

# DANIEL MARTIN

Ing. I.D.N. Lic. ès Sciences

CONSEIL EN BASES DE DONNEES - EXPERT ASSERMENTE EN INFORMATIQUE

3, Chemin Départemental 110 - 78200 Menerville - FRANCE

Tél. (33) 1.34.78.09.95

e-mail [dmartin@dial.oleane.com](mailto:dmartin@dial.oleane.com) - <http://worldserver2.oleane.com/dmartin>

## **Stockage et interopérabilité en XML**

# Stockage et interopérabilité en XML

Ce texte analyse les caractéristiques et avantages du langage XML et des langages associés XSL, XLL et XQL en matière de stockage et d'interopérabilité. A titre d'exemple de solution, il décrit le serveur XML eXcelon de Object Design.

## Table des matières

<b>1</b>	<b>Définition des services de filtrage et transformation</b>	<b>4</b>
1.1	Mécanisme de conversion	5
<b>2</b>	<b>Les solutions XML</b>	<b>6</b>
2.1	Document XML et DTD	6
2.1.1	DTD et DCD	6
2.2	Comparaison de XML et HTML	7
2.3	XML est un standard de stockage et de partage de documents	7
2.3.1	Restriction	8
2.3.2	Recherches de données XML puissantes et performantes	8
2.3.3	Interpréteurs XML et vitesse d'analyse des documents	8
2.4	XML: une solution d'interopérabilité intéressante	9
2.5	Langage XSL	9
2.5.1	XSL: le langage des feuilles de style	9
2.5.2	XSL: le langage de transformation de données	10
2.6	XML + XSL: une puissante solution d'interopérabilité	11
2.6.1	Intégration des applications: synchrone ou asynchrone?	13
2.7	Accès à des documents XML par programme: standard DOM	13
2.7.1	XML et Java	14
2.7.2	Une alternative à la génération dynamique de pages dans le serveur	14
2.7.3	Utilisation de langages de script	14
2.8	Resource Description Framework (RDF)	15
2.9	Différences entre XML+XSL et d'autres solutions d'interopérabilité	15
2.9.1	L'interopérabilité limitée des protocoles client-serveur classiques	15
2.9.2	Interopérabilité nécessaire à une coopération ouverte	16
2.9.3	Avantages de XML pour la coopération ouverte entre applications	16
2.9.4	Autres différences d'interopérabilité entre XML et client-serveur	17
2.9.5	Recommandations de choix entre client-serveur et XML	18
2.9.6	Comparaison de XML avec EDI	18
2.9.7	XML a souvent besoin de HTML pour l'affichage	18
2.10	Deux langages complémentaires de XML: XQL et XLL	19
2.10.1	Langage XQL (eXtended Query Language)	19

2.10.2	Langage XLL (eXtended Link Language)	19
<b>2.11</b>	<b>Résumé sur les avantages de XML</b>	<b>20</b>
<b>2.12</b>	<b>Site XML francophone</b>	<b>20</b>
<b>3</b>	<b>Serveurs de données XML</b>	<b>21</b>
<b>3.1</b>	<b>Pourquoi des serveurs XML</b>	<b>21</b>
<b>3.2</b>	<b>SGBD XML oui, mais orienté-objets</b>	<b>21</b>
<b>3.3</b>	<b>Ce qu'on attend d'un SGBD pour serveur XML</b>	<b>22</b>
3.3.1	Gestion de tous types et structures de données	22
3.3.2	Mécanismes de stockage (persistance)	22
3.3.3	Conversion de données vers / depuis XML	23
3.3.4	Modes d'accès aux données et interfaces correspondantes	23
3.3.5	Mécanismes transactionnels	24
3.3.6	Performance	24
3.3.7	Recherche de données et XQL	24
3.3.8	Fonctions pour développement (modélisation et codage)	26
3.3.9	Conformité aux standards	26
<b>3.4</b>	<b>Exemple de serveur XML: eXcelon de Object Design</b>	<b>26</b>
3.4.1	Objectifs et principes de fonctionnement du produit	27
3.4.2	Exemples de données d'une base eXcelon	28
3.4.3	Structure de stockage: XMLStore	28
3.4.4	Fonctionnement de eXcelon	30
3.4.5	Performance d'un serveur eXcelon	32
3.4.6	Sécurité d'un serveur eXcelon	32
<b>3.5</b>	<b>Offres de serveurs et outils XML du mlarché</b>	<b>32</b>
3.5.1	L'offre de Microsoft	32
3.5.2	Autres offres	33

# 1 Définition des services de filtrage et transformation

J'écris que deux applications "coopèrent" lorsqu'elles échangent des données et/ou des services. C'est le cas par exemple dans une architecture client-serveur, lorsque la partie cliente de l'application coopère avec la partie serveur. Dans la coopération entre logiciels le middleware doit souvent *filtrer* les données qu'il prend à l'un pour les passer à l'autre. Ce filtrage commence par *reconnaître* certaines données pour les retenir ou les éliminer.

En général, toutefois, le filtrage ne suffit pas: il faut *transformer* les données, à l'aide d'opérations de changement de structure, de format, de calcul, etc.

Les services proposés ici ont pour but de transformer des données "d'entrée" en données "de sortie" par réécriture.

Exemple 1: transformer l'ensemble des lignes de table produites par une sélection dans la base de données de production en un document XML.

Exemple 2: transformer le document XML ci-dessus en un ensemble d'instructions SQL INSERT qui soit acceptable pour le SGBD du data warehouse et qui représente un certain niveau de consolidation (remplacement de lignes de détail par une ligne de total).

Dans les deux exemples ci-dessus il faut:

- Reconnaître les diverses colonnes du listing de sélection, par une comparaison appelée en anglais "pattern-matching"
- Choisir les lignes et les colonnes à retenir pour la sortie, par une opération de filtrage
- Récrire les données retenues dans un texte de sortie dont la structure convient aux besoins, par une opération de transformation.

Dans le texte qui suit, les opérations de reconnaissance, de filtrage et de transformation nécessaires à une réécriture conforme aux besoins seront appelées *conversion*.

Un service de conversion est nécessaire de plus en plus souvent pour créer des statistiques, interfacier deux applications ou fournir des résultats avec une structure (pas une mise en page!) imposée. Nous décrivons ici un mécanisme assez général que l'on peut mettre en oeuvre en tant que service. Ce mécanisme étant destiné à permettre la coopération de deux logiciels (celui qui a fourni les données d'entrée et celui qui reçoit les données de sortie) c'est un mécanisme de *middleware*.

## 1.1 Mécanisme de conversion

Pour qu'une conversion soit automatique, il faut qu'elle s'applique à des données d'entrée dont la structure suit des règles précises permettant de reconnaître ses éléments. Le problème de reconnaissance de la structure d'un tel texte par analyse syntaxique a été très bien étudié en théorie des langages, où il y a divers types de grammaires. En pratique, les structures les plus puissantes (sur le plan sémantique) que l'on sache à la fois décrire, reconnaître et traduire automatiquement en une autre structure sont les structures arborescentes (on dit aussi hiérarchiques).

Le mécanisme de transformation de données proposé ici est donc basé sur la reconnaissance d'éléments de la structure des données d'entrée, et des valeurs que prennent certains éléments. Tout élément reconnu peut être récrit automatiquement en sortie selon des règles précises. Ces règles peuvent aussi prendre en compte:

- des valeurs lues en entrée
- des tests sur des valeurs d'entrée ou l'existence d'un élément
- des demandes d'itération jusqu'à ce qu'une condition soit remplie.

Par définition, un élément non reconnu sera rejeté (au sens du filtrage) et ignoré pour la transformation; il fera l'objet ou non d'un message d'erreur.

Voici des exemples de données que l'on sait décrire sous forme analysable, donc convertir automatiquement:

- un ouvrage technique illustré
- un objet logiciel, avec ses méthodes et ses données
- un message email avec pièces jointes
- une liste de lignes de sortie d'une sélection dans une base de données
- une fenêtre graphique affichée par une application
- un arbre de décision avec des liens vers d'autres données.

Les éditeurs de logiciel s'orientent en ce moment vers une approche de ces problèmes de conversion basée sur le langage XML. Chaque éditeur fournira une passerelle entre ses structures de données propriétaires et le langage XML: Oracle fournira une passerelle entre les données de son SGBD et XML, SAP une passerelle entre ses propres données et XML, etc. Ces passerelles fonctionneront dans les deux sens: des données propriétaires vers XML et inversement.

## 2 Les solutions XML

XML est un langage de description de structures de documents recommandé par l'organisme de normalisation W3C (World Wide Web Consortium). Une structure est, par définition, un ensemble de sous-structures appelées *éléments*. Un document peut être un livre, avec comme éléments un titre, des chapitres, sections, paragraphes, figures, légendes, etc. Un document peut aussi être un message email, ses éléments étant alors une en-tête (émetteur, destinataire, date...), un corps de texte et des pièces jointes. Il peut, enfin, s'agir d'un message entre deux applications, par exemple entre une application commerciale et une application de comptabilité. Rédigé à l'aide du langage informatique XML, un document est comme un code source de programme: il doit respecter les règles syntaxiques du langage.

### 2.1 Document XML et DTD

Sur le plan conceptuel, un document XML a deux parties:

- le document proprement dit, obligatoirement présent,
- une description formelle optionnelle, appelée Document Type Definition (DTD). Cette description contient les règles syntaxiques que doit respecter le document. Tout se passe comme pour un programme que l'on écrit dans un langage de programmation: le langage a une grammaire, qui décrit les structures syntaxiques permises pour un programme et les mots réservés que l'on emploiera. Le DTD contient aussi, dans cette grammaire, la liste des éléments (concepts) du document; chaque concept est représenté par un mot réservé, comme le mot <prénom>. L'ensemble de ces mots constitue le vocabulaire autorisé ou, en langage savant "l'espace de noms". Comme HTML, XML dérive de SGML (Structured Generalized Markup Language); SGML aussi utilise des DTD.

