientifiques utilisant ces notations typographiques. Dans le second cas (CML), la typographie liée à l'expression de la chimie relève plutôt d'applications très sectorielles.
- *Vocabulaires liés à Internet* Il s'agit de tous les vocabulaires liés à l'évolution d'Internet, avec :
  - les **vocabulaires de base**, incluant, d'une part, la mise en place des codifications de langues et de pays (IETFLANG), et leur utilisation dans les documents XML (XMLLanguage) ; d'autre part, les notions d'*URL* de base (XMLBase).
  - les **vocabulaires liés au Web** : incluant le langage lui-même (HTML) et sa représentation sous forme XML (XHTML) ou sa transposition pour le WAP (WML) ; incluant aussi les formulaires (xforms) et tout ce qui permet d'utiliser des *Feuilles de styles*.
- *Domaines verticaux* Il s'agit de tous les vocabulaires sectoriels. Certains sont donnés ici à titre d'exemple, le souci d'exhaustivité ne pouvant être réalisé que dans le cadre d'une activité de veille technologique dynamique et quotidienne.
  - *Métadonnées* : HyTime, RDF, TopicMaps, etc.
  - *Multimédia* : SMIL, etc.
  - *Documents* : EAD, NewsML, DocBook, BiblioML, DublinCore, TEI, etc.
  - *Commerce électronique* : ebXML, BizTalk, etc.

4.3. Architecturer des recommandations

La notion de typologie ici définie permet d'organiser les recommandations entre elles et de définir à quel type d'objectifs répond une recommandation. Elle ne résout cependant pas les questions d'architectures applicatives sous-tendues par l'utilisation de celles-ci. Aujourd'hui, réaliser des applications XML relève du supermarché d'articles de bricolage pour construire sa maison : tous les rayons existent, pour tous les besoins ! La seule question est de savoir ce qu'il faut prendre et comment utiliser un matériau avec un autre.

En effet, si beaucoup de recommandations existent, répondant à bon nombres de besoins, peu d'efforts sont réalisés pour définir comment elles s'architecturent entre elles pour mieux coopérer. Par exemple, si Infoset est défini, beaucoup de débats existent sur le fait que ce modèle soit concurrent de celui utilisé par XPath ; les recommandations, quant à elles, ne parlent que de compatibilités.

Pour fournir un nouvel exemple, XMLBase et XInclude semblent être d'excellentes recommandations pour faciliter la gestion de site Web. En environnement ouvert, où les traitements sont réalisés sur les données, sur le serveur ou encore sur le poste client, la question est de savoir si ces recommandations sont utilisables en l'absence d'une définition de leur champ applicatif. Qui doit l'implémenter ?... Les logiciels de consultation du Web ? Les *parsers* ? Les outils de manipulation de document ?

L'enjeu est de taille, car une bonne partie des recommandations nécessite de coder des objets typés standardisés *à l'intérieur des documents eux-mêmes*. Du coup, ce n'est pas seulement le Web qui doit reconnaître ces informations, mais bien l'ensemble des chaînes applicatives. Par exemple, on peut espérer pouvoir utiliser XLink et XPointer sur nos futurs logiciels de navigation Internet. Faut-il tout de suite le faire ? Cette réflexion doit être liée à l'ensemble des autres outils de traitement de l'information et également au fait de savoir comment ces outils implémentent ces recommandations ou, s'ils ne les implémentent pas, à la nature de l'effort de développement à réaliser pour ce faire.

Certes, il semble peu raisonnable de vouloir architecturer toutes les recommandations du W3C et celles d'autres organisations de normalisation. Il semble, pour le moins, possible de définir des classes de recommandations de base (augmentation du pouvoir de représentation d'XML, HTML, etc.) et de les architecturer entre elles dans des notions de profil d'implémentation, comme cela est fait au sein de l'ISO avec les « profils d'application » des standards CGM ou, il y a bien longtemps, ODA.

DOM montre une autre voie possible : une bonne partie du cœur de la recommandation s'intéresse aux notions d'*implémentation*, définissant des méthodes permettant de savoir ce qui est implémenté par un outil logiciel. Cela n'est pas suffisant, car tout se passe, dans ce cas, au niveau des applicatifs et non pas au niveau des données elles-mêmes.

Avec les *profils d'application* de l'ISO, le document définit lui-même de quel niveau du standard il relève : le niveau étant par ailleurs normalisé. Les applicatifs savent alors directement comment traiter l'information contenue. En allant plus loin, Interpress de Xerox proposait, il y a déjà quelques dizaines d'années, une méthode de description de documents permettant à des logiciels d'*internaliser* uniquement ce qu'ils savaient traiter, pour, après traitement, *externaliser* l'information modifiée, au bon endroit, dans le document préalablement internalisé.

Ce sont toutes ces réflexions qui manquent au sein du W3C, pour permettre à XML d'atteindre ses véritables enjeux : devenir l'ASCII du millénaire et pas seulement l'ASCII du Web.

5. Construire une application XML

Utiliser XML ne relève pas seulement de l'utilisation d'un *éditeur* et d'un *formateur*. Si XML définit des structures d'informations permettant à des processus informatiques de s'appuyer sur les sémantiques sous-jacentes pour effectuer des traitements automatisés, la notion d'application XML prend alors toute sa valeur.

Cette section examine tout d'abord l'état des réflexions sur les modèles documentaires, pour ensuite définir l'état de maturité des outils eux-mêmes.

5.1. Notions de modèles de document

En général, dans la sphère XML, la plupart des spécifications sont liées à des documents bien formés, sans notion de modèle de document. La notion de modèle semble peu intéressante, car, finalement, les documents seraient souvent issus de *process* applicatifs, par exemple, générés par une base de données.

Dans le monde documentaire, les documents XML créés sont réalisés par des auteurs, au travers d'*éditeurs* structurés qui s'auto-adaptent à des DTD, à des modèles de document. Demain, ces modèles seront peut-être formalisés sous forme de schema, mais la notion de modèle persistera : d'une part, pour valider la saisie et, d'autre part, pour permettre de s'appuyer sur une spécification électronique destinée à utiliser des outils d'aide à la saisie de structure.

De façon plus générale, pour l'échange inter-applications entre partenaires, il est d'expérience souvent difficile, pour toutes sortes de raisons techniques ou organisationnelles, de présupposer que l'émetteur de l'information envoie des documents conformes à ce qui était attendu.

La validation permet alors de s'assurer que le document est conforme à un modèle commun et partagé. Si un document est conforme, cela veut dire que les programmes applicatifs « savent » de quoi sera composé le document... ces programmes sont alors plus aisés à écrire, car ils ne sont pas obligés d'implémenter tous les tests de conformité de l'information à ce qui est présupposé attendu.

Cette section s'intéresse d'abord au positionnements relatifs des DTD et des schema. Elle analyse ensuite les problèmes posés par la non utilisation des *NameSpace* dans les DTD.

5.1.1. DTD *versus* Schema

Beaucoup d'acteurs du W3C et de la communauté XML ne reconnaissent pas la notion de DTD. Ils lui préfèrent la notion de schema, davantage à même d'obtenir de *typer des données*, de *prendre en compte les espaces de noms (NameSpace)* et de *définir des contraintes* dans les modèles.

Cependant, les schema ne permettent pas, comme dans les DTD, de déclarer des entités qui définissent des fragments XML ensuite utilisables dans les instances. Par exemple :

```
<!DOCTYPE XMLFR PUBLIC "-//XML-FR//DTD Data Base//EN" [
   <!-- Définition du nom logique et physique du fragment -->
   <ENTITY infos PUBLIC "-//XML-FR//Informations Database//EN"
   "http://mesInfos.xml">
]>
<!-- Utilisation du nom logique -->
&infos;
```

Du coup, autant, grâce à une DTD, il est possible de faire partager des fragments d'informations à un ensemble de documents. Avec les schema, il sera nécessaire de reporter ces définitions dans la déclaration locale à chaque document (l'*internal subset) :* les *parsers* d'arbres bien formés ne sont pas supposés lire les jeux d'entités externes.

