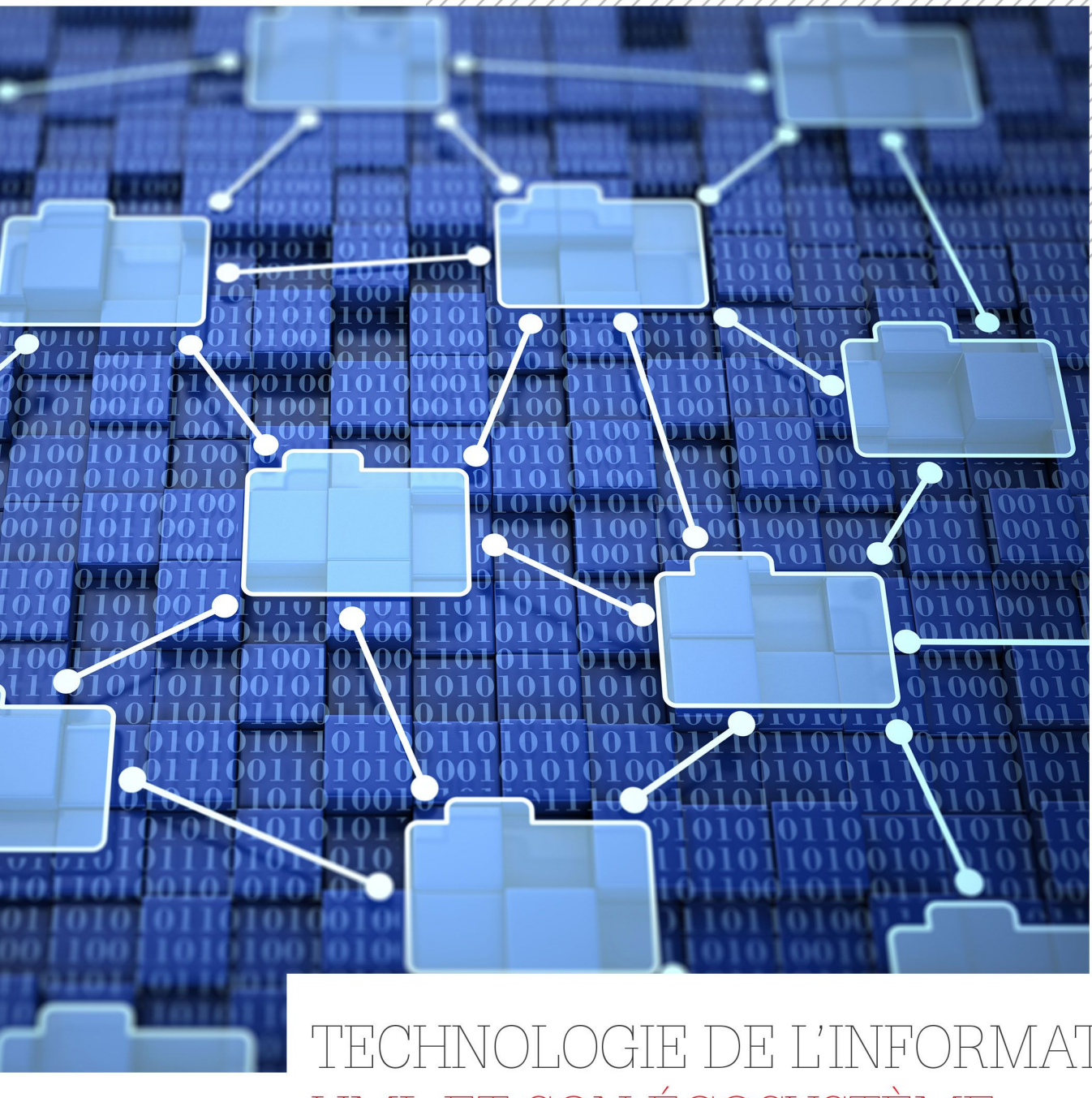
TECHNIQUES
DE L'INGÉNIEUR

LES FOCUS
TECHNIQUES DE L'INGÉNIEUR



TECHNOLOGIE DE L'INFORMATION
XML ET SON ÉCOSYSTÈME

Juin / 2021

TECHNIQUES
DE L'INGÉNIEUR

Réf. : **H3502 V2**

XML et son écosystème

Date de publication :
10 juin 2021

Cet article est issu de : **Technologies de l'information | Documents numériques Gestion de contenu**