Accompagné de son DTD, un document XML est comme un programme: il doit se conformer strictement aux règles de sa grammaire. On dit qu'il est "valide" lorsqu'on peut en vérifier la syntaxe avec cette grammaire. Sans DTD, un document peut quand même être "bien formé": sa structure est alors réputée décrite par elle-même et peut se contenter de respecter les règles générales du langage XML.

#### 2.1.1 DTD et DCD

Une DTD provient du monde SGML. Elle définit la structure d'un document, la liste des balises (tags), et leur structuration. Elle s'exprime en langage SGML.

Il existe une autre façon de décrire cette structure: le DCD (Document Content Definition), texte appelé "Schema" par Microsoft. Elle exprime la même chose qu'une DTD, mais la syntaxe employée est elle-même du XML. De plus, une DCD contient également la définition des types des attributs (entiers, chaînes de caractères, limites mini et maxi, etc.) ce qui permet d'avoir des analyseurs syntaxiques XML qui valident non seulement la structure d'un document XML, mais aussi les types et valeurs contenues.

## 2.2 Comparaison de XML et HTML

Différence entre SGML et XML: SGML a été conçu pour le stockage de documents dans de grandes bibliothèques centralisées, alors que XML a été fait pour une utilisation dans des environnements répartis, aussi bien pour le *stockage de documents* que pour *l'interopérabilité par échange de données*.

Le langage XML permet de définir et de nommer des éléments, avec leurs sous-éléments. Un document a forcément une structure arborescente, accessible en entier à partir de son élément de plus haut niveau. Ce type de structure est bien plus puissant et général que la structure relationnelle. Un arbre décrit la composition de chaque élément (ses sous-structures); un élément peut être un hyperlien pointant vers n'importe quel objet informatique n'importe où, y compris vers un élément de document XML (le même document ou un autre).

- HTML définit le format de mise en page (affichage ou impression) d'un document, alors que XML en définit la structure, le contenu, la sémantique, indépendamment de la mise en page. Les documents XML ont un DTD, les documents HTML n'en ont pas.
- La grammaire de HTML est fixe, définie par le standard, avec ses mots réservés et ses structures entre balises (tags). XML, au contraire, permet de définir n'importe quelle structure dans la mesure où elle est arborescente, avec notamment les balises que l'on veut. On peut ainsi définir des structures standard au niveau d'une profession, comme on a défini des formats standard d'échange de documents dans la norme EDI. C'est ainsi que des DTD standards ont été définis pour l'automobile, la chimie, les banques, les mathématiques, etc.
- Les documents HTML ont une structure séquentielle avec une en-tête (header) et un corps (body). Les documents XML, eux, sont des hiérarchies.
- En HTML, un hyperlien désigne un objet externe pour incorporation à la page en cours, remplacement de cette page ou affichage dans une fenêtre séparée; en XML les hyperliens sont beaucoup plus puissants: voir paragraphe XML ci-dessous.

## 2.3 XML est un standard de stockage et de partage de documents

De ce qui précède on peut tirer une première conclusion: XML est bien adapté au stockage de documents quelconques: ouvrages techniques illustrés, emails, codes de programmes, listings de sortie, etc. Les données stockées en XML sont indépendantes des systèmes d'exploitation de stockage ou d'accès, des langages de programmation et des formats de mise en page.

*XML est donc un standard universel de stockage.*

Il est donc précieux pour la coopération, car tous les clients qui accéderont aux données XML les verront de la même façon, avec la même structure et la même sémantique.

### 2.3.1 Restriction

Aujourd'hui le standard XML ne supporte que des textes ASCII. Cette distinction disparaîtra bientôt et XML supportera des types de données plus riches. Dès aujourd'hui, toutefois, les hyperliens permettent à XML de supporter des données quelconques; un document XML peut désigner n'importe quel autre texte, programme ou fichier de données et l'incorporer logiquement en tant qu'élément de sa structure. Lorsque l'élément ainsi désigné est actif (c'est un module de code exécutable) XML peut même lui passer des paramètres et en récupérer le résultat pour incorporation dans sa hiérarchie en lieu et place de la désignation (invocation) initiale.

### 2.3.2 Recherches de données XML puissantes et performantes

(Voir si nécessaire le texte d'introduction et de vulgarisation:

"Transmission des connaissances: une solution en XML" à l'adresse

[http://worldserver2.oleane.com/dmartin/Transmission\\_des\\_connaissances.htm](http://worldserver2.oleane.com/dmartin/Transmission_des_connaissances.htm))

Dans de grands systèmes de fichiers qui stockent des dizaines de milliers de pages HTML la recherche de textes qui satisfont certains critères est difficile, car le texte HTML doit être analysé mot à mot, puisqu'il n'existe pas de vocabulaire standard. Si un mot est mal orthographié, ou remplacé par un synonyme, une recherche peut échouer. Ensuite, des recherches basées sur les mots sans leur contexte peut produire trop de résultats sans intérêt. Exemple: une recherche de pages Internet basée sur le mot "Java" peut produire des pages concernant le langage informatique et d'autres concernant l'île.

Dans une bibliothèque de textes XML, la présence du DTD permet, sans parcourir tout un texte mot à mot, de savoir par exemple si le texte a un élément de type "arbre à cames": XML permet des recherches sémantiques d'éléments basées sur le DTD d'un document, sans balayage mot à mot du document, ce qui est très puissant et performant.

XML ne remplacera pas de puissantes applications spécialisées comme Fulcrum ou Verity Topic, mais c'est un standard qui peut aider de telles solutions à devenir plus efficaces et plus faciles à mettre en oeuvre.

Dans le domaine du commerce électronique, par exemple, XML permettra de chercher le produit approprié dans une masse de catalogues de vente électronique.

### 2.3.3 Interpréteurs XML et vitesse d'analyse des documents

Les documents XML sont traités par des interpréteurs XML, analogues aux interpréteurs HTML. L'interpréteur XML s'appuie sur la description DTD du document (lorsqu'elle existe) pour accéder à celui-ci en comprenant sa structure.

Les interpréteurs XML sont extrêmement rapides, parce que la technique d'analyse syntaxique est bien connue. La vitesse d'analyse que l'on peut espérer obtenir dans

un PC est de l'ordre de 1 MB/s: un texte de 50 KB (soit une vingtaine de pages) peut être analysé en environ 50 mS. Le résultat de l'analyse est une représentation orientée-objet en mémoire de l'arbre du document.

## 2.4 XML: une solution d'interopérabilité intéressante

XML est aussi une solution d'interopérabilité intéressante. Le langage peut-être utilisé pour créer des messages auto-descripteurs destinés aux échanges entre des applications aussi différentes que la comptabilité et la saisie de commandes. De tels messages sont indépendants des systèmes d'exploitation, des langages de programmation et des formats d'affichage. Des applications qui utilisent ces messages comprennent la structure et la sémantique des données qu'elle reçoivent, ce qui était impossible avec HTML.

XML permet à un client d'échanger des données avec beaucoup de sources de données ou de serveurs hétérogènes, sans connaître d'avance les structures, les types et les divers formats des données qu'il reçoit. Cette transparence est précieuse pour la coopération. Elle est inconcevable dans une architecture client-serveur traditionnelle.

Exemple d'utilisation de XML pour intégrer des applications hétérogènes: l'Object Management Group (OMG) a défini un mécanisme standard d'interopérabilité entre dictionnaires de développement appelé Structured Metadata Interchange Format (SMIF); il est basé sur XML pour offrir de riches possibilités de modélisation de données.

## 2.5 Langage XSL

De par sa conception même, XML sépare un document proprement dit de sa grammaire DTD, de son format d'affichage et de ses mécanismes d'hyperliens. Un document XML ne permettant pas, tel quel, de décrire le format d'affichage ou d'impression d'un document, la normalisation est en train de le compléter par un standard de mise en page, XSL (eXtended Style Language). Une description XSL (que l'on écrit en XML) s'applique à un document comme le fait son DTD, mais pour définir l'affichage des divers éléments. XSL est un complément "feuille de style" de XML comme CSS est un complément feuille de style de HTML. Le langage permet la reconnaissance, dans un texte XML, d'éléments syntaxiques ressemblant à un modèle donné, pour leur appliquer un format de mise en page. En anglais on parle de "pattern-matching". Les noeuds à reconnaître sont décrits à l'aide d'une notation de type "arborescence de répertoire": "book/author" est un modèle permettant de reconnaître les éléments "author" contenus dans un élément "book"; notez ici que "book" et "author" sont des éléments (noeuds) d'un arbre XML, pas des répertoires. XML est déclaratif (non-procédural).

### 2.5.1 XSL: le langage des feuilles de style

Une description XSL décrit la mise en page d'un document XML, en général dans un fichier distinct de celui de ce document. XSL est un complément de mise en page

de XML comme CSS (Cascading Style Sheets) est un complément d'affichage de HTML. Un utilisateur peut avoir ses feuilles de style XSL personnelles pour définir la manière dont il affichera les documents XML qu'il reçoit.