En toute logique, les DTD pourraient être éliminées au profit des schema. Cette élimination sera certainement très progressive : dès lors que la recommandation sera finalisée et acceptée ; dès lors que tous les outils basés sur des DTD seront capables de prendre en compte, à la place, les schema ; dès lors, enfin, que toutes les DTD existantes seront modifiées, pour être compatibles avec les schema. Il restera alors comme seul rôle aux DTD la définition des jeux d'entités.

Par ailleurs, si les schema ne s'intéressent qu'à un arbre d'objets typés, tout ce qui est de l'ordre de la fourniture de valeurs d'attributs par défaut, utilisées avec les DTD, ne peut pas fonctionner, car c'est le rôle du *parservalidant*. En effet, autant un *parser* (lisant un document et sa DTD) fournit un jeu d'Infoset complet résolvant les notions de valeurs par défaut, autant les schema, basés sur les Infoset, ne peuvent déduire cette information, le *parsing* étant déjà réalisé. Ce sont alors des *post-process* applicatifs qui doivent être mis en place.

Enfin, les DTD permettaient, *au travers de la déclaration du document*, de définir quels objets étaient susceptibles d'être échangés entre deux partenaires. Cette notion n'existe plus avec les schema et devra être reconstruite dans les applications traitant les informations reçues.

5.1.2. Modèles documentaires et espaces de noms

Les espaces de noms *(NameSpace)* se définissent dans les documents XML eux-mêmes, car toute la solution repose sur le fait que l'on doit pouvoir les utiliser dans des documents « bien formés ». Cependant, si la spécificité du monde documentaire est de se reposer sur des modèles, il est alors intéressant de comprendre comment les espaces de noms sont utilisables dans les recommandations ayant trait aux modèles documentaires. En utilisant les espaces de noms dans les modèles, il serait possible de demander à un *éditeur* de générer automatiquement toute l'information de nommage nécessaire chaque fois qu'un espace de noms est utilisé. Ce même *éditeur* pourrait aussi prendre en charge, les notions de préfixage d'objets.

Par ailleurs, ill serait aussi possible de valider la conformité d'un document à son modèle, en utilisant les modèles liés aux espaces de noms.

Dans les schema, l'attribut xsi:schemaLocation définit où se trouve le modèle défini sous forme de schema lié à un espace de noms ; il est alors possible de valider complètement un document. Il n'en va pas de même avec les DTD, car la recommandation XML ne prend pas en compte les notions d'espaces de noms, si ce n'est pour expliquer qu'il vaut mieux ne pas utiliser le caractère « : » dans les noms d'éléments et d'attributs.

> *Note: The colon character within XML names is reserved for experimentation with name spaces. ... In practice, this means that authors should not use the colon in XML names except as part of name-space experiments, but that XML processors should accept the colon as a name character.*
> Extrait de *Extensible Markup Language (Second Edition) (Version 1.0)*

Faut-il être plus explicite ?

> *It would of course be very useful to have namespace-aware validation: to be able to associate each URI used in a universal name with some sort of schema (similar to a DTD) and be able to validate a document using multiple such URIs with respect to the schemas for all of the URIs. The XML Namespaces Recommendation does not provide this. The reason is is that DTDs have many other problems and missing features in addition to lack of namespace-awareness. So the plan is to come up with a new schema mechanism that fixes the problems with DTDs and, as part of this, provides namespace-awareness. This work is being done by the XML Schema WG.* Extrait de James Clark, *XMLNameSpace*

Du coup, et pendant toute la période où les DTD continueront à exister, il est nécessaire de trouver une ingénierie pour intégrer aux DTD des spécifications issues d'espaces de noms externes. Il est plus généralement nécessaire de trouver une ingénierie permettant d'intégrer à un modèle, des modèles liés à des espaces de noms réalisés tant avec des schema qu'avec des DTD.

Sur le premier sujet, différentes solutions sont proposées sur les listes de débats (liste sur *Does DTD validation work with namespaces?*). Par exemple, pour faire porter aux documents des attributs de déclaration d'espaces de noms, de façon automatique, il suffit d'utiliser les attributs déclarés « FIXED » pour automatiquement générer la bonne information.

Plus précisément, intégrer des espaces de noms en utilisant des jeux d'entités externes n'est pas trivial et il faut mettre en place un minimum d'ingénierie. Les questions soulevées sont différentes selon qu'il s'agit d'intégrer des attributs ou des éléments :

- intégrer des attributs
  La seule solution est d'appeler sur les bons éléments de sa propre DTD propriétaire, les déclarations d'attributs externes.
  Pour bénéficier de la spécification externe et de ses évolutions, il est alors nécessaire que toutes les déclarations d'attributs soient déclarées comme des entités paramètres dans la spécification externe.
- intégrer des éléments
  À supposer que la spécification liée à un espace de noms existe et qu'elle soit

une DTD valide, il n'y a pas trop de problèmes, si on intègre un modèle d'objet composite complètement autonome.
Si le jeu externe doit lui-même utiliser une partie de la spécification (DTD) propriétaire appelante, l'intégration sera plus complexe à réaliser ; ce sera le même problème avec les schema ou avec les DTD. Par exemple, supposons le cas bien connu des tableaux où il est nécessaire d'utiliser un modèle externe (celui de *OASIS Tables*) ; on veut, dans les cellules, pouvoir faire appel à ses propres objets. Il est alors nécessaire d'intervenir, depuis l'extérieur, sur la spécification des tableaux. Quand cette spécification est correctement conçue, comme c'est le cas pour *OASIS Tables*, pas de problèmes : il suffit de redéfinir une entité paramètre de contenu avant de réaliser l'appel au modèle externe ; sinon, on est obligé de dupliquer, en local, le jeu externe, afin de le modifier.
Le problème d'écriture de DTD se complexifie encore, lorsqu'on travaille avec des modèles où il y a une inter-relation étroite entre les deux vocabulaires, comme c'est le cas par exemple pour XSLT et *XHTML* lors de l'écriture d'un programme de transformation de XML vers HTML. Ecrire la DTD est une opération non triviale, voire même impossible du fait des relations de dépendance entre jeux d'entités paramètres qui ne sont pas exprimables. Il faut alors définitivement passer au schema, qui semble permettre d'exprimer ce type de contraintes.

De façon plus générale, si certains espaces de noms utilisent des schema, tandis que d'autres utilisent des DTD, il faut comprendre les contraintes sur les outils de validation, qu'il s'agisse des *parsers* ou des *éditeurs* interactifs, afin qu'ils puissent accomplir leurs rôles. Le W3C ne répond pas à la question, les DTD semblant par trop dépréciées.

En conclusion, une position semble-t-il raisonnable serait d'intégrer aux DTD un minimum de support des espaces de noms, pour, au moins, pouvoir utiliser des documents composites contenant des fragments autonomes issus de différents modèles. (Par exemple, une documentation technique contenant des tableaux OASISTables, des équations MathML et des graphiques SVG.) Toute la question sera alors de comprendre jusqu'où devrait aller la spécification : il n'est pas rare de rencontrer, dans une documentation technique, un tableau contenant, d'une part, un graphique contenant une référence à une pièce de rechange et, d'autre part, une autre cellule contenant une notation mathématique contenant elle-même un appel à une valeur technique définie en base externe.