par **Gérald KEMBELLEC, Nicolas TRAVERS**

Mots-clés
XML | XPath | XQuery | RDF

Résumé Cet article traite de la structuration de fichiers XML, de la manière de les produire, de les utiliser, de les requêter à travers divers prismes. En effet, après une courte introduction historique sur les causes industrielles et intellectuelles qui ont amené à l'avènement d'XML comme format de stockage de données et d'informations, l'article revient sur les grammaires et vocabulaires qui permettent la structuration et la qualification documentaire dans l'industrie ou la culture. L'article se poursuit par l'application des règles du XML dans la gestion des connaissances et par une incursion dans le Web des données liées. Enfin, l'article présente XML comme structure, vecteur de stockage et de partage de données : il explore le potentiel d'XML comme base de données, les méthodes de requêtage, d'échange et de flux de données.

Keywords
XML | XPath | XQuery | RDF

Abstract This article deals with the structuring of XML files, how to produce, use, and to query them through various prisms. After a short historical introduction on the industrial and intellectual causes that led to the rise of XML as a data and information storage format. Then the article introduces to the grammars and vocabularies that allow the structuring and documentary qualification in industry or culture. The article continues by applying the rules of XML in knowledge management and by an incursion into the Web of linked data. Finally, the article presents XML as a structure, a vector for storing and sharing data: it explores the potential of XML as a database, methods for querying, exchanging and flow of data.

Pour toute question :
Service Relation clientèle
Techniques de l'Ingénieur
Immeuble Pleyad 1
39, boulevard Ornano
93288 Saint-Denis Cedex

Par mail :
infos.clients@teching.com
Par téléphone :
00 33 (0)1 53 35 20 20

XML et son écosystème

par **Gérald KEMBELLEC**

Chargé de recherche au Centre Historique Allemand, Département des Humanités Numériques
Maître de conférences au CNAM en détachement
Laboratoire Dicen-IdF (EA 7339) / Thématique(s) de recherche : Data, médiation, valorisation
Paris, France

et **Nicolas TRAVERS**

Enseignant-Chercheur au Léonard de Vinci Research Center, Pôle Universitaire
Maître de conférences au CNAM en détachement
Laboratoire DVRC / Pôle Universitaire, Paris, France
Chercheur associé au laboratoire CEDRIC / CNAM, Paris, France

Note de l'éditeur

Cet article remplace l'article intitulé XML : gestion de contenu, de Sylvie Calabretto et Tiphaine Accary, paru chez les Techniques de l'Ingénieurie en 2006

L'apparition des systèmes d'information dans l'industrie a conduit à la création des langages informatiques pour stocker des données au sein de dispositifs et créer de l'information, mais aussi décrire cette dernière et effectuer des computations. Enfin, d'autres langages furent inventés pour présenter les données ou les informations qui en sont issues. Dans cet article, nous présentons une focale sur un langage en particulier, le XML, qui offre la particularité de

se décliner pour répondre à la plupart des besoins cités – hors la programmation. Le XML et ses dialectes offrent la possibilité de stocker, décrire, filtrer, interroger et présenter les contenus. De plus, il s'agit d'un langage libre et ouvert, ne nécessitant pas de logiciel spécifique pour son utilisation.

Nous présentons ici un aspect historique expliquant l'apparition de XML, ses évolutions et sans être exhaustif, les différents cadres d'usage. Cet article est illustré par de nombreux exemples techniques pour mieux appréhender les dialectes les plus représentatifs de ce langage.

1. Évolution conceptuelle des langages documentaires

Le temps des grands explorateurs, celui de l'imprimerie et plus tard celui de la révolution industrielle ont grandement modifié le rapport de l'humain à son environnement. Les distances se sont raccourcies dans la perception du commun, le savoir s'est démocratisé, puis industrialisé, n'étant plus le fait d'une élite de copistes lettrés garants de l'inscription et de la transmission des écritures saintes et doctes ou encore de celle des chartes. Le XX^{e} siècle a amené bon nombre de modifications dans la relation de l'homme à l'écrit, avec une volonté de structurer l'information et de la formaliser par un ancrage dans des inscriptions compréhensibles à la fois par les humains, mais aussi par les machines mécaniques puis électroniques, nouveaux acteurs de l'industrie.