### **2.5.2 XSL: le langage de transformation de données**

Le principe de XSL consiste à reconnaître un élément XML puis à le mettre en page. La mise en page étant une sorte de transformation, les organismes de standardisation sont en train d'étendre le langage XSL, pour permettre la description de n'importe quelle transformation de texte réalisable par des processus de type "analyser la syntaxe puis générer un code" comme ceux d'un compilateur. Par exemple, on pourra définir comment transformer automatiquement les données tabulaires d'un document en instructions INSERT du langage SQL. La version Microsoft de XSL utilisée dans Internet Explorer 5 peut générer des textes HTML + CSS à partir d'un document XML + XSL; cette conversion est utilisée pour afficher un document XML + XSL comme ceux de OFFICE 2000 (WORD, EXCEL, POWERPOINT) à l'aide de l'interpréteur HTML de Internet Explorer.

L'exemple ci-dessous montre le code qui convertit un document XML en HTML.

```

<xsl:stylesheet>
  <xsl:template match = "/">
    <HTML>
      <BODY>
        <xsl:process-children/>
      </BODY>
    </HTML>
  </xsl:template>
  <xsl:template match = "author">
    <H1>
      <xsl:process-children/>'s fabulous
    </H1>
  </xsl:template>
  <xsl:template match = "recipe_name">
    <H2>
      <xsl:process-children/>
    </H2>
  </xsl:template>
  <xsl:template match = "meal">
    <TABLE><TR><TD><H3>EAT FOR:</H3></TD>
      <TD><H3><xsl:process-children/></H3></TD>
    </TR></TABLE>
  </xsl:template>
</xsl:stylesheet>

```

Exemple de feuille de style XSL: c'est du XML et cela rappelle HTML  
Ici, l'interpréteur XSL traduit le document XML en HTML pour affichage dans un interpréteur HTML

La norme XSL sera stabilisée courant 1999, mais d'ores et déjà les parties qui sont publiées constituent une spécification utilisable. On peut prévoir, à terme, un remplacement de HTML + CSS par XML + XSL, solution plus puissante et plus générale.

## 2.6 XML + XSL: une puissante solution d'interopérabilité

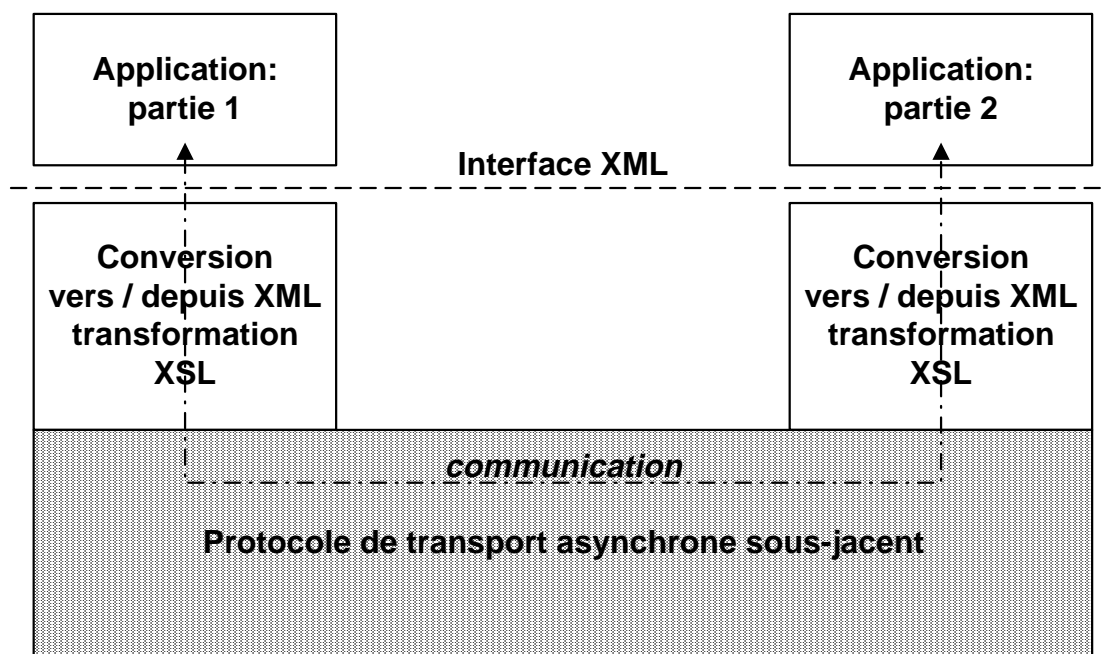
Le couple XML + XSL ouvre des perspectives considérables en matière d'interopérabilité. Il permet de définir des messages auto-descripteurs, permettant d'interfacer deux applications aussi distinctes qu'une facturation et une comptabilité,

et de le faire indépendamment des systèmes d'exploitation, langages de programmation, formats de mise en page et protocoles réseau. Qui plus est, chaque application qui utilise des données en comprend la structure et la sémantique, contrairement à HTML: cela permet des architectures puissantes. XML permet à un client d'échanger des données avec de multiples sources ou serveurs, sans que le client connaisse à l'avance les formats et les types de données auxquelles il accédera.

XSL permet, au départ ou à l'arrivée d'un message, de définir et d'exécuter des transformations automatiques de données. De telles transformations peuvent servir à adapter les données d'une application émettrice A aux besoins d'une application réceptrice B.

Le mécanisme de transformation de données de XSL ressemble au mécanisme de traduction de langage utilisé par un compilateur; voici comment on peut réaliser une conversion de données entre deux applications A et B:

- un analyseur syntaxique reconnaît des éléments dans le texte XML provenant de A
- il leur applique des règles de transformation, en générant du texte XML avec une structure arbitraire, définie en XSL par l'utilisateur; la traduction peut prendre en compte des valeurs de données du texte source, et pas seulement la structure; la logique de prise en compte peut comprendre des conditions et des boucles, par exemple telles qu'elles existent en JavaScript; le résultat de la traduction s'appelle un "arbre résultat" (results tree)
- le résultat de la traduction (du XML) est passé au mécanisme d'entrée de B, qui l'analyse et le transforme si nécessaire pour qu'il convienne à B.



### Interopérabilité XML avec transformations XSL

En résumé, XML + XSL est une solution d'interopérabilité permettant de créer des passerelles de données entre une application A et le format XML, même pour des structures de données fort complexes. Une telle passerelle peut:

- extraire des données de A et créer un document XML, ou
- charger un document XML créé par ailleurs dans A

La passerelle peut, en outre, filtrer et transformer des documents XML grâce à des opérations décrites en XSL, pour adapter son contenu aux besoins d'une autre application.

On parviendra ainsi à interfacier des progiciels d'application, des SGBD, des documents EDI, des data warehouses, des référentiels, etc. On pourra aussi construire des filtres puissants et souples pour bloquer les messages email ou faire des réponses automatiques, filtrer des données ou des transactions avant un processus de réplication, etc.

### **2.6.1 Intégration des applications: synchrone ou asynchrone?**

En principe, la coopération des applications se fera par messages asynchrones, mais étant donné la vitesse considérable d'interprétation et de transformation de XML/XSL on peut même l'envisager de manière synchrone, avec une session maintenue.

En outre, le fait de stocker des données XML provenant d'une application avant de les utiliser dans une autre application permet à chaque application de travailler à son rythme: par exemple, A peut produire des données à chaque transaction, plusieurs fois par seconde, alors que B consomme ces données une seule fois par jour. Cette approche est d'autant plus utile que l'intégration concerne davantage d'applications. Voir aussi, à ce propos, le paragraphe "Serveurs de données XML" ci-dessous.

## **2.7 Accès à des documents XML par programme: standard DOM**

Pour un programme orienté-objets, un document XML est un véritable objet, et ses éléments sont eux-mêmes des objets. Tous ces objets sont accessibles par programme. Le standard DOM (Document Object Model) définit les méthodes d'accès à ces objets et une API pour le langage Java. En tant que programme orienté-objets, l'interpréteur XML offre une API avec des méthodes permettant de parcourir le document, de rechercher un élément, d'ajouter, supprimer ou modifier des éléments, etc. Pour un programme accédant à cette API, l'interpréteur XML est un objet passerelle permettant d'accéder à un autre objet, le document, dont les éléments sont eux-mêmes des objets. Par conséquent, si on désire utiliser un langage orienté-objets pour les opérations de filtrage et de transformation des documents XML, c'est désormais possible à partir de Java et de Visual Basic, qui offrent tous deux des classes d'accès.

Cela signifie que les opérations de conversion sur un document XML peuvent être effectuées sous le contrôle d'un programme Java ou Visual Basic, lorsque la transformation XSL n'est pas retenue, par exemple parce qu'on la juge insuffisamment puissante. Un interpréteur XML confère donc aux documents qu'il gère un caractère dynamique.

Notons que le même modèle DOM définit aussi l'accès aux éléments d'un document HTML: si on dispose d'un interpréteur HTML supportant un accès par programme semblable à celui d'un interpréteur XML, on peut dynamiquement, dans le client, modifier le contenu et la mise en page du document HTML. On dit alors qu'on utilise DHTML (Dynamic HTML). Chez Microsoft, Internet Explorer supporte DHTML et XML conformément au modèle DOM. Par analogie, XML pourrait aussi s'appeler "Dynamic XML".

### **2.7.1 XML et Java**

Le langage Java offre la portabilité du code, XML la portabilité des données: les deux sont donc complémentaires. On accédera donc à des objets XML (documents et leurs éléments) portables à l'aide de classes Java portables elles aussi. Le slogan Java "Write Once, Run Everywhere" peut s'étendre à XML sous l'une des deux formes "Write Once, Store Everywhere" ou "Write Once, Publish Everywhere".

Sun et des partenaires travaillent à définir un ensemble de classes Java d'accès à des objets DOM / XML constituant une extension de la bibliothèque standard du langage. Le projet correspondant s'appelle "Project X". Sun et plusieurs autres éditeurs offriront des interpréteurs XML écrits en Java et offrant une API appelée "SAX" (Simple API to XML). Cette API, déjà spécifiée, est conforme au modèle DOM. L'objectif de Sun et de ses partenaires est de permettre la création de JavaBeans et de Enterprise JavaBeans standards permettant l'accès à des documents et messages XML.