Ce minimum d'intégration serait réalisé en l'attente du vote et de l'implémentation des schema. Les schema devraient être capables de supporter des modèles d'intégration plus complexes que les DTD. Il resterait alors pour seul rôle aux DTD celui de définir les entités générales des *internal* et *external subsets*.

5.2. État de maturité du marché logiciel

XML, en tant que recommandation de base est une spécification mûre. Elle est d'autant plus mûre qu'il existe beaucoup de spécifications d'utilisation d'XML, dont beaucoup sont d'une excellente facture. La qualité de ces spécifications s'explique aisément.

Ainsi, XSLT et XSL ont été créées très rapidement, car elles sont héritières de la norme *DSSSL* de l'ISO, qui avait déjà été investi d'un temps important consacré à la définition des concepts qui lui étaient nécessaires. Ainsi aussi, *XLink* et XPointer suivent le même chemin, spécifications elles-mêmes héritières de la norme *HyTime* de l'ISO, qui avait posé les bases de la synchronisation multimédia et des notions de mise en relations d'informations.

Dans un premier temps, les outils sont, sans exception, venus du monde SGML. En effet, passer d'outils SGML à des outils XML ne posait que peu de problèmes, à la question près, mais fondamentale, d'*Unicode XML*. La comparaison avec SGML s'arrête là. En effet, à la différence du lent développement des outils SGML, les outils XML se développent maintenant extrêmement rapidement : ceci est certainement lié aux besoins importants révélés par le Web, en termes de réutilisabilité de l'information.

Ceci est aussi lié au fait que XML sort du monde documentaire, pour s'intéresser à tout échange de données entre processus informatiques. Du coup, l'offre étant bien réelle et existante, il est possible de tester les recommandations aisément et de directement décider comment les utiliser dans des applications industrielles. Ceci est d'autant plus vrai que beaucoup de projets sont développés en « sources partagés » : ils concernent surtout les outils partagés et de base, comme les *parsers* et langages d'application.

5.2.1. Outils d'édition XML

À tout bien tout honneur, la spécificité d'XML dans le domaine documentaire est qu'il faut rédiger des documents. Il faut donc bien commencer par cette catégorie d'outils.

Aujourd'hui, les *éditeurs*XML « florissent », mais peu sont réellement utilisables dans le domaine documentaire. SGML a prouvé que l'objectif d'un *éditeur* structuré n'est pas seulement de permettre de saisir du texte et des balises de structures. Il s'agit de masquer la complexité de cette saisie, au profit d'un outil permettant à l'utilisateur de structurer ses idées... de s'intéresser à des contenus rédactionnels plutôt qu'aux modes opératoires nécessaires à la rédaction de documents conformes à un modèle.

Si des outils comme XMLspy sont certainement utilisables dans des environnements de rédaction de programmes XSLT ; beaucoup hésiteront à les mettre dans des mains de rédacteurs. Du coup, dans ce domaine, l'offre se résume aux *éditeurs* du monde SGML, qui sont maintenant passés à XML.

Le leader souvent reconnu est Epic, son outil prend en compte XML et ses DTD. Il annonce pour cette année une prise en compte progressive des schema. Son compétiteur, dans le monde SGML, était Adobe FrameMaker + SGML. Celui-ci sera pour l'instant difficile à utiliser en environnement hétérogène, du fait de la prise en compte d'XML au seul moment de l'*export* de documents (et pas au moment de l'*import*). Reste XMetaL, qui vise un marché plus ouvert, avec des prix nettement plus bas. Outil de bonne facture, il permet aussi de réaliser de véritables environnements de saisie.

Par ailleurs, une offre semble de développer autour d'*add-on* à Word. Ainsi, S4Text propose, avec sa version 2, une solution qui semble viable pour beaucoup de documents XML. On suivra aussi avec attention WorX SE, dont la proposition paraît intéressante.

5.2.2. Outils de validation et de traitement

Dans ce domaine, que ce soit pour des implémentations de SAX, de DOM ou de XSLT, il n'existe que peu d'offre à proprement parler commerciale. Il semble que tous les produits actuels soient réalisés en « sources partagées ». Les développements avancent relativement vite et l'unique problème qui se posera à tout intégrateur de système est la course aux versions : durant le temps de cette étude, trois versions successives de Xalan ont été utilisées.

Plus précisément, du côté des outils de validation, les résultats des tests réalisés lors de cette étude sont probants : les principaux outils du marché fonctionnent et sont plutôt rapides (pour des résultats plus approfondis, voir section 6.).

Du côté des outils de traitement, l'offre XSLT, DOM et SAX est vaste et semble assurer une bonne conformité aux standard. Elle devrait être testée plus en profondeur, avant de se prononcer plus franchement.

Du côté des recommandations elle-mêmes, certains se poseront des questions sur le fait de savoir, dans des activités d'intégration de systèmes, quelle recommandation utiliser ? D'un point de vue théorique, tout semble simple ! XSLT est un langage de transformation lié à des notions de présentation et de formatage de document : dédié au formatage, ce n'est donc pas un langage de programmation pour réaliser n'importe quelle opération sur des documents. DOM est une spécification de programmation en cours de standardisation, il est basé sur des Infoset et, donc, sur l'utilisation d'un arbre d'objets typés *complet*. SAX, basé sur une lecture séquentielle d'événements dans un flux XML, n'est pas une recommandation du W3C, et il semble qu'elle ne le sera pas.

D'un point de vue pratique, le recouvrement entre XSLT et DOM est important, vu la puissance du langage XSLT. Du coup, il est aujourd'hui beaucoup utilisé pour des transformations simples, depuis XML vers XML. Le choix sera donc lié à des architectures de systèmes et la coopération des utilisations est tout à fait possible.

Entre DOM est SAX, même s'il existe un recouvrement, il vaut mieux s'intéresser à des notions de complémentarité. En effet, dès que l'on sort du Web, et de ses présupposés petits documents, il n'est pas envisageable d'utiliser uniquement DOM. Par exemple, une implémentation DOM ne sera pas capable de manipuler un document de quelques centaines de méga-octets. En revanche, le mode événementiel de SAX permettra des traitements sur des documents de cette taille. Par ailleurs, beaucoup préfèrent aujourd'hui SAX à DOM, du fait de la simplicité de son *API*.

Pour finir, il serait important que la spécification SAX ne soit plus seulement l'œuvre d'une communauté de développeurs. En effet, le débat du jour, au moment où cette étude est publiée, concerne la reprise de la maintenance de SAX, du fait que David Megginson décide d'en abandonner le 'leadership'. Sur la liste de diffusion (*Who will maintain SAX*), personne ne souhaite prendre cette responsabilité et tous pensent que c'est une communauté formelle qui doit la prendre ... avec une préférence pour OASISopen. Le W3C a été rapidement éliminé, du fait de ses présupposées lenteurs à mettre en place et à faire évoluer une recommandation. Pourtant, partie intégrante des recommandations du W3C, SAX permettrait aux acteurs DOM et SAX de confronter leurs standards, afin

de mieux les faire coopérer. Il serait possible d'envisager un fonctionnement dual permettant, comme le proposait le logiciel Balise, de conjuguer dans un même programme le mode événementiel et le mode arborescent.

5.2.3. Outils de présentation

Pour utiliser des documents XML sur Internet, peu de choses à dire... Tout peut être réalisé en utilisant XSLT et l'offre logicielle est suffisante. Il n'en va pas de même lorsqu'il s'agit de publier sur papier : que ce soit avec des technologies SGML ou XML, la problématique reste complexe et peu d'outils « se » lancent sur le sujet.