Dans un premier temps, des penseurs de différentes disciplines ont théorisé la structuration des connaissances et l'accès indexé à l'information, y compris à distance et les possibles contextes d'utilisation.

Paul Otlet avait imaginé à la fin des années 1930 un dispositif utopique permettant de lire à domicile, à distance donc, des pages de documents stockés dans de grandes bibliothèques grâce à un télescope électrique pointé vers une salle d'exposition dédiée [1]. L'ingénieur américain Vannevar Bush, chercheur au MIT, décrivit ensuite en 1945 dans son article fondateur « *As we may think* », le Memex, un dispositif documentaire technique imaginaire posant les bases théoriques des documents numériques hypertextes [2]. En 1968 Douglas Engelbart, le père des interfaces hommes-machines, présenta NLS (*oNLine System*) le premier dispositif d'édition de documents en ligne intégrant la gestion des liens. Enfin, après un long saut dans le temps, c'est en 1991 que le physicien Tim Berners Lee proposa au sein du projet « *World Wide Web* », HTML un langage de présentation documentaire simplifié, dérivé du complexe SGML et appuyé sur le protocole HTTP. Il eut l'idée d'appliquer au réseau Internet existant la technique des liens hypertextes avec les adresses (URL et URI) au sein d'une interface, améliorant ainsi les précédents travaux présentés. Cependant, ce que ce langage de présentation gagnait en simplicité, il le perdait en capacité de structuration. C'est la raison pour laquelle le besoin de structuration documentaire s'est de nouveau fait ressentir et qu'est apparu en 1998 un nouveau format de structuration de document à balises : l'*eXtensible Markup Language* ou XML.

2. Contexte historique et généalogie de l'XML

Après cette petite introduction des théories qui ont préfiguré l'usage du format XML à travers le Web, voici une courte histoire de la gestion technique des documents en contexte industriel et scientifique.

2.1 GML

Dès 1967, l'idée d'un langage séparant le fond (les informations) et la forme (les instructions de présentation) est apparue. On en attribue généralement la paternité à l'ingénieur William W. Tunnicliffe. Deux ans plus tard, en 1969, Charles Goldfarb [4], chef de projet R & D chez IBM, se lançait dans un projet de Gestion Électronique de Documents en contexte juridique nécessitant l'usage d'un langage documentaire précis. Avec deux collègues, E. Mosher et R. Lorie, ils créèrent ledit langage suivant le principe de séparation fond/forme de Tunnicliffe. Ce langage sobrement nommé *Generalized Markup Language* (GML, les initiales des patronymes des trois ingénieurs correspondent également à l'acronyme GML) permettait d'offrir dans le cadre du droit une structure documentaire formelle avec des éléments imbriqués. Cette avancée avait pour objet de permettre des fonctions d'assistance à la recherche juridique, notamment relatives aux jurisprudences. Une offre commerciale d'éditique basée sur le GML fut également proposée par IBM principalement dans une optique de publipostage pour être utilisée avec SCRIPT, le principal outil de mise en page de la firme.

Le langage GML était composé de balises de formalisation du texte que l'on peut retrouver dans les langages qui ont suivi, jusqu'au HTML moderne. Une balise GML est composée d'un radical explicite en majuscule, préfixé par le caractère « : » et suffixé par un point.

Par **exemple** une liste ordonnée (*ordered list*) est initiée par la balise « :OL. », chaque item de la liste (*list item*) par : « :LI. ». Enfin, la liste sera terminée par la balise fermante de liste ordonnée (*end ordered list*) : « :EOL. »

Les imbrications des éléments et sous-éléments sont matérialisées par des sauts de ligne suivis d'une indentation.

Voici un exemple d'extrait de code GML qui présente sous forme d'énumération l'historique des langages à balises :

```
:OL.
                    :LI. GML
                    :LI. SGML
                    :LI. HTML
                    :LI. XML
                    :LI. xHTML
:EOL.
```

Important : On remarque la fermeture implicite des balises « :LI. », le simple fait de passer à la ligne suffit ici à indiquer au parseur qu'il y a fermeture de l'item.

Le rendu de ce code sera :
1/ GML
2/ SGML
3/ HTML
4/ XML
5/ xHTML

Le langage GML est publié en 1978 et commercialisé en 1980. La galaxie des langages de documentation à balises était loin de s'arrêter ici. Charles Goldfarb avait entamé dès 1974 la conception d'une suite au GML : le *Standard Generalized Markup Language* ou SGML.