### **2.7.2 Une alternative à la génération dynamique de pages dans le serveur**

Avec leur comportement dynamique, HTML et XML peuvent être utilisés dans un client pour personnaliser le comportement en fonction de saisies de l'utilisateur et de la logique applicative. On peut ainsi, par exemple, ne faire apparaître un texte que lorsque c'est nécessaire; un format d'affichage peut mettre en évidence un élément particulier. La personnalisation du comportement dans le client est une alternative à la génération dynamique de pages sur-mesure dans le serveur; elle apporte souvent une moindre charge réseau et de meilleurs temps de réponse.

### **2.7.3 Utilisation de langages de script**

Le standard DOM n'exige pas un langage de la puissance de Java pour accéder aux documents: un langage de script comme JavaScript ou VBScript peut aussi offrir les classes nécessaires. Le JavaScript de Microsoft, appelé parfois JScript, est conforme aux recommandations du standard ECMAScript (ECMA-262) et supporte l'API Java du Document Object Model. Internet Explorer 5 contient un interpréteur JavaScript complet.

## 2.8 Resource Description Framework (RDF)

L'organisme de standardisation W3C (World Wide Web Consortium) a publié le 24/2/99 une recommandation officielle appelée RDF. Celle-ci décrit un modèle et une spécification syntaxique *pour utiliser des métadonnées*.

Les métadonnées de RDF sont des "données qui décrivent des ressources Web". Exemples: "Titre", "Auteur" ou "Editeur". RDF permet l'interopérabilité entre des applications qui échangent des données dont elles comprennent la sémantique. RDF est utile dans des applications comme:

- la découverte de ressources par des moteurs de recherche
- la création et l'exploration de catalogues décrivant des documents stockés en format XML, avec leurs hyperliens.

RDF utilise XML pour décrire les données. On peut le considérer comme un complément de XML permettant à des applications de comprendre quelles données sont disponibles sans que les créateurs de ces applications se soient mis d'accord au préalable (caractère auto-descriptif).

## 2.9 Différences entre XML+XSL et d'autres solutions d'interopérabilité

### 2.9.1 L'interopérabilité limitée des protocoles client-serveur classiques

Dans l'architecture client-serveur traditionnelle, le modèle de communication fait que le client envoie une demande de service à laquelle le serveur répond. Exemples de protocoles d'interopérabilité client-serveur:

- Un protocole d'accès à un SGBD relationnel, comme Net8 de Oracle ou ODBC; ce type de protocole supporte *les fonctions* nécessaires à l'exécution d'instructions SQL.
- Un protocole d'appel de fonctions, comme les divers RPC (il n'y a pas de standard officiel RPC), CORBA ou DCOM; ce type de protocole peut faire communiquer des programmes d'application entre eux, des objets applicatifs avec un SGBD orienté-objets ou des objets applicatifs entre eux.
- Le protocole de transfert de fichiers FTP, File Transfer Protocol, qui fonctionne en mode client-serveur.

A part FTP, *les autres protocoles transfèrent des appels de fonctions, pas des données*. Ils ont été conçus pour des applications non extensibles, où un client coopère avec un seul serveur ou un nombre limité de serveurs. Ils ne sont pas destinés à des clients qui veulent coopérer avec n'importe quel type de serveur, car ces protocoles orientés-fonctions sont définis pour un ensemble de fonctions non évolutif. FTP, lui, peut transférer des fichiers entre n'importe quel client FTP et n'importe quel serveur FTP, mais son ensemble non extensible de services limite son utilisation à un seul type de service: le transfert de fichiers.

## 2.9.2 Interopérabilité nécessaire à une coopération ouverte

De telles restrictions sont acceptables dans un environnement de production stable, dont les applications ont été prévues pour fonctionner longtemps sans évolution. Elles sont inacceptables lorsqu'un client veut accéder sans programmation sur-mesure à n'importe quel service susceptible de l'intéresser: ce type de *coopération ouverte* demande à l'évidence *un protocole qui ne transporte que des données*, sans fonctions prédéfinies autres que celles strictement nécessaires à ce transport; les serveurs correspondants ne doivent pas permettre un accès direct à des fonctions autres que l'accès aux données, et ces données doivent avoir une structure universellement compréhensible, donc auto-descriptive.

Un exemple permet de mieux comprendre la difficulté d'assurer l'interopérabilité ouverte en passant par des fonctions, celui de deux progiciels puissants que l'on voudrait faire coopérer. Un progiciel de logistique comme SAP offre des API et fonctions adaptées à la logistique, et un progiciel de Ressources Humaines comme PeopleSoft des API et fonctions adaptées à la gestion du personnel. Si on veut faire coopérer SAP et PeopleSoft, par exemple pour tirer les conséquences sur le personnel d'un plan de charge industriel, il faut écrire une application *sur-mesure* qui appelle les fonctions de deux API, ce qui est difficile, coûteux et peu évolutif.

## 2.9.3 Avantages de XML pour la coopération ouverte entre applications

*XML est un standard de données.* En tant que tel il peut être utilisé avec de nombreux modèles de communication en plus du client-serveur (qui peut transporter des données XML dans les arguments de ses appels de fonction). On peut, par exemple, utiliser des données XML avec des communications de type publier-s'abonner.

XML apporte ici une ouverture considérable, impossible à obtenir avec les protocoles client-serveur. C'est qu'un protocole est fait, par définition, pour supporter certains services et pas d'autres, certaines données et pas d'autres. Un échange XML transfère des données alors qu'un échange RPC, CORBA, ou DCOM transfère des appels de fonctions: ce n'est pas la même chose.

Dans l'exemple ci-dessus on pourrait réaliser une interopérabilité ouverte en commençant par des passerelles standard SAP-XML et PeopleSoft-XML créées par les éditeurs correspondants (ce qui est en train de se faire). On extrairait donc, en XML, des données de l'un des progiciels pour les remettre à l'autre, avec une éventuelle opération de conversion au passage. La logique d'extraction doit pouvoir être rédigée indépendamment de la logique de conversion, ce qui est plus simple et plus évolutif. Si l'API de l'un des progiciels évolue l'autre ne s'en aperçoit pas, car il continue à voir les mêmes données d'échange en XML.

## Conclusions sur l'interopérabilité ouverte

- XML - ou un standard *de données* comparable - est la seule approche possible pour une coopération ouverte entre applications.

- L'interopérabilité client-serveur traditionnelle convient pour une coopération non évolutive, pas pour une coopération ouverte. Cette dernière est gênée par la liste figée des services d'un protocole, qui limite aussi la liste des serveurs prêts à les fournir.

## 2.9.4 Autres différences d'interopérabilité entre XML et client-serveur

### Modèle de données

Le modèle relationnel de données, le plus utilisé en client-serveur, est basé sur des tables indépendantes. Ce modèle est bien moins puissant que celui de XML, dont il est un sous-ensemble. Le modèle de XML, une hiérarchie avec des hyperliens, est proche du modèle objet d'un SGBD orienté-objets.

### Transparence par rapport aux structures de données

Une coopération entre applications par appels de fonctions RPC, CORBA ou DCOM nécessite des interfaces prédéfinies partagées par le client et le serveur. Ces interfaces ne peuvent évoluer que par versions successives, ce qui n'est pas assez souple pour une coopération ouverte. Les messages XML, au contraire, sont auto-descripteurs. Leur structure lisible ne nécessite pas de définir les fonctions applicatives avant d'échanger des données; chaque nouvel échange peut utiliser une nouvelle structure de données.

### Transparence par rapport au format des données

L'interopérabilité FTP, RPC, CORBA ou DCOM est *binaire* (chaque programme voit les données dans son format natif, le protocole assurant les conversions éventuellement nécessaires). Au contraire, XML est non binaire: une application qui s'en sert doit utiliser des données au format standard XML pour sa communication. Aujourd'hui, le standard XML ne supporte que les données ASCII 7 bits, mais les extensions en cours d'approbation permettront de supporter en 1999 des types de données bien plus riches.

### Langages de programmation et compétences

Les protocoles ODBC, RPC, CORBA et DCOM sont techniques et faits pour utilisation par des programmeurs dans des programmes. XML, au contraire, est à la portée de personnel techniquement moins pointu, vite formé et utilisant un simple éditeur.

### Stockage et transformation des données

RPC, CORBA et DCOM n'ont rien à voir avec le stockage des données sur disque et leur transformation éventuelle, alors que ce sont là deux domaines d'utilisation importants de XML.

### Sécurité et firewalls (pare-feux)

Au sens de la sécurité, HTML et XML traversent les firewalls, car ils aboutissent à un interpréteur qui protège du code binaire éventuellement dangereux. CORBA et DCOM sont en général arrêtés par un firewall; pour leur permettre de passer, on

peut les transporter par dessus HTTP, mais avec une perte de performance sensible.

### **Standard / propriétaire**

XML et CORBA sont des standards officiels. COM et ODBC sont des technologies propriétaires de Microsoft supportées seulement par des logiciels propriétaires. Il y a de nombreuses offres ODBC chez de nombreux éditeurs, mais leur compatibilité laisse à désirer. Il n'y a pas de protocole standard d'accès client-serveur à un SGBD relationnel: les protocoles de ce type sont propriétaires, mais tous les SGBD supportent une version de ODBC. Il y a plusieurs RPC, tous propriétaires.