Publier sur le Web Il est tout d'abord nécessaire de décider de la façon de publier sur le Web. Les ressources sont-elles publiées sous forme XML ou issues d'une transformation d'une base XML pour être publiées en HTML ? Si elles sont issues d'une base XML, qui en assure le formatage ? Le serveur Web ou le poste client ?

S'il faut transformer une base de production XML en une base HTML, tout peut être réalisé en utilisant XSLT. Il en va de même au niveau du serveur, où il suffit de déclencher un processus de conversion sur celui-ci. Par exemple, le serveur Apache intègre ces fonctionnalités grâce à Cocoon, du groupe XMLApache.

Sur le poste client, aujourd'hui, seul Microsoft Internet Explorer, dans ses versions 5.0 et supérieures, prend en compte XML, avec, aujourd'hui, des beta versions de ses outils. Du coup, utiliser XML sur le poste client ne peut se pratiquer que dans des environnements où l'on maîtrise complètement les équipements matériels et logiciels des personnes consultant l'information.

Tout ceci relève de la proposition purement technique : la véritable question serait de comprendre pourquoi utiliser XML sur des postes de travail, clients du Web.

Tout d'abord, de quel XML est-il question ? Est-il dans les prévisions de Boeing (ou d'Aérospatiale) de diffuser ses documentations sur Internet *telles qu'elles existent en interne* ? Certainement pas, car il figure dans ces documentations des informations relevant du secret industriel que ces industries ne souhaitent pas partager. Ainsi, même si l'information délivrée est sous forme XML, la plupart du temps, elle sera au préalable transformée, vers une version publiable, en fonction du lecteur.

Une transformation vers un modèle XML *ad hoc* ou une transformation vers HTML est de même nature et ne prend pas plus de temps : la question reformulée revient alors à comprendre quelles sont les limites de HTML, comme format de consultation ? Ces limites correspondent à des applications aujourd'hui émergentes où le navigateur Web doit recevoir une information intelligente capable de coopérer avec des applicatifs locaux.

Exemple ? Une personne consulte son relevé de compte en banque via son navigateur et souhaite l'importer automatiquement dans son gestionnaire de compte bancaire favori... Pour que cette coopération puisse se réaliser, plutôt que de n'avoir qu'une représentation typographique d'informations, il faut obligatoirement un format de données identifiant de façon non ambiguë les différents éléments d'informations liés à

une opération bancaire. Cet exemple a le mérite d'être simple à comprendre : il est limité, en ce sens que si la consultation de telle information est réalisée de façon régulière, il semble préférable de recevoir cette information par courrier, de façon à pouvoir automatiser le traitement d'un tel type d'information.

Un autre exemple bien connu (car souvent mis en avant sur le site de Microsoft aux débuts de XML) consiste à permettre à l'utilisateur de « voir » les informations reçues de différentes manières, selon différentes feuilles de styles. Comme ces feuilles de styles sont souvent préparées par le concepteur, il n'y a pas ici de nouvelles fonctionnalités proposées... tout juste un artifice permettant de décharger le serveur Internet d'un ensemble de transactions.

Ces deux exemples sont liés a des applications grand public. Pour les Intranet ou les Extranet, les exemples foisonnent : le système de consultation est conçu comme un véritable poste de travail qui donne, d'une part, accès aux ressources centrales et départementales de l'entreprise et, d'autre part, aux ressources locales. Il est alors du rôle du poste de travail de réaliser la coopération de ces différentes ressources et XML est alors le meilleur format de communication : ce que l'on voit à l'écran doit aussi être utilisé pour la coopération d'applications.

Finalement donc, hors la sphère privée (Intranet ou Extranet) à forte valeur ajoutée, où il faut, dans tous les cas, agir sur le poste de consultation en termes d'intégration de systèmes, afin de réaliser une coopération avec des applicatifs locaux, voire départementaux, on s'aperçoit que les exemples grand public sont difficiles à trouver. Du coup, il est possible de penser que les navigateurs Internet implémenteront lentement des environnements de traitement XML, n'incluant pas seulement XSLT, mais aussi *XLink*, XPointer et tout *XSL*.

En revanche, s'il s'étend comme promis, le poste multifonctionnel de consultation issu des technologies WAP utilisera beaucoup plus rapidement ces technologies. Par exemple, une personne ayant aujourd'hui un *nokia communicator* peut, dès aujourd'hui, « beamer », envoyer une carte de visite à un collègue ayant un *Palm Pilot*. Le résultat du transfert est extrêmement décevant, car l'information transférée ne contient que de l'information textuelle sans aucune structure. Le récepteur doit alors tout « copier/coller » dans son annuaire personnel, information par information. Ici, le fait d'être dans un environnement applicatif où les systèmes ne sont pas tous de même nature nécessite des protocoles permettant de réaliser des échanges selon un format neutre. Les promoteurs du Wap ne s'y sont pas trompés, qui utilisent dès à présent XML dans leur format *WML* de consultation et d'interaction sur téléphone portable.

Publier sur papier Dans le monde SGML, publier sur papier se faisait soit par transformation vers un langage de composition (voire de photocomposition), soit en utilisant des outils intégrés, utilisant une spécification électronique de définition de formatage. Dans tous les cas, les résultats n'étaient pas trop décevants, tant que la présentation n'était pas trop complexe et qu'elle ne faisait pas trop intervenir de réserves graphiques dans des textes multi-colonnes. Dans tous les autres cas, il fallait intervenir manuellement. Dans le domaine XML, il en va à peu près de même. Ceci peut s'expliquer par le

fait que la typographie liée à l'impression est ancrée dans notre histoire et, malheureusement peut-être, cette typographie est alors très complète et, donc, complexe.

Dans une première approche, dans des recommandations comme XSL, il n'y a rien de plus que ce que l'on retrouve dans les traitements de texte usuels (ou, plutôt, dans les outils de PAO usuels). Dans une seconde approche, c'est un peu plus complexe, car les outils de PAO n'ont pas l'habitude de résoudre des *mises en pages complexes de façon automatisée*, à partir de documents n'ayant *a priori* aucune indication de style. Du coup, XSL, dans le prolongement de DSSSL, introduit les notions « d'*aire* » qui permettent de définir, en fonction du flux d'entrée, des aires dans lesquelles vont « couler » des sous-arbres XML. Ces notions d'aires sont nécessaires, par exemple, dès lors que l'on veut faire du formatage multilingue, multicolonnes et synchronisé (comme au Parlement européen) ou dès lors que l'on veut faire de la documentation technique.

Ces notions, définies dans les recommandations de formatage sont nouvelles et n'ont, à ce jour, été que très peu implémentées, du fait que tous les outils de PAO étaient, par nature et par utilisation, interactifs. Il s'agit donc d'inventer de nouveaux outils et il est important de savoir quels acteurs seront intéressés par ce marché.

Du côté du logiciel en « sources partagées », FOP semble prometteur. Par ailleurs, certains nouveaux acteurs entrent sur ce marché, comme XEP Rendering Engine. En revanche, on ne voit pas l'ombre d'un balbutiement du côté des logiciels de PAO traditionnels, qu'ils soient « bas de gamme », ou « haut de gamme ». Les éditeurs de logiciel semblent attendre. Il reste encore les outils du monde SGML, comme Epic, comme 3B2 ou encore comme XyEnterprise Production Publisher ; Adobe FrameMaker + SGML sera aussi utilisable, à une conversion de XML vers SGML près, pour être capable d'importer de l'information XML.

6. Les parsers

Un des objectifs de cette étude était de réaliser un état des lieux des outils de *parsing*, de leur qualité et de leur utilisation en environnement ouvert, dès lors qu'il est nécessaire de valider des documents reçus. L'objectif n'était pas de réaliser un ensemble de tests poussés, comme peuvent le proposer le NIST ou d'autres associations de test de conformité. L'objectif était plutôt de regarder si, dans des cas connus d'utilisation, tout fonctionnait correctement.