2.2 SGML

Si le GML permettait une ébauche de séparation du fond et de la forme avec l'usage de balises, le SGML va plus loin. En effet, les données textuelles sont maintenant désolidarisées du formalisme par une symbolique des balises plus forte : les balises sont plus repérables, car désormais encadrées par des chevrons. De plus la présentation est externalisée grâce à la feuille de style qui peut être plurielle selon le(s) support(s) d'affichage souhaité(s). Le même document pouvait donc être présenté pour l'impression ou pour l'affichage sans modification, uniquement en changeant de feuille de style DSSSL (*Document Style Semantics and Specification Language*, la feuille de style pour SGML). Pour aller encore plus loin sur la séparation fond/forme, la structure du document – c'est-à-dire le bon agencement des balises - fut également spécifiée dans une grammaire sortie du document, ce que l'on appelle le *Document Type Definition* ou DTD (La feuille de style et la grammaire sont théoriquement placées dans des fichiers séparés, mais il est également possible de les placer en en-tête de document).

On note deux types de balises SGML, celles qui vont donner des caractéristiques d'édition au texte encadré comme la casse ou l'alignement, et celles dites descriptives ou sémantiques qui décrivent le texte encadré.

Si l'on reprend l'exemple donné précédemment en GML, sa traduction en SGML passerait par une DTD (nous reviendrons plus tard sur la rédaction d'une grammaire DTD) qui définirait qu'une liste est un ensemble composé d'un ou de plusieurs items, d'une feuille de style DSSSL et bien sûr du code de balisage avec le contenu encadré de balises :

```
<OL>
        <LI>GML</LI>
        <LI>SMGL</LI>
        <LI>HTML</LI>
        <LI>XML</LI>
        <LI>xHTML</LI>
<OL>
```

Le rendu resterait le même, les présentations étant déléguées aux feuilles de style pour le ou les supports d'affichage.

Le SGML trouve un écho fort dans la documentation au sein de l'industrie américaine au début des années 1980, IBM en tête, mais aussi en Europe dans les secteurs de pointe comme l'aéronautique et devient la norme ISO 8879 en 1986 (voir section « Normes et standards »). Du monde de l'industrie, le SGML rejoint celui de la défense, mais aussi celui de la recherche. Il devient un langage documentaire utilisé dans de prestigieux instituts de recherche dont le CERN, au sein duquel il va trouver une implémentation particulière qui va révolutionner et démocratiser le monde de la communication.

2.3 HTML

Lorsqu'en août 1991, Tim Berners-Lee annonce depuis le CERN la mise en ligne de la première page du projet World Wide Web, il donne un lien de document portant une nouvelle extension : le .html [5]. La volonté de Berners-Lee était de proposer un système de partage de documents qui pourraient être disponibles à distance, en réseau donc, et que ces documents puissent former un maillage hypertexte selon la formule proposée par Ted Nelson. Cette première mouture du Web avait donc une vocation documentaire et convoquait à la fois l'usage du tout nouveau protocole de transfert hypertexte (HTTP), l'adressage des ressources avec les identifieurs de ressources uniques (URI, forme générique des URL) et un nouveau langage autorisant l'ancrage de liens : le HTML.

La création d'un langage documentaire hypertexte devait répondre à deux critères : adhérer à une communauté préexistante et être suffisamment simple pour être utilisable [3].

D'un côté, le SGML évoqué précédemment était utilisé au CERN et était considéré comme l'avenir de l'hypertexte dans les communautés de recherche sur le sujet, de l'autre ce langage n'était pas réputé pour être trivial à manipuler. Le choix d'un compromis a été fait par Berners Lee avec la première mouture HTML qui simplifiait le SGML pour le rendre accessible à une communauté plus vaste. Ce choix était judicieux puisqu'il a conduit à l'adoption et à la démocratisation du Web.

Si le HTML a permis une simplification structurelle du SGML, cette orientation vers la présentation a forcément eu des conséquences en matière d'organisation documentaire. Le HTML avait perdu la capacité de description documentaire fine qui était associée au SGML. Afin de proposer un langage de description et de structuration d'une granularité plus fine que le HTML, mais avec une syntaxe plus simple que le SGML, un compromis a été proposé avec un nouveau langage : le XML.