#### **2.9.5 Recommandations de choix entre client-serveur et XML**

- Utiliser une architecture client-serveur lorsque la coopération entre clients et serveurs peut être définie à l'aide de fonctions, et que celles-ci sont stables.
- Utiliser XML pour structurer des données échangées entre applications (en général avec une conversion) lorsque les applications ont des API et des services trop différents pour supporter une interopérabilité par appels de fonction.

Lorsqu'il faut intégrer plus de deux applications hétérogènes il est probable que l'approche XML sera indispensable.

Le stockage des données intermédiaires XML peut être utilisé pour une intégration non-synchrone comme pour une persistance qui a une valeur en elle-même.

#### **2.9.6 Comparaison de XML avec EDI**

XML est une solution de stockage et d'interrogation de données, EDI non.

XML est parfaitement adapté à un accès aux données par l'intermédiaire du Web, EDI non.

XML est plus souple et plus facile à mettre en oeuvre que EDI, mais il ne dispense pas, lui non plus, de se mettre d'accord sur les données à transmettre! Avec XML, on se mettra d'accord sur les descriptions DTD des documents ou messages pour l'activité concernée.

#### **2.9.7 XML a souvent besoin de HTML pour l'affichage**

Lorsqu'il faut afficher les données XML reçues par un client pour qu'un utilisateur les voie on a besoin d'un standard universel d'affichage, parce que XML ne décrit pas implicitement le format à utiliser: il en est même indépendant.

Il se peut que XSL remplace un jour HTML+CSS pour la mise en page. Mais en attendant, pour afficher du texte XML avec un mécanisme standard on le convertit en HTML. XSL décrit alors la mise en page de chaque élément XML affichable. Microsoft a besoin d'une telle conversion dans Internet Explorer 5 et OFFICE 2000: les données WORD 2000, EXCEL 2000 et POWERPOINT 200 sont représentées et transportées sur le Web en format HTML +XML, et la conversion vers HTML + CSS

nécessaire à un affichage fidèle dans un navigateur est effectuée par la partie XSL de l'interpréteur XML.

Bien entendu, une application n'a pas besoin d'un mécanisme d'affichage *standard*: elle peut formater du XML comme bon lui semble.

## 2.10 Deux langages complémentaires de XML: XQL et XLL

### 2.10.1 Langage XQL (eXtended Query Language)

En plus de XSL, un autre futur standard est en cours d'étude: XQL (eXtended Query Language). XQL permet la recherche dans une arborescence XML. C'est une extension des possibilités de reconnaissance de structure (pattern-matching) de XSL permettant, comme XSL, de décrire des critères de recherche. Exemple: "book/author" signifie "trouver les éléments "author" contenus dans des éléments "book", c'est-à-dire une arborescence dont la branche "book" contient une branche "author". Les recherches concernent une partie de structure à reconnaître et dont certains éléments satisfont peut-être, en plus, des critères de valeur. Avec XQL on peut définir des hyperliens vers tous les noeuds qui satisfont certains critères. La chaîne de caractères d'une requête XQL peut être incluse dans un URL. XQL est déclaratif, comme SQL. Le résultat d'une recherche XQL est une arborescence ou un graphe XML.

### 2.10.2 Langage XLL (eXtended Link Language)

Autre standard en cours d'étude, XLL définit des hyperliens puissants pour les documents XML. Un lien peut être:

- *externe*: il pointe alors vers un objet autre que le document d'origine
- *interne*: il pointe vers un élément du même document.

La spécification de travail actuelle langage XLL a deux parties:

- *XLink*, qui spécifie des constructions syntaxiques que l'on peut insérer dans des documents XML pour décrire des liens entre objets. Un lien, ici, est une relation explicite entre deux objets de données ou portions d'objets. XLink utilise une syntaxe XML pour créer des structures capables de décrire les liens unidirectionnels simples, du type utilisé dans HTML aujourd'hui, aussi bien que des liens à terminaisons multiples.
- *XPointer*, destiné à être utilisé avec XLink. La spécification de travail actuelle définit des constructions syntaxiques qui supportent l'adressage dans les structures internes de documents XML. On peut ainsi désigner des éléments, des chaînes de caractères et d'autres parties de documents XML, que ceux-ci contiennent ou non un attribut particulier d'identification.

Exemples d'hyperliens définis par XLL:

- lien bidirectionnel;
- liens à destinations multiples;

- liens typés (à sémantique précise);
- liens d'inclusion ou de remplacement de contenu à partir d'un autre document;
- bases de données de liens: actuellement, un lien HTML utilise une adresse fixe de système et de fichier pour désigner un objet. Avec XLL, de telles adresses pourront être stockées dans une base de liens, notamment sous forme logique et indirecte. Cela facilitera la maintenance des liens lorsqu'un objet change d'adresse.

## 2.11 Résumé sur les avantages de XML

XML apporte:

- La possibilité de *structurer* des documents quelque soit leur complexité
- *L'extensibilité*, en définissant toutes les balises ("tags") dont on a besoin
- *La validation*, permettant de vérifier la conformité de la syntaxe (donc de la structure) d'un document par rapport à la grammaire générale du langage XML et, éventuellement, à celle du DTD de ce type de documents si il existe
- *L'indépendance par rapport aux formats de présentation*
- *L'indépendance par rapport aux systèmes d'exploitation* qui stockent les données XML ou qui y accèdent
- *Des recherches de puissantes*, associant reconnaissance de valeurs (comme le relationnel) et reconnaissance de structures arborescentes
- La possibilité de *transformer les données* par des techniques de type compilation
- *L'accès aux données et le transfert de messages par le Web*
- La possibilité de créer des *mécanismes d'interopérabilité* entre applications qui sont particulièrement ouverts
- *Le support de types de liens nouveaux et puissants*, grâce à XLL.

## 2.12 Site XML francophone

Un site XML francophone intéressant, à partir duquel on peut aussi trouver les sites anglophones est à l'adresse: <http://www.chez.com/xml>.

## 3 Serveurs de données XML

### 3.1 Pourquoi des serveurs XML

Lorsque l'interopérabilité XML est utilisée de manière asynchrone, ce qui est fréquent, il faut un mécanisme de stockage des données en format XML. Outre l'interopérabilité, ce stockage de données en format indépendant des plates-formes est intéressant en lui-même. Un SGBD pour documents XML est donc extrêmement utile, tant pour ses fonctions de stockage de documents que pour son rôle de middleware. Un tel SGBD tournera dans un serveur appelé "serveur XML".

En tant que middleware, XML est particulièrement bien adapté à un rôle de passerelle entre une application et des données *non conçus pour l'Internet* et un *accès par navigateur*. Internet Explorer 4 et 5 (et bientôt Netscape Communicator) supportent directement XML, qu'ils affichent après traduction en HTML. Et il est facile d'interfacer un serveur Web avec serveur XML.

Les avantages d'une passerelle XML interfaçant une application existante avec l'Internet sont multiples:

- pas d'impact sur l'application existante, qui n'est nullement modifiée
- les formats de présentation et saisie sur le Web des données et transactions de l'application sont indépendants de cette application, d'où une bonne souplesse d'évolution
- le caractère asynchrone de la passerelle/serveur XML protège l'application des pics de demandes, qui sont automatiquement étalés dans le temps
- la fonction de cache de données du serveur XML (voir ci-dessous) améliore la performance d'accès aux données XML.

### 3.2 SGBD XML oui, mais orienté-objets

Pour qu'un serveur XML et son SGBD offrent des fonctionnalités riches (puissance de recherche, gestion des conflits d'accès, atomicité des transactions, sécurité, etc.) il faut d'abord qu'il puisse gérer une structure arborescente. Disons-le tout net: un SGBD relationnel (en abrégé: SGBDR) est mal adapté à ce rôle, car il devrait mettre en oeuvre de nombreuses tables et d'innombrables jointures. Un SGBD orienté-objets (en abrégé: SGBDO), au contraire, convient parfaitement comme "moteur de stockage XML".

Un serveur de données XML *moderne* doit être adapté au traitement de requêtes émanant de programmes orientés-objets. Les seuls SGBD construits dès le départ comme solution de persistance pour programmes orientés-objets sont les SGBD orientés-objets. Ils offrent une solution de persistance à la fois *plus élégante, plus facile à utiliser et plus performante pour manipuler des données complexes* qu'un SGBD relationnel ou un SGBD relationnel muni d'une extension objets.

### 3.3 Ce qu'on attend d'un SGBD pour serveur XML

Comme tout SGBD, un SGBD XML doit pouvoir *stocker des données* et *permettre d'y accéder*. Les données XML étant arborescentes (structure très générale, dont la table relationnelle et les champs de bits longs ou "BLOBS" ne sont que des cas particuliers) le SGBD doit permettre de définir et de gérer ce type de structure. Voyons en détail les besoins à satisfaire dans ces domaines.

#### 3.3.1 Gestion de tous types et structures de données

- Objets avec données traditionnelles et sous-objets en arborescence infinie: un objet, au sens programmation orientée-objets, peut contenir des données classiques (nombres entiers et flottants, chaînes de caractères, etc.), des tableaux et d'autres objets, eux-mêmes composites. Un SGBD XML doit pouvoir définir, stocker et manipuler de tels objets.
- Tous types fonctionnels de données: tables relationnelles, HTML, texte ASCII, texte WORD ou PDF, schémas et dessins techniques, emails, photos, son et vidéo, BLOBS (chaînes de bits parfois très longues), etc.