Cette section présente donc les principaux résultats en commençant, tout d'abord, par une explication de vocabulaire, par ce que recouvrent les termes de *parsers*, « *reader* », processeur XML, etc.

6.1. Qu'est-ce qu'un parser?

Dans le domaine SGML, tout était admis comme simple ! Un document ne pouvant être lu sans son modèle, un *parser* était supposé lire le document SGML et le valider en fonction de son modèle. Naturellement, ce n'était que le sens commun donné à une spécifi-

cation ardue à lire qui, cependant, précisait les notions de « *validating* SGML *parser* », de façon indépendante de celles de « SGML *application* ».

Quoi qu'il en soit, décider qu'un *parser* doit reporter des erreurs conformément à une spécification de validation est plus difficile à réaliser dans le domaine XML. Ceci est lié, d'une part, à la double vision que l'on peut avoir d'un même document (il est *bien formé* ou *valide*) et, d'autre part, à l'arrivée d'une recommandation concurrente aux DTD d'XML : les schema.

Du coup, qu'est-ce qu'un *parser* ?

> *Parser: a computer program that breaks down text into recognize strings of character for further analysis.* *Extrait de* Webster OnLine

Un *parser* serait donc un petit module de programme, dont le seul objectif serait de découper un flux d'entrée en un ensemble de « mots » (*token*) qui seront ensuite proposés à d'autres applications. Un *parser* n'est donc pas un programme en soi : il est associé à un module de traitement et c'est la nature de ce module qui prime. La spécification XML parle alors de « processeur XML ».

Un processeur XML inclut des notions de validation lexicale et structurelle de document. Il propose aussi son module de traitement *ad hoc*, basé sur une représentation de l'information contenue dans le document (fichier) lu. Aujourd'hui, la représentation utilise soit l'interface *SAX*, soit des interfaces définies dans le « cœur » de DOM.

Construire une application XML revient donc à utiliser des *parsers* pour des outils fonctionnels différents :

- Validation XML
  Ce sont des outils dont le rôle est de *parser* un document et d'en définir leur validité, pour savoir s'ils sont « valides » et/ou « bien formés ».
  Valide s'entend aujourd'hui au regard d'une DTD ; ce sera, demain, valide aussi au regard d'une DTD et/ou d'un schema. Du coup, on est en droit de penser que l'architecture de ce type d'outil différenciera le *parsing* de la fonction de validation.
- Chargeur de document XML
  Outil logiciel assurant la fonction de *parsing*, de validation et la fonction de fournir une représentation du contenu du document à d'autres modules fonctionnels. Dans les implémentations actuelles, les représentations utilisées sont celles nécessaires à DOM ou SAX ; n'importe quelle autre représentation est possible, par exemple une implémentation basée sur les Infoset.
- Processeurs XML
  Ce sont des modules fonctionnels qui *parsent* et utilisent l'information *parsée*.
  Y en aura-t-il beaucoup, packagés en tant que tels, sans utiliser des couches de validation et de chargement externes ? Certainement, ils existeront, mais dans des cas bien précis, d'environnements de développement peu ouverts (par exemple, ceux utilisés dans les téléphones portables).

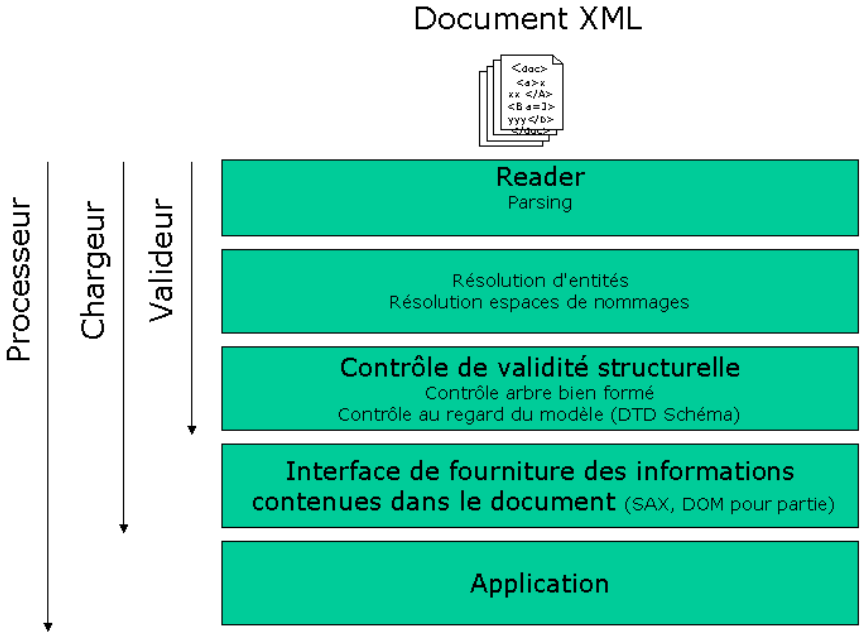

FIGURE 1 – *Architecture d'un processeur* XML

Sinon, ces outils utiliseront des chargeurs de documents. C'est ainsi que bon nombre d'implémentations de XSLT existantes sont basées sur une interface SAX, pour la lecture des documents XML.

Comprendre comment peuvent s'architecturer ces composants logiciels est encore une autre affaire. On se référera avec profit à l'article de Simon Saint-Laurent sur le sujet *Toward a layered model for* XML. Pour lui, il est nécessaire de construire une véritable architecture logicielle en couches allant du *parsing* jusqu'à la représentation de l'information (figure 2).

6.2. Logiciels testés

Les logiciels testés tournent soit en environnement Java, soit en environnement Microsoft Windows. Ils ont été choisis soit parce qu'ils étaient représentatifs du marché, soit parce qu'ils étaient stratégiques, dans ses choix internes d'entreprise.