3. Structuration XML

XML (pour *eXtensible Markup Language*) est un langage qui permet de structurer de l'information textuelle. La première version apparaît en 1998 suite aux recommandations du *XML Core Working Group* du W3C (voir section « Normes et standards »), dont le formatage est fortement inspiré du SGML et du HTML vus précédemment. La version 1.1 de XML (voir section « Normes et standards ») est la dernière modification apportée à ce format en 2008. Les principales différences viennent d'une part de l'utilisation des caractères *Unicode 4.0*, aussi bien dans le texte (version 1.0) que les balises ou les attributs, et d'autre part d'une liberté accrue sur les noms des balises. Nous allons voir dans ce qui suit la structure d'un document XML.

3.1 Balises

La principale ressemblance avec les langages de formatage repose sur les balises (ou éléments), comme pour HTML. Elles permettent de « marquer » les différentes parties d'un contenu. Le but de ces balises est de permettre aux applications qui vont les lire de se repérer dans le contenu. L'exemple de la figure **1** illustre le contenu formaté d'un document XML.

Un document XML est précédé d'un en-tête de description avec la version, ici « 1.1 » et l'encodage du texte « UTF-8 » (voir section « Normes et standards »). Un document XML est composé d'une seule et unique balise racine, ici « livre ». La balise racine peut, quant à elle, être composée de plusieurs balises et textes.

Toute balise ouverte (ex. « <livre> ») doit toujours être fermée explicitement, soit avec une nouvelle balise du même nom en ajoutant le caractère « / » au début (ex. « titre » ou « chapitre »).

Pour ce faire, la balise ouvrante peut soit être fermée directement (ex. « <saut/> »), soit avec une balise de fermeture commençant par un slash (ex. « </livre> »).

Chaque nom de balise est encadré par les caractères « < » et « > », et les caractères sont sensibles à la casse (minuscules et majuscules). Le nom ne doit pas contenir d'espaces, et depuis

```
<?xml version="1.1" encoding="UTF-8"?>
<livre>
        <chapitre> texte
                <para>texte</para> texte
                <para>texte</para>
        </chapitre>
        <chapitre>
                <para>texte</para>
                <saut/>
                <para></para>
        </chapitre>
</livre>
```

Figure 1 – Exemple de document XML – Livre avec chapitres

la version XML 1.1, il est possible d'utiliser des caractères *Unicode* (à préciser dans l'en-tête comme nous le verrons dans le chapitre suivant). Le nom peut commencer par n'importe quel caractère, quelle qu'en soit la langue, plus les caractères spéciaux « _ » et « : », excepté les chiffres (0-9). Par la suite, le nom peut, en plus, intégrer les chiffres, les tirets « - » et les points « . » ou « · » (unicode #xB7). Plus de détails sont disponibles dans la spécification de XML Name [6].

NB

Si vous souhaitez utiliser les caractères « < » et « > » dans du texte, il faut les remplacer par les entités nommées « < » et « > ». En effet, ces deux caractères sont réservés pour la définition de balises et ne peuvent donc être utilisés tels quels dans le texte.

Un texte ou « CDATA » (voir § 4.2) peut être intégré à n'importe quelle place dans le document. Il exprime le fait d'avoir une information liée à la balise « père », dont l'ordre dépend du placement des balises précédentes et suivantes.

Pour manipuler les documents XML, et faire à la fois la coloration syntaxique et la vérification, plusieurs logiciels existent : *Editra, Emacs, Notepad++, JEdit, SublimeText, Oxygen* (Voir section « Logiciels »).

3.2 Arborescence

Comme nous avons pu le constater, un document XML correspond à une hiérarchie de balises imbriquées les unes dans les autres. Cette hiérarchie est appelée arborescence et forme un arbre d'éléments que nous appellerons arbre DOM présenté plus bas. Elle pourra être exploitée par la suite pour identifier les nœuds de l'arbre. Cette arborescence exprime la profondeur de détails donnée à un élément avec des sous-éléments dont les propriétés raffinent l'élément parent (et ses ascendants).

Toute balise ouverte à l'intérieur d'un élément doit obligatoirement être fermée avant la fermeture de la balise parente. Il ne peut pas y avoir d'enchevêtrement de balises. Le contre-exemple de la figure **2** illustre un document non valide au niveau de son arborescence.