#### 3.3.2 Mécanismes de stockage (persistance)

- Stockage de longue durée ou de courte durée (cache) également performants
- Intégration de données de diverses sources (mainframes, UNIX, OFFICE, etc.) *en un même format XML et un même lieu, la base XML*
- Structures de données adaptées à l'utilisation par des programmes orientés-objets, qui doivent les voir comme si elles étaient résidentes en mémoire. Il s'agit ici d'éviter les difficiles conversions nécessaires entre des données stockées selon un modèle de tables relationnelles et des programmes qui les voient selon un modèle objet. Avec un SGBDO les données sont vues et manipulées selon le même modèle, le modèle objets. En outre (et contrairement aux SGBD dits "universels" qui n'acceptent que certaines structures d'objets) un SGBDO accepte de manière native n'importe quelle structure d'objet définissable dans l'un des langages qu'il supporte: C++, Java, etc.
- Pour accès rapide aux données: indexation et groupage physique des objets fils d'un même objet dans la même page que l'objet père; cela évite, par exemple, les coûteuses jointures d'un SGBDR.
- Journalisation (logging) pour protection en cas d'incident, comme tout SGBD
- Souplesse d'évolution de la structure des données, bien plus que le relationnel. Un SGBDR est fait pour traiter de nombreuses fois (nombreuses transactions) des données à structure stable. Lorsqu'on veut modifier une structure (ajout de tables ou de colonnes à une table, modification d'un format de colonne) il faut mettre à jour le dictionnaire de données et effectuer une réorganisation physique qui prend du temps. Un SGBDO, au contraire, se joue de ces difficultés: la structure des données de sa base est dans la logique du code des applications clientes. Lorsque l'application est prête à traiter la nouvelle structure de données, la base est prête à

supporter cette structure. Avec un SGBDO, l'évolution d'un schéma de données est simple, d'où une grande souplesse pour s'adapter aux besoins d'une entreprise.

### 3.3.3 Conversion de données vers / depuis XML

Il faut pouvoir accepter les données telles qu'elles sont dans leur "source de données", par exemple sous forme de message email, de document WORD ou de base relationnelle, et les convertir en format XML pour stockage et/ou accès par des applications. On trouve donc chez divers éditeurs:

- soit des passerelles de conversion directe de leurs données propriétaires en XML
- soit des classes d'accès à ces données comme les classes OLE DB de Microsoft, qu'une application peut utiliser pour lire les données, en effectuant elle-même la conversion en XML.

La solution au besoin inverse (à partir d'un arbre XML générer des données dans un format quelconque) est en général fournie sous forme de passerelles propriétaires, comme celles permettant de charger des bases de données pour applications SAP ou PeopleSoft à partir de texte XML.

De toute manière, pour utiliser des données en format XML il faut disposer:

- d'un analyseur syntaxique XML
- d'un mécanisme qui stocke les données de sortie de l'analyse précédente dans une base de données capable de les recevoir.

Le serveur XML eXcelon, décrit ci-dessous, offre ces deux fonctions. Il supporte en outre les accès suivants:

- accès à partir d'un navigateur sans programmation
- interface Java
- conformité au modèle DOM (Document Object Model)
- Microsoft COM (Component Object Model) et OLE DB.

### 3.3.4 Modes d'accès aux données et interfaces correspondantes

- Des programmes orientés-objets COM ou CORBA doivent pouvoir utiliser ces mécanismes d'interopérabilité (localement ou à distance) pour accéder à une base XML. Cet accès sera possible seulement si toute la base (et tout sous-ensemble de la base) est visible pour ces programmes en tant qu'objets COM ou CORBA. En pratique, il faut que le serveur XML soit conforme au Document Object Model (DOM) décrit précédemment. On pourra alors, par exemple, accéder directement à des éléments XML arborescents du serveur à partir de pages actives ASP Microsoft, qu'elles soient interprétées dans le serveur ou dans le client.
- API conforme au modèle DOM pour accès direct à la base XML par des programmes orientés-objets en C++ et Java.
- Classes passerelles d'accès depuis des scripts VBScript, JavaScript, etc. ou depuis des langages comme C++, Visual Basic ou Java.

- Import et export de données sans programmation: en envoyant un ordre au serveur XML on doit pouvoir lui faire charger des données dans sa base à partir d'un format usuel (comme le texte ASCII avec délimiteurs) ou extraire des données de sa base XML et les fournir dans un format usuel.
- Accès à partir d'un navigateur: le serveur XML doit pouvoir recevoir des URL et générer du texte HTML en retour ou mettre à jour sa base XML.
- Accès à des SGBD traditionnels: le serveur XML doit supporter soit l'API native du SGBD, soit ODBC, soit OLE DB, etc.
- Accès à des transactions, telles qu'elles sont supportées par un moniteur transactionnel comme CICS ou un AS/400. Cet accès est effectué à travers des classes fournies par l'éditeur du logiciel transactionnel. Ces classes passerelles sont alors utilisées par un module d'extension du serveur XML, qui effectue les accès et conversions nécessaires.
- Accès à des objets ou fonctions OFFICE (WORD, PowerPoint, EXCEL) et des messages ou fonctions EXCHANGE. Cet accès est possible soit à travers l'interface COM des progiciels correspondants, soit à travers des passerelles OLE DB fournies par Microsoft.

### 3.3.5 Mécanismes transactionnels

L'accès aux données XML peut avoir lieu dans un contexte transactionnel, où il faut préserver l'intégrité des données. Une transaction doit notamment être effectuée complètement ou pas du tout (Atomicité), même si elle met en jeu plusieurs serveurs. Il faut, d'autre part, offrir des services de Cohérence, Isolation multiutilisateurs et Durée (persistance): on dit qu'une transaction doit bénéficier de services ACID. Un serveur XML doit donc supporter ces mécanismes, exactement comme tout autre SGBD.

### 3.3.6 Performance

La performance comprend deux notions: le temps de réponse et l'aptitude à monter en charge ("scalability"). Le temps de réponse implique, dans le cas d'un serveur XML:

- La présence d'un analyseur syntaxique XML ("parser") rapide
- Des caches client et serveur, dont l'alimentation à partir de la base de données et la synchronisation sont automatiques pour des raisons de simplicité d'emploi et de cohérence des données.

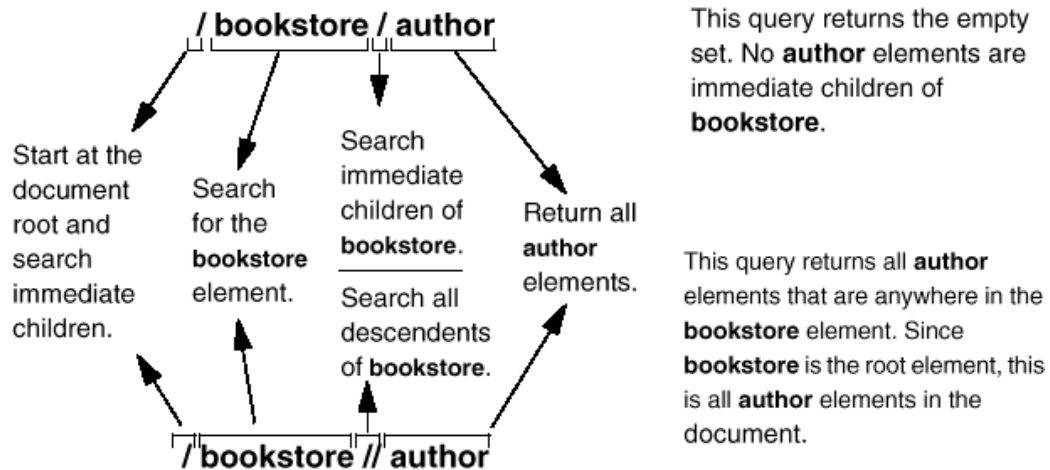
La montée en charge implique les mêmes techniques de performance qu'un SGBDR (fast commit, group commit, parallélisme, threads, support SMP, etc.) Ces techniques sont toutes mises en oeuvre dans le SGBDO ObjectStore, qui est à la base du serveur XML eXcelon, décrit ci-dessous.

### 3.3.7 Recherche de données et XQL

(Voir section sur XQL dans le texte précédent). Les besoins sont:

- Support de recherches puissantes sur la base XML

- Reconnaissance de structure (différence avec le relationnel)
- Pas de jointures coûteuses.



## Deux exemples de requêtes XML: noter le rôle des "/"

Exemple d'arborescence renvoyée par eXcelon pour "/bookstore//author":

```
<?xml version="1.0" ?>
- <xlsql:result xmlns:xlsql="http://www.ObjectDesign.com/eXcelon/
namespaces/query"
xmlns:my="http://www.placeholder-name-here.com/schema/">
- <author>
  <first-name>Joe</first-name>
  <last-name>Bob</last-name>
  <award>Trenton Literary Review Honorable Mention</award>
</author>
- <author>
  <first-name>Mary</first-name>
  <last-name>Bob</last-name>
  <publication> Selected Short Stories of <first-name>Mary
  </first-name><last-name>Bob</last-name></publication>
</author>
- <author>
  <first-name>Toni</first-name>
  <last-name>Bob</last-name>
  <degree from="Trenton U">B.A.</degree>
  <degree from="Harvard">Ph.D.</degree>
  <award>Pulizer</award>
  <publication>Still in Trenton</publication>
  <publication>Trenton Forever</publication>
</author>
</xlsql:result>
```

### 3.3.8 Fonctions pour développement (modélisation et codage)

Les besoins sont:

- Disposer d'un outil de définition de structure XML à interface utilisateur graphique
- Support du modèle DOM
- Pouvoir développer des *extensions du serveur* permettant à celui-ci d'avoir des comportements particuliers pour le traitement de certains objets de sa base XML: recherche, transformation, mise à jour, etc. Le SGBD XML muni de ces extensions offre des services nouveaux, adaptés à l'application particulière. Ces extensions doivent avoir des API conformes au modèle DOM. On les écrira de préférence en Java, seul langage dont les API DOM sont standard aujourd'hui.
- Un serveur XML doit convenir comme niveau 2 d'une architecture à 3 niveaux (serveur d'applications ou passerelle d'interopérabilité) ou à 2 niveaux (serveur de données).