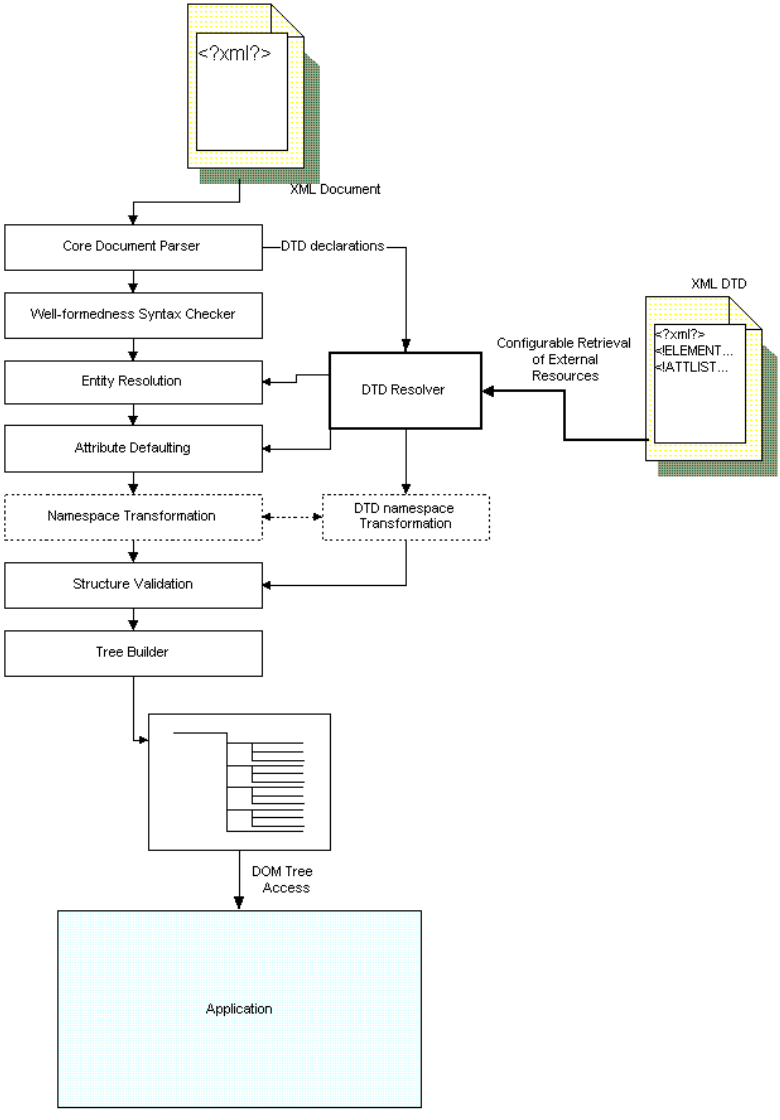

FIGURE 2 – *Building a namespace-aware, validating, tree-building parser out of standard components* — Simon Saint Laurent - 1999

Les logiciels testés sont les suivants (voir glossaire en page 83) :

- Microsoft XML Parser de *Microsoft*
  La raison principale de ce choix est liée à Microsoft Internet Explorer, logiciel incontournable dès lors qu'il s'agit de lire des documents XML sur le Web.
  La version testée est la version 3.0, publiée beta test, en juillet 2000.
  Microsoft XML Parser supporte la recommandation XML 1.0 et inclut des fonctionnalités de *parsing* liées aux schema, à DOM (niveau 1) et à SAX (version 2). En mode SAX, le *parser* ne sait pas valider un document au regard d'une DTD.
- Xerces Java Parser (Java), du groupe en source libre XMLApache
  Une des raisons principales du choix de ce logiciel est la cohérence des différents outils de ce groupe, avec Xalan, FOP, tous intégrables au fameux serveur Web Apache. L'autre raison est la reconnaissance de ce groupe dans la communauté XML.
  La version testée est la version 1.2.0.
  Xerces Java Parser supporte la recommandation XML 1.0 et inclut des fonctionnalités de *parsing* liées aux schema, à DOM (*niveau 2*) et à SAX (version 2).
- Java API for XML Parsing (Java), de *SUN*
  Ce choix est lié à *SUN*, et à son influence sur Java dans les environnements ouverts.
  La version testée est la version 1.0.1.
  Java API for XML Parsing supporte la recommandation XML 1.0 et inclut des fonctionnalités de *parsing* liées à DOM (niveau 1) et à SAX (version 1).
- Oracle XML Parser (Java)
  La raison de ce choix est stratégique : *EDF* utilise beaucoup cette base de données.
  La version testée est la version 2.0.2.9.0.
  Oracle XML Parser supporte la recommandation XML 1.0 et inclut des fonctionnalités de *parsing* liées à DOM (niveau 1) et à SAX (version 1).

Note : XP - an XML Parser in Java, de James Clark, faisait partie de la liste initialement choisie. À la lecture de la documentation, il s'est avéré que ce logiciel ne proposait pas de mode validant. Du coup, l'orientation très documentaire de cette étude reposant sur le besoin de documents valides au regard de DTD, nous a amené à supprimer ce logiciel au profit de Java API for XML Parsing.

6.3. Synthèse des résultats

Tout d'abord, il est intéressant de remarquer que, dans un environnement Windows, il n'existe pas de programme *packagé*, directement exécutable. Valider un document nécessite, pour tous les outils, de développer un programme applicatif. De prime abord, ce programme n'est pas aisé à réaliser, du fait de la faiblesse des documentations existantes. Il faut bien prendre cette information comme un indicateur du peu de cas fait par les outils concernant cette activité autonome de validation. Étonnant !

Quoi qu'il en soit, d'un point de vue performances, les *résultats sont extrêmement positifs en termes de rapidité de traitement*. Pour toute personne habituée au *parsing* SGML, le pari d'XML consistant à définir des syntaxes simples et explicites, non basées sur la DTD, est gagné.

Plus précisément, tous les *parsers* testés sont très rapides en mode *non validant*. Concernant le mode *validant*, seul Oracle XML Parser affiche un temps de traitement plus long,

mais cependant très correct. Microsoft XML Parser n'a pas pu être testé en mode *validant*, dans la mesure où ce mode n'est pas encore disponible avec son implémentation SAX ; la validation au regard d'un modèle fonctionne seulement avec DOM.

D'un point de vue fonctionnel, *tous les parsers testés sont capables de valider un document* pour savoir s'il représente bien un *arbre bien formé* ou s'il est *conforme à une* DTD. Cette validation s'entend sur le jeu de tests simples, développé pour cette étude et représentant les erreurs communes que l'on peut, d'expérience, trouver dans des outils de ce type.

D'un point de vue mise en œuvre, même si Microsoft XML Parser soigne un peu plus ses messages d'erreur que les autres, les résultats sont encore peu probants et il est souvent difficile de comprendre l'erreur trouvée, et aussi de savoir où elle se trouve. Cela est d'autant plus vrai que, par défaut, tous les programmes arrêtent le *parsing* à la première erreur rencontrée. Corriger un document XML contenant plus d'une erreur relève alors de la gageure et on espère que cette activité ne sera pas fréquente, grâce à l'utilisation d'éditeurs structurés et *validants*. Dans le détail :

- En mode arbre bien formé, Xerces Java Parser semble contenir un peu plus de bogues que les autres, surtout quant à la prise en compte des jeux de caractères, qu'ils soient Unicode, XML ou ASCII. En mode *validant*, outre un bug sur la prise en compte des ID/IDREF, il n'implémente pas les sections conditionnelles *Extensible Markup Language (Second Edition) (Version 1.0)*. D'un point de vue performances, il est relativement efficace dans les deux modes. Du point de vue de la documentation, il est relativement complet et offre, en plus de la documentation de l'*API*, quelques commentaires sur les exemples fournis, des FAQ et des notes de version. Il est également important de préciser que Xerces Java Parser sert de couche basse par défaut au processeur Xalan. Bien qu'acceptant théoriquement d'autres *parsers*, Xalan est en pratique uniquement disponible avec Xerces Java Parser. On peut trouver des exemples d'implémentations suffisamment clairs dans la distribution du produit.
- Oracle XML Parser n'implémente pas non plus les sections conditionnelles. Autrement, il passe l'ensemble des tests sans problème. On regrettera cependant la faible qualité de ses messages d'erreur. D'un point de vue performances, il est extrêmement efficace en mode *non validant* ; les performances sont moins bonnes, mais très acceptables, en mode *validant*. La documentation est clairement réduite à la documentation de l'*API* (type JavaDoc) n'offrant pas d'information supplémentaire. La distribution du *parser* contient un exécutable (ORAXML) permettant de lancer le *parsing* sur un document XML : c'est le seul produit offrant directement un tel outil. Un transformateur XSLT (ORAXSL) est également disponible dans la distribution, s'appuyant sur la couche basse testée. Enfin, très peu d'exemples d'implémentations sont fournis.
- Java API for XML Parsing passe tous les tests sans problème, y compris ceux de performances, où il est le plus efficace. La distribution fournit une documentation assez réduite, contenant cependant des *Release Notes*. Les exemples d'implémentation fournis sont suffisamment clairs.
- Microsoft XML Parser passe tous les tests sans problème, en mode DOM. Les performances sont aussi très acceptables en mode *non validant* (en mode SAX). Le SDK correspondant à la DLLmsxml a un très mauvais programme d'installation. Le SDK (offrant des *API* en C++ et en Visual Basic) contient une documentation. Aucun exemple n'est disponible dans la distribution, mais il est possible d'en télécharger depuis le site de *Microsoft*.