3.3 Séquence

Le nom d'une balise n'est pas unique et peut se répéter. Cela permet d'exprimer une notion de séquence comme ici le fait qu'un livre contient plusieurs *chapitres*, eux-mêmes composés de plusieurs paragraphes. Nous pouvons remarquer qu'une séquence

```
<livre>
   <chapitre>
      <para>
   </chapitre>
      </para>
</livre>
```

Figure 2 – Contre-exemple : l'erreur du chevauchement de balises

n'est pas forcément composée d'un seul nom de balise, en effet la séquence de *para* est également composée de *sauts* (de page).

L'ordre des éléments d'une séquence est important. Comme nous le verrons dans le langage XPath (voir § 6.2), ces séquences pourront être interrogées.

3.4 Attributs

Toute balise ouvrante peut contenir un ou plusieurs attributs. Le nom de l'attribut dans une balise est unique, il représente l'association d'une paire clé/valeur directement à l'élément (ou balise) considéré, et permet ainsi de lui donner des propriétés. L'ordre des attributs d'un élément n'a pas d'importance. Le document XML de la figure **3** détaille l'exemple de la figure **1** en lui associant des attributs.

Nous pouvons constater que l'attribut « titre » peut être associé aussi bien à « livre » que « chapitre » ; toutefois il ne pourra être répété deux fois dans un même élément.

Le nom de l'attribut respecte les mêmes règles que le nom d'une balise. Il est suivi directement par le caractère « = » et d'une valeur de type CDATA encadrée par des guillemets. On pourra identifier quelques types d'attributs particuliers comme les identifiants « ID », références « IDREF/IDREFS » détaillés par la suite. Le type d'un attribut est défini dans le schéma (voir § 4.2 et 4.4).

NB

Le guillemet étant réservé pour délimiter la valeur d'un attribut, il ne peut être utilisé dans le texte correspondant. Il sera alors remplacé par l'entité nommée « " ».

Attribut ID

Ce type d'attribut permet d'identifier l'élément associé de manière unique. La valeur de l'identifiant est unique pour l'ensemble du document.

```
<?xml version="1.1" encoding="UTF-8"?>
<livre titre="titre du livre">
        <chapitre titre="titre du chapitre 1" page="1"> texte
                <para>texte</para> texte
                <para>texte</para>
        </chapitre>
        <chapitre titre="titre du chapitre 2" page="10">
                <para>texte</para>
                <saut type="page"/>
                <para></para>
        </chapitre>
</livre>
```

Figure 3 – Exemple de document XML étendu (voir exemple de la figure 1)

```
<?xml version="1.1" encoding="UTF-8"?>
<livre titre="Titre du livre" livreID="MonLivre">
   <chapitre titre="Titre du premier chapitre" page="1" chapID="Chap1">
      <para>texte 1</para>
      <para suite="Chap2">texte 2</para>
   </chapitre>
   <chapitre titre="Titre du chapitre 2" page="11" chapID="Chap2">
      <para>texte 1</para>
      <saut type="page" />
      <para suite="Chap1">texte 2</para>
   </chapitre>
</livre>
```

Figure 4 – Illustration des IDREF

Deux documents XML distincts peuvent avoir une valeur d'identifiant identique. En effet, les deux documents étant eux-mêmes identifiés de manière unique, leur contenu l'est également.

Il est à noter que l'attribut de type « ID » ne doit pas forcément avoir pour nom « ID ». En effet, le type n'est défini que par le schéma DTD ou XSD associé au document XML (voir sections 5.2 et 5.4).

Attribut IDREF(S)

Ce type d'attribut permet de faire référence à un élément identifié par un ID. La valeur doit correspondre à un élément du document, ou identifier un document externe explicitement auquel l'élément ciblé est identifié.

Les attributs de type IDREFS permettent de spécifier une séquence de références, les valeurs d'identifiants sont alors séparées par des espaces.

La figure **4** illustre l'utilisation d'attributs de type ID (pour les éléments « livre » et « chapitre ») et IDREF (attribut « suite »). Toutefois, ce typage n'est valable que lorsque le document XML est associé à un schéma DTD ou XSD donnant pour ces attributs le type « ID » ou « IDREF(s) ». L'attribut « suite » est utilisé dans cet exemple pour faire référence au chapitre faisant suite au chapitre courant. Nous pouvons constater que la suite de « Chap1 » se trouve explicitement identifiée dans le chapitre « Chap2 », et donc à l'extérieur du présent document XML (mais avec un ID similaire).