### 3.3.9 Conformité aux standards

La nouveauté de la technologie XML fait qu'elle évoluera encore pendant plusieurs années. La meilleure façon, alors, de protéger ses investissements en code développé et en formation est d'adopter un serveur XML qui respecte le plus possible de standards et impose le moins possible d'approches propriétaires.

Les domaines dans lesquels il existe des standards qui concernent un serveur XML sont, aujourd'hui:

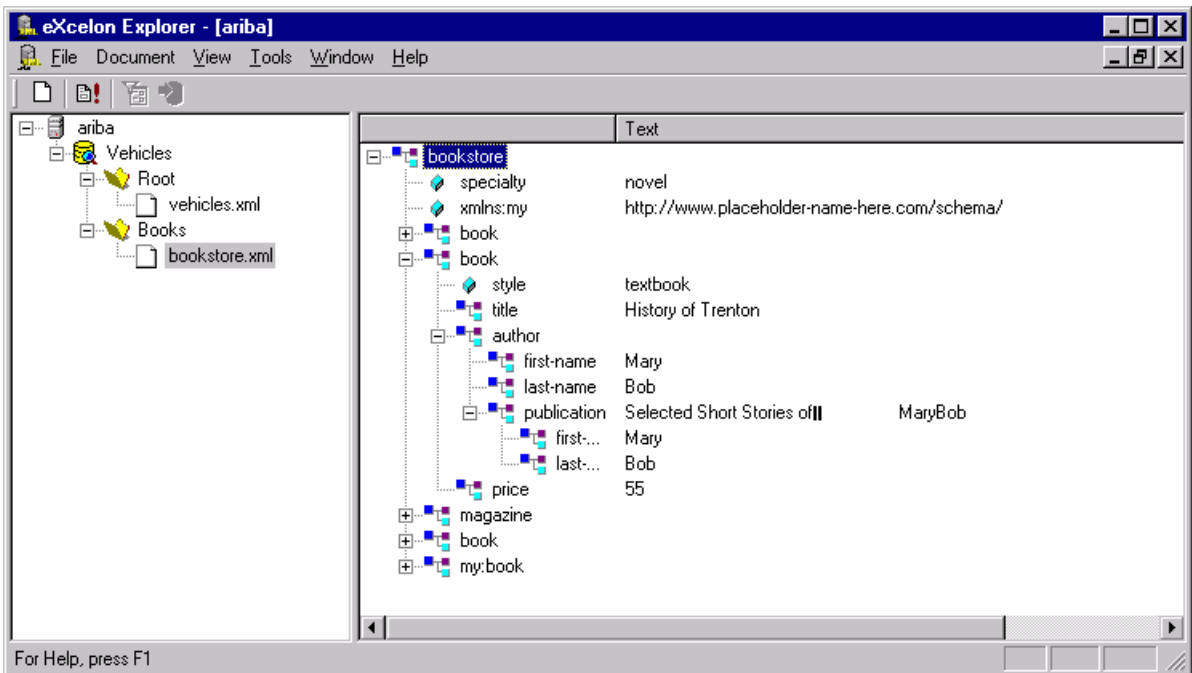
- interface utilisateur graphique: format de mise en page HTML + CSS, services d'interprétation de HTML fournis par un navigateur standard comme Internet Explorer 5
- applications: modèle de données DOM portant sur une structure XML arborescente
- alphabet: UNICODE
- langage: Java (le seul dont l'API DOM est standard aujourd'hui)
- langage de script: ECMAScript (appelé improprement JavaScript ou JScript)
- développement: XMI (standard OMG basé sur XML pour le partage d'informations de conception d'une application) et RDF (Resource Description Framework, modèle et spécification de syntaxe pour les métadonnées)
- interopérabilité: COM, CORBA, HTML.

## 3.4 Exemple de serveur XML: eXcelon de Object Design

Le serveur XML eXcelon, décrit succinctement ci-dessous, répond à l'ensemble des besoins exprimés ci-dessus. En général, il y répond directement, par les logiciels fournis par Object Design; parfois, il y répond grâce à des classes d'accès fournies par des tiers, comme c'est le cas pour accéder à des mainframes.

### 3.4.1 Objectifs et principes de fonctionnement du produit

eXcelon est un serveur de données XML qui stocke celles-ci dans sa base de données, bâtie sur un moteur SGBD orienté-objets ObjectStore. eXcelon reçoit ses données de toutes sortes de sources et les convertit en objets stockés sous forme d'arborescence dans sa base: voir figure ci-dessous, qui montre la représentation arborescente d'un objet "book" et sa représentation XML, tels qu'ils apparaissent dans l'outil d'interface utilisateur Explorer de eXcelon.



```
<book style="textbook">
  <title>History of Trenton</title>
  <author>
    <first-name>Mary</first-name>
    <last-name>Bob</last-name>
    <publication>
      Selected Short Stories of
      <first-name>Mary</first-name>
      <last-name>Bob</last-name>
    </publication>
  </author>
  <price>55</price>
</book>
```

Interface utilisateur Explorer d'eXcelon et l'arborescence XML "book"

eXcelon sert:

- à stocker des données en format universel XML, et à permettre d'y accéder
- de passerelle entre des sources de données quelconques et le format XML; une application accédant à eXcelon ne se préoccupe plus, alors, du format d'origine de ces données: eXcelon centralise et uniformise des données hétérogènes de provenances multiples.

### 3.4.2 Exemples de données d'une base eXcelon

eXcelon n'invente pas la structure des données: il faut la lui fournir sous forme de description XML (DTD). Par contre, eXcelon peut accéder à de nombreux types de données par l'intermédiaire de passerelles:

- Données de SGBD relationnels, accédées en ODBC
- Données de SGBD orienté-objets ObjectStore, accédées de manière native
- Données accessibles à travers la passerelle OLE DB de Microsoft, telles que email Exchange ou Lotus, feuilles de calcul EXCEL, etc.
- Images, graphiques et vidéo
- Transactions mainframe accédées à travers des classes passerelles, etc.

### 3.4.3 Structure de stockage: XMLStore

eXcelon stocke ses données dans des espaces logiques appelés "XMLStores". Un XMLStore est donc la représentation logique d'un ensemble de ressources utilisées par eXcelon pour y stocker des documents (XML ou multimédia) ou des modules logiciels d'extension du serveur.

Physiquement, un XMLStore contient des données XML et/ou des données binaires (BLOBS). Les données XML sont stockées dans un format qui résulte de l'analyse syntaxique (parsing) de leur texte XML.

La structure d'un XMLStore est *arborescente*, comme celle d'un système de fichiers. On le crée comme un fichier, en choisissant la commande "New" dans la fenêtre de l'outil d'administration eXcelon Manager (décrit ci-dessous). Un même serveur eXcelon peut gérer de nombreux XMLStores.

Un XMLStore peut contenir 0, 1 ou plusieurs répertoires (directories) exactement comme un système de fichiers et pour les mêmes raisons: connaître le contenu du XMLStore, accélérer l'accès à ses objets et définir des permissions de sécurité.

Selon le cas, un XMLStore est accessible à partir du serveur où il est stocké ou d'autres serveurs du réseau. En plus de la souplesse d'accès, l'utilisation "en parallèle" de plusieurs serveurs permet une bonne montée en charge. Bien entendu, eXcelon sait alors gérer les éventuels conflits d'accès et même les "étreintes fatales" ("deadlock").

Pour importer des données dans un XMLStore, on peut:

- effectuer un “glisser-déposer” (drag and drop) ou un “copier-coller” (copy and paste) depuis une interface utilisateur graphique comme l’Explorateur de Windows
- importer un fichier d’un système de fichiers
- importer des données depuis une source ODBC, OLE DB ou ADO.