6.3.1. Méthodologie de test

Pour les outils Java, un programme a été écrit permettant d'appeler la validation de documents. Ce programme a été réalisé au niveau *SAX*, des interfaces proposées. En ce qui concerne Microsoft XML Parser, le programme a été réalisé au niveau DOM car le mode *validant* n'était pas accessible avec les interfaces SAX.

Les tests réalisés ont été définis en fonction d'un ensemble restreint de fonctionnalités qui, d'expérience avec les documents et/ou avec SGML, paraissent importantes pour traiter de l'information structurée.

Ce jeu de tests est alors non exhaustif et surtout extrêmement subjectif. Par ailleurs, les tests se sont limités aux fonctionnalités proposées dans la recommandation XML 1.0 : les espaces de noms et les schema n'ont donc pas été pris en compte.

Les tests réalisés se sont intéressés, plus précisément, aux points suivants :

- jeux de caractères :
  - Support UnicodeXML dans les caractères latins connus de la langue française et hors le jeu HTML dans les :
    - PCDATA,
    - noms d'élément et d'attribut,
    - valeurs d'attribut ;
- documents bien formés :
  - tests de base
    - un élément n'est pas fermé, n'est pas ouvert,
    - reconnaissance des éléments vides,
    - syntaxe de base des attributs,
    - prise en compte des *Processing Instructions*, des commentaires ;
  - tests poussés pour valider :
    - que l'on ne peut avoir des noms d'élément et d'attribut ayant des caractères « : », en dehors des *namespaces*,
    - la prise en compte de *l'internal subset des* DTD, sur des exemples simples de définitions de contenus avec des éléments ou des définitions d'attribut,
    - qu'il existe toujours bien une entité system dans une définition d'entité ;
- DTD, valider :
  - le minimum quant aux déclarations d'éléments et d'attributs ;
  - la reconnaissance des sections conditionnelles ;
  - la possibilité de déclaration multiple d'attribut pour un même élément ;
  - que la définition énumérée de plusieurs valeurs d'attribut d'un même élément peut être la même ;
  - l'utilisation de jeux d'entités paramètres ou générales, internes ou externes.

6.4. Résultats bruts

On trouvera les résultats bruts sur le web dans le document initial :
http://www.mutu-xml.org/xml-base/articles/PUB-EDFSTD-FR-6.html

Annexe 1 — Mutu-XML : Mutualiser l'effort de montée en compétences sur XML

XML l'incontournable

XML est un standard mondial, défini par le *World Wide Web Consortium* (W3C) qui permet de décrire la structure d'un document et de qualifier chacune des informations qu'il contient. Un document XML peut être lu et exploité de façon automatisée sur un navigateur, mais également lu et exploité par une application informatique qui en comprend le contenu et le « sens ».

XML est aujourd'hui incontournable ! Adopté par les plus grandes entreprises mondiales, il est à la base des principaux standards d'échanges d'informations de demain : le Web (et ses versions mobiles), le commerce électronique et les EDI, etc. Au-delà, **XML est en train de devenir le fondement de la production et de l'échange de toutes les données numériques.**

Rapidement, les entreprises qui n'utiliseront pas XML seront à la fois moins efficaces en interne et de plus en plus exclues des flux d'échanges électroniques qui structureront les marchés.

XML nécessite un effort important de montée en compétences

Toute l'information technique existe sur XML, mais éparse, peu synthétisée, difficile d'accès et généralement rédigée en anglais. Par ailleurs, l'information est ardue à lire, lorsqu'on entreprend de travailler les recommandations ; elle est également peu synthétisée, pour qui s'intéresse aux sites spécialisés sur le sujet.

En France, les experts sont rares ; les techniciens, les documentalistes, les responsables Web... ne savent pas comment se saisir du sujet. Les décideurs manquent des informations de synthèse sur les enjeux, les objectifs, les applications, les conditions de mise en œuvre ; quand ils en disposent, ils cherchent en vain les compétences capables de les exploiter.

Pour changer cet état de fait, il est nécessaire de donner accès à une information plus pertinente et élaborée, rédigée en français et pensée pour un public français. Cette information doit aider la compréhension et la mise en œuvre. Elle doit dégager les objectifs et les enjeux, identifier les besoins et les applications, préciser les conditions et les méthodes de réalisation.

Dans bon nombre d'entreprises, les départements Recherche et Développement ou les services de Veille dépensent beaucoup d'énergie à rechercher de l'information sur XML. Ils auraient tout à gagner à bénéficier d'une information commune, partagée, organisée et opérationnelle. En effet, connaître XML n'offre pas en soi d'avantage concurrentiel particulier : l'avantage concurrentiel se gagnera au travers de l'utilisation intelligente d'XML, en interne et dans les échanges avec les partenaires de l'entreprise —

ce qui suppose que le plus grand nombre de partenaires de l'entreprise connaisse et utilise également XML.

Les objectifs du projet Mutu-XML

« Mutualiser l'effort de montée en compétences sur XML » (Mutu-XML) est un projet ouvert, à but non lucratif, placé sous l'égide de la Fondation Internet Nouvelle Génération (FING). Son objectif général est de favoriser le développement des compétences et des usages de XML en France.

Mutu-XML se fixe pour objectifs de :

- synthétiser et publier en français les informations de référence et d'actualité essentielles sur XML ;
- aider l'ensemble des acteurs à découvrir, comprendre et utiliser XML :
  - objectifs et enjeux,
  - principes, standards, fonctionnement,
  - implémentations et applications,
  - méthodes et conditions de mise en œuvre,
  - outils et compétences, etc. ;
- favoriser l'intégration ou l'acquisition de compétences XML dans les entreprises et les sociétés de service, pour rendre possible la conduite de projet dans ce domaine ;
- faire connaître les réalisations exemplaires des organisations et des sociétés de services françaises ;
- créer une ressource communautaire permettant l'échange et l'élaboration collective.

Pour atteindre ces objectifs, Mutu-XML doit délivrer une information de synthèse et à forte valeur ajoutée, afin que chacun puisse y trouver l'essentiel, sans pour autant se noyer dans de l'information brute.

À qui s'adresse Mutu-XML ?

- aux décideurs soucieux de comprendre l'intérêt et les enjeux de XML ;
- aux informaticiens, désireux de connaître les standards, les outils, les applications ;
- aux documentalistes et archivistes qui cherchent à améliorer l'exploitation de l'information de leurs bases de références ;
- aux responsables de projets Web qui veulent enrichir leurs sites ou faciliter la transformation de leurs informations en direction de plusieurs terminaux (mobiles, etc.) ;
- aux sociétés de services et de conseil qui souhaitent préparer leur avenir et celui de leurs clients ;
- etc.

Un projet sous l'égide de la FING

Mutu-XML est un projet autonome placé sous l'égide de la Fondation pour l'Internet Nouvelle Génération (FING). Cette fondation *reconnue d'utilité publique*, qui a reçu le 10 juillet 2000 le soutien officiel du Gouvernement français, est un projet de Recherche et Développement collectif centré sur les usages de l'Internet de demain.

Le rattachement à la FING garantit l'indépendance de Mutu-XML vis-à-vis des acteurs du secteur. Il favorise également les synergies entre un projet à forte orientation technique et le monde des applications et des usages. Par ailleurs, le rattachement à la FING améliore la visibilité publique du projet.

Mutu-XML est en même temps un projet autonome, qui réunit son propre budget et dispose de ses propres structures d'animation et de régulation. Il est financé par la participation des entreprises et des organisations publiques ou associatives.