Ce document XML servira de référence pour les exemples par la suite.

3.5 Un document bien formé

Le XML est à la fois un type de document et un langage qui obéissent tous deux à un formalisme garantissant l'interopérabilité des fichiers et de leurs contenus entre deux systèmes logiques, qu'il s'agisse de cognition humaine ou d'interprétation machine. Pour ce faire, un fichier XML doit obéir à certaines règles qui sont garantes de son usabilité.

En premier lieu, un fichier XML doit contenir un certain nombre d'informations pour être déclaré « bien formé », la plupart sont celles explicitées en partie 3 :
1/ Un en-tête doit expliciter qu'il s'agit bien d'un document de type XML. On y retrouve la spécification du format XML utilisé (1.0 ou 1.1). Une autre information particulièrement sensible y est stockée, celle de l'utilisation d'un format d'encodage. Dans l'exemple si après, nous choisissons la version 1.1 du format XML et l'encodage Unicode « UTF-8 » (forme la section Normes et standards) des caractères pour une compatibilité universelle. Ex. d'en-tête :
<?xml version="1.1" encodage="UTF-8"?>
2/ Une balise racine unique (voir § 3.1).
3/ Ne pas avoir de « chevauchement » de balises.
4/ Toute balise ouverte doit être fermée.
5/ Les contenus d'attributs doivent être encadrés de guillemets anglo-saxons simples ou doubles (voir § 3.4).
6/ Les caractères spéciaux, ceux ayant une utilité en XML, ne doivent pas être utilisés dans les contenus : c'est le cas des guillemets pour la cause évoquée au § 3.4.

Pour plus de détails, vous pouvez consulter la section « *well formed* » de page XML du W3C à ce sujet (voir section Normes et standards).

4. Universalité XML : espaces de nommage et schémas

4.1 Un document valide ?

Pour qu'un document soit déclaré « valide », c'est-à-dire exempt d'erreurs et donc d'une compatibilité maximale, il doit bien sûr valider le prérequis du formalisme : le document doit donc être « bien formé », sur le modèle de ce qui a été évoqué précédemment. Pour valider les problématiques de syntaxe et d'imbrication de balises, il existe plusieurs méthodes. Un outil simple et officiel est le validateur du W3C (https://www.w3schools.com/xml/xml_validator.asp) en ligne qui va vérifier point par point ces aspects. Pour les utilisateurs de systèmes d'exploitation basés sur Linux (dont Mac OS), un bon outil de vérification en ligne de commande est xmllint. Les éditeurs de code source ne sont pas égaux face à la détection d'erreurs, mais certains comme XML Spy, Oxygen ou JEdit, avec le plug-in XML, sont tout à fait aptes à déboguer les erreurs de syntaxe, les chevauchements de balises ou les erreurs d'encodages (Voir section « Logiciels »). Cependant, le site du W3C a une vocation didactique complémentaire.

Pour qu'un document soit valide et prêt à être partagé dans de bonnes conditions, le fait d'être bien formé est une condition nécessaire, mais pas toujours suffisante. En effet, dans certains cas, des communautés de pratiques ayant des besoins spécifiques vont devoir se doter d'outils complémentaires. Ces compléments vont être des « garde-fous » pour encadrer les pratiques d'XML. Par analogie avec les outils du linguiste, on va pouvoir les comparer de manière triviale à des dictionnaires (les termes que l'on utilise) et des grammaires (la manière d'agencer les termes), voire les deux : des manuels qui vont à la fois présenter les termes et leurs usages. Dans les quatre parties qui vont suivre, nous introduisons les concepts de DTD (*Document Type Definition*), de *namespaces* (dictionnaires), de schémas de données et son alternative : RELAX-NG.

4.2 Les DTD

La première étape d'uniformisation dans le formalisme d'un document XML est le *document type definition* ou DTD. Ce type d'informations permet qu'un fichier XML bien formé soit déclaré valide. Il n'est cependant pas nécessaire de disposer d'une DTD pour afficher un fichier XML. Dans ce cas, le fichier XML est déclaré « *standalone* » (autonome). Notons au passage qu'il est possible d'intégrer la DTD directement dans le document XML ou de l'externaliser dans un fichier à part. Par convention, les informations de DTD sont placées dans un fichier à part (extension .dtd) qui sera convoqué depuis le fichier XML.