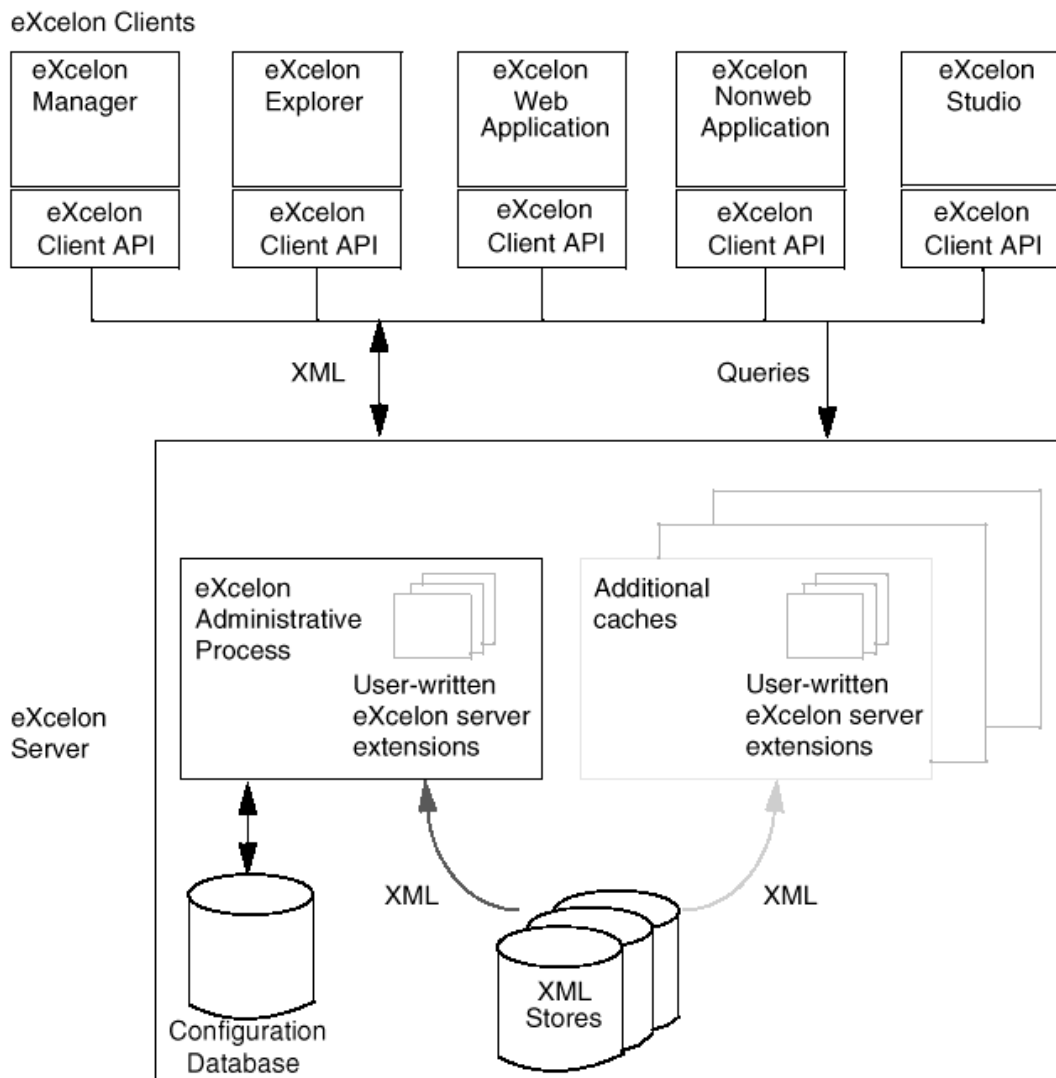
Les données d’un XMLStore sont visualisables et éditables, au sens de leur structure comme dans le détail avec l’Explorateur eXcelon.

La mise en oeuvre de XMLStores est donc très simple et intuitive: avec eXcelon, définir, stocker et visualiser des données est facile.

### 3.4.4 Fonctionnement de eXcelon

#### Principe

(Voir le schéma fonctionnel d'un serveur XML eXcelon ci-dessous)



Architecture fonctionnelle de eXcelon

Une base de données XML est répartie dans un ou plusieurs systèmes serveurs: eXcelon sait gérer des bases réparties. La répartition peut servir:

- avec des données *différentes* dans les divers serveurs: à rapprocher *les données* de leurs utilisateurs. Lorsque les données d'un même serveur sont accédées par de multiples autres serveurs, cela permet:
  - de rapprocher *le traitement* (dans ces autres serveurs) des clients de ces serveurs;
  - de paralléliser l'accès aux données d'un même serveur.

- avec des données *identiques* dans plusieurs serveurs: à organiser une répartition de la charge d'accès entre de multiples serveurs.

## Clients eXcelon

De multiples clients peuvent accéder à un même serveur, les conflits d'accès et l'intégrité transactionnelle étant gérés. Les types de clients utilisables sont:

- *eXcelon Manager*: outil d'administration servant à définir et configurer des XMLStores, installer des extensions du serveur, etc.
- *eXcelon Explorer*: outil d'interface utilisateur graphique d'accès aux XMLStores permettant la recherche, la visualisation et la mise à jour des données XML.
- *Une application Web*, située dans un autre système que le serveur XML ou dans le même système, et communiquant avec le serveur eXcelon en HTML pour accéder aux données en lecture, recherche et mise à jour.
- *Une application orientée-objets quelconque*, communiquant avec le serveur eXcelon:
  - à travers l'API client native de eXcelon
  - à travers une API COM accédant directement à l'interface COM de eXcelon
  - à travers une application intermédiaire tournant dans le serveur et utilisant l'API native eXcelon pour accéder au serveur; la communication avec cette application intermédiaire peut être de n'importe quel type: COM, CORBA, RPC, etc. Un cas particulier intéressant est celui où l'application intermédiaire invoquée par le client est une extension du serveur (voir ci-dessous).
- *eXcelon Studio*: outil de définition des structures XML d'un document sous forme de DCD (Document Content Definition); permet de définir des documents reliés par des liens.

## Serveur eXcelon, caches de données et modules d'extension

Un serveur eXcelon tourne sous Windows NT, et (aujourd'hui) seulement sous NT.

Un serveur eXcelon gère un nombre indéterminé de XMLStores en utilisant des caches de données pour accélérer la performance d'accès. La relation entre XMLStores et caches est de type "N à P":

- plusieurs XMLStores peuvent envoyer des données dans un même cache
  - plusieurs caches peuvent demander des données à un même XMLStore.
- Pour optimiser la performance, on peut définir le nombre de caches que l'on veut et la manière dont ils seront utilisés; on obtient ainsi, par exemple, des économies d'accès disques et un équilibrage des charges.

On peut étendre les possibilités d'un serveur eXcelon en développant des modules applicatifs appelés "extensions du serveur" (server extensions). Un tel module, écrit en Java portable (JDK Sun), manipule les données d'un XMLStore dont il donne une vision DOM.

## Interrogation XQL

L'interrogation en XQL est possible:

- avec une interface utilisateur graphique de formulation des requêtes depuis l'Explorateur d'eXcelon
- en envoyant un URL ou une invocation de méthode contenant une requête à une extension serveur
- à travers l'API client de eXcelon.

La possibilité d'interroger de manière simple et performante une base XML est une des fonctions les plus précieuses de eXcelon.

Enfin, eXcelon offre de puissants mécanismes d'indexation XML permettant de faire des recherches *de structure* en XQL. (Pour l'intérêt de la reconnaissance de structure, impossible saut cas particuliers rares avec un SGBDR, voir le texte général sur XQL au paragraphe "Langage XQL").

### 3.4.5 Performance d'un serveur eXcelon

(Voir paragraphe "Performance" dans la section "Ce qu'on attend d'un SGBD pour serveur XML"). eXcelon offre une performance considérable, en temps de réponse comme en aptitude à monter en charge, car il dispose de tous les mécanismes de performance cités. Des installations en production lourde attestent de cette performance.

### 3.4.6 Sécurité d'un serveur eXcelon

La sécurité d'eXcelon s'appuie sur celle de Windows NT. Elle met en oeuvre les mécanismes de:

- identification d'un utilisateur par nom d'utilisateur et mot de passe pour une session
- propriété d'un fichier ou répertoire système, XMLStore ou répertoire d'un XMLStore
- droits d'accès en lecture, écriture, suppression et exécution
- rôles d'un utilisateur, qui définit un ensemble de permissions qu'il a
- audit des actions de sécurité, en historisant dans un journal (log) les événements qui ont un intérêt au sens sécurité.

## 3.5 Offres de serveurs et outils XML du marché

Standard officiel et solution d'interopérabilité puissante, XML est adopté par un nombre croissant de fournisseurs. Chacun offre une passerelle XML vers / depuis son progiciel.

### 3.5.1 L'offre de Microsoft

Microsoft offre déjà, dans Internet Explorer 4.x, un interpréteur XML incomplet. Un interpréteur complet, conforme à la norme XML 1.0 et tout à fait utilisable, sortira au premier trimestre 1999 avec Internet Explorer 5, et il supportera XSL. Il y aura aussi un interpréteur XML sous forme de dll utilisable dans toute application, MSXML; ce

composant COM fonctionnera dans tout système, client ou serveur, supportant COM. Ces interpréteurs seront indispensables pour OFFICE 2000, successeur de OFFICE 97 mi-99. Avec des outils de Visual Studio, qui les supportera complètement en 99, ces interpréteurs permettront de construire des solutions d'interopérabilité puissantes et performantes. Et si l'on achemine les messages grâce au moniteur asynchrone MSMQ, ces solutions seront particulièrement robustes en cas d'indisponibilité d'un système ou d'un élément de réseau.

### 3.5.2 Autres offres

- Oracle proposera en 1999, avec Oracle 8i, une offre d'interopérabilité XML intéressante. On pourra stocker des documents XML dans la base de données, accéder à ses éléments, effectuer des conversions entre modèles XML et relationnel, effectuer des recherches textuelles ConText sur des documents XML. L'interpréteur XML du noyau du SGBD, écrit en Java, respecte la norme XML 1.0 et offre une API Java. Par rapport à eXcelon, cette offre a les inconvénients suivants:
  - Basée sur le SGBD "Universel" Oracle 8, elle ne profite pas des avantages intrinsèques à un véritable SGBDO comme ObjectStore
  - Elle n'a pas, comme eXcelon, d'interopérabilité directe avec les objets COM de Microsoft, donc avec OLE DB et ActiveX
  - Oracle 8 est plus complexe à administrer et plus gourmand de ressources que ObjectStore
  - Apparue plus tard que eXcelon, elle sera moins mature pendant un certain temps.
- Le SGDB orienté-objets POET a une offre XML. Celle-ci est orientée "dictionnaire de contenu" et destinée à la gestion de bibliothèques de documents XML de toutes tailles, notamment pour les groupes qui collaborent à l'élaboration d'un document. Ce n'est pas un serveur de données XML comme eXcelon.
- WebMethods est un éditeur qui offre des analyseurs syntaxiques XML et des solutions de passerelle d'interopérabilité. Ses outils ne sont pas des serveurs de données comme eXcelon, mais bien plus des solutions d'interopérabilité entre applications. WebMethods est un partenaire de eXcelon, par exemple pour fournir une solution d'interopérabilité entre eXcelon et le progiciel intégré SAP.
- Divers éditeurs d'outils ont annoncé des offres d'interopérabilité XML: Netscape et IBM, par exemple.