Contribuer à Mutu-XML, c'est :

- influer sur l'orientation du projet et le contenu du site ;
- bénéficier d'une veille permanente, à forte valeur ajoutée et adaptée à vos besoins ;
- faire partie de la communauté française des professionnels actifs autour de XML, pour échanger et confronter points de vues, interrogations et expériences ;
- coordonner vos efforts avec ceux des autres experts actifs dans les groupes et consortiums internationaux ;
- disposer d'une tribune pour faire connaître vos compétences et vos réalisations ;
- pouvoir participer à la commande et au pilotage d'études communes, et bénéficier de leurs résultats ;
- apparaître comme une entreprise pionnière dans le développement et l'utilisation de XML et des technologies associées ;
- s'afficher en tant que sponsor, dans ses actions de communication et sur le site du projet.

Contacts

Site Web : http://www.mutu-xml.org/index.html

Pour en savoir plus :

- Pierre Attar (Tirème : pattar@tireme.fr)
- Jacques-François Marchandise (FING : jfm@proposition.fr)

Annexe 2 : glossaire des noms et des produits cités

La page web de ce rapport (voir l'url donnée ici en note 1) propose une liste plus détaillée d'outils, portails, formations, FAQ et listes de discussions, DTD, acteurs et produits avec les url correspondantes. [N.D.L.R.]

AFNOR Association française de normalisation
http://www.afnor.fr/
Apache *The Apache Software Foundation*
http://www.apache.org/
Balise http://www.us.balise.com
BiblioML http://www.culture.fr/BiblioML/fr/dtds.html
BizTalk Microsoft BizTalk Server
http://www.microsoft.com/france/biztalk/default.asp
CML *Chemical Markup Language*
Voir la DTD dans *Journal of Chemical Information and Computer Science*, Vol xxx, 1999.
CSS *Cascading Style Sheets*
http://www.w3.org/TR/REC-CSS2/
DocBook http://www.docbook.org/xml/4.1.2/index.html
DOM Modèle de Document Objet
http://www.w3.org/TR/REC-DOM-Level-1/
DSSSL Sémantique de présentation de documents et langage de spécifications
ftp://ftp.oml.gov/pub/sgml/WG8/DSSSL/dsssl96b.pdf
DTD Définition de type de document
http://www.w3.org/TR/REC-xml#sec-prolog-dtd
Dublin Core http://purl.org/DC/documents/rec-dces-19990702.htm
EAD *Encoded Archival Description for Libraries*
http://lcweb.loc.gov//ead/tglib/tlhome.html
ebXML *Electronic business XML*
http://www.ebxml.org/specdrafts/req428.htm
EDI Échange de données électroniques
http://edifact-wg.org/WG9620Official9620Doc/98g104.PDF
EDIFACT *Electronic Data Interchange for Administration, Commerce and Transport*
http://www.edifact-wg.org/
Epic http://www.arbortext.com/Products/products.html
FING Fondation Internet Nouvelle Génération
http://www.fing.org/
GCA *Graphic Communication Association*
http://www.gca.org/
HTML *HyperText Markup Language*
http://www.w3.org/TR/html4/
HyTime *Hypermedia/Time-based Structuring Language*
http://www.ornl.gov/sgml/wg8/docs/n1920/
HyTime UG *HyTime Users' Group*
http://www.hytime.org/

IETF *The Internet Engineering Task Force*
http://www.ietf.org/
IETF LANG *Tags for the Identification of Languages*
http://www.ietf.org/rfc/rfc1766.txt
IPTC *International Press Telecommunications Council*
http://www.iptc.org/
ISO Organisation Internationale de Normalisation
http://www.iso.ch/
ISO 12083 EMS *The Electronic Manuscript Standard*
http://www.techstreet.com/cgi-bin/detail?product_id=52643
Java API for XML Parsing http://java.sun.com/xml/download.html/
Microsoft XML parser http://msdn.microsoft.com/downloads/toc.asp?
PaneName=Contents&URL=/code/sample.asp?url=/
msdn-files/027/000/541/msdncompositedoc.xml&ShowPane=true=sel
MTIC Mission interministérielle de soutien technique pour le développement des technologies de l'information et de la communication dans l'administration
http://www.mtic.pm.gouv.fr/
Mutu-XML Mutualiser l'effort de montée en compétences sur XML
http://www.mutu-xml.org/
OASIS *Organization for the Advancement of Structured Information Standards*
http://www.oasis-open.org/
Oracle XML Parser http://technet.oracle.com/tech/xml/parser_java2/index.htm
SAE J2008 *Recommended Organization of Vehicle Service Information for Interchange*
http://www.sae.org/servlets/otherProduct?
PROD_CD=J2008CD&PROD_TYP=SFTWR&COMMON_SUCCESS=TRUE
SAX *Simple API for XML*
http://www.megginson.com///SAX/index.html
Schema http://www.w3.org/TR/2000/CR-xmlschema-0-20001024/
SGML Langage normalisé de balisage généralisé
http://www.iso.ch/catf/d16387.html
SMIL *Synchronize Multimedia Integration Language*
http://www.w3.org/TR/smil20/
Style Sheets and XML *Associating Style Sheets with XML documents*
http://www.w3.org/TR/xml-stylesheet/
SVG *Scalable Vector Graphics*
http://www.w3.org/TR/2000/CR-SVG-20001102/index.html
TEI *Text Encoding Initiative*
http://www.tei-c.org/uic/ftp/P4beta/index.htm
Voir aussi *Cahiers GUTenberg* 24, juin 1996.
Unicode *The Unicode Standard*
http://www.unicode.org/
Unicode XML *Unicode in XML and other Markup Languages*
http://www.w3.org/TR/unicode-xml/
URI *Uniform Resource Identifiers*
http://www.ietf.org/rfc/rfc2396.txt

URL *Uniform Resource Locators*
http://www.ietf.org/rfc/rfc1738.txt
URN *Uniform Resource Names*
http://www.ietf.org/rfc/rfc2141.txt
WAP *Wireless Application Protocol*
http://www1.wapforum.org/tech/terms.asp?
doc=WAP-100-WAPArch-19980430-a.pdf
WML *Wireless Markup Language*
http://www1.wapforum.org/tech/documents/WAP-191-WML-20000219-a.pdf
W3C *World Wide Web Consortium*
http://www.w3.org/
Xalan http://xml.apache.org/xalan/
XEP Rendering Engine http://www.renderx.com/FO2PDF.html
Xerces Java Parser http://xml.apache.org/xerces-j/index.html
XForms http://www.w3.org/TR/xforms-datamodel/
XHTML *Extensible HyperText Markup Language*
http://www.w3.org/TR/xhtml1/
XLink *XML Linking Language*
http://www.w3.org/TR/xlink/
XML Langage de balisage étendu
http://babel.alis.com/web_ml/xml/REC-xml.fr.html
XML Apache http://xml.apache.org/
XML Base http://www.w3.org/TR/2000/CR-xmlbase-20000908/
XML Query *XML Query Language*
http://www.w3.org/TR/query-datamodel/
XML-Signature http://www.w3.org/TR/xmldsig-requirements
XML Spy http://www.xmlspy.com
XPath *XML Path Language*
http://xmlfr.org/w3c/TR/xpath/
XPointer *XML Pointer Language*
http://www.w3.org/TR/WD-xptr
XSL Langage de feuilles de styles extensible
http://www.w3.org/TR/2000/CR-xsl-20001121/
XSLT *XSL Transformations*
http://xmlfr.org/w3c/TR/xslt/
XTM *XML Topic Maps*
http://www.topicmaps.org/xtm/1.0/core.html