Un fichier de DTD est donc un méta document puisqu'il s'agit d'un document décrivant un autre document. La DTD peut être vue comme une grammaire parce qu'elle décrit le nombre et la hiérarchie des balises autorisées dans le ou les fichier(s) XML qui l'utilisent.

Cette DTD peut être spécifique au système d'information d'une entité (entreprise ou administration par exemple) et dans ce cas sera qualifiée de « SYSTEM » ou alors partagée par une communauté de pratique (implémentant une norme ou un standard) et alors sera indiquée « PUBLIC » dans la partie « Doctype » du fichier XML qui la convoque.

Si nous reprenons l'exemple de la figure **4**, une structure XML propre à un éditeur d'ouvrages littéraires qui souhaite se doter d'une grammaire DTD, le fichier XML devra intégrer un doctype qui charge le fichier DTD livre.dtd en le qualifiant de « SYSTEM » (figure **5**).

Le fichier de DTD devra décrire de manière précise l'agencement des balises, leurs attributs, le nombre d'éléments fils autorisés, le type des contenus autorisés et leur caractère obligatoire.

4.2.1 Déclaration des éléments dans la DTD

Une balise sera déclarée dans la DTD par la balise <!ELEMENT> dont l'usage est :

<!ELEMENT nom-de-la-balise (contenus)>.

Les éléments peuvent être :
1/ Des balises filles séparées par des virgules.
2/ S'il y en a plusieurs de même type, elles seront suivies soit d'un « + » qui signifie qu'il s'agit d'une liste ou d'une « * » qui signifie une suite. Dans la pratique, cela signifie aussi en termes de cardinalités : « + » = au moins un, « * » = 0, 1 ou plusieurs.
3/ Le type de contenu de la balise si elle n'a pas de balises filles : ce peut être par exemple du texte parsable, par exemple noté #PCDATA.
4/ Un élément qui ne contient pas de contenus ni de sous-éléments, dont la fonction sera donc de déclarer des attributs uniquement, sera déclaré EMPTY.

4.2.2 Déclaration des attributs dans la DTD

Ensuite, les attributs obligatoires ou possibles seront présentés un à un par la balise <!ATTLIST> dont l'usage est :

```
<!ATTLIST nom-balise nom-de-attribut1 type-attribut obligatoire-ou-pas>
<!ATTLIST nom-balise nom-de-attribut2 type-attribut obligatoire-ou-pas>
```

```
<?xml version="1.1" encoding="UTF-8"?>
<!DOCTYPE livre SYSTEM "livre.dtd">
<livre titre="Titre du livre" livreID="MonLivre">
  ....
</livre>
```

Figure 5 – Illustration de l'appel de la DTD dans livre.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT livre ( chapitre* ) >
<!ATTLIST livre
   livre ID ID #REQUIRED
   titre CDATA #REQUIRED >
<!ELEMENT chapitre ( para | saut )* >
<!ATTLIST chapitre
   chapID ID #REQUIRED
   page NMTOKEN #REQUIRED
   titre CDATA #REQUIRED >
<!ELEMENT para ( #PCDATA ) >
<!ATTLIST para
   suite IDREF #IMPLIED >
<!ELEMENT saut EMPTY >
< !ATTLIST saut
   type CDATA #FIXED "page" >
```

Figure 6 – Illustration de la DTD livre.dtd correspondant à la grammaire appelée dans livre.xml

Pour décrire un attribut, il faut préciser si ce dernier :

– correspond à un identifiant (unique donc) ou à un pointeur avec respectivement ID et IDREF (voir § 3.4) ;
– est obligatoire, optionnel ou fixe avec #REQUIRED, #IMPLIED ou #FIXED ;
– a des valeurs fixées (choix bridé) qui seront alors présentées entre parenthèses et séparées par le signe « | », une valeur par défaut pouvant être définie ;
– se présente sous la forme type CDATA, c'est-à-dire sous forme de chaîne de caractères libres ou NMTOKEN sous forme de valeurs normalisées (ex : date).

Dans l'exemple de la figure **6**, un livre va être composé d'une suite de balises <chapitre>. Chaque chapitre sera composé de <para> et de <saut>. Un livre aura obligatoirement un identifiant unique en attribut qui sera nommé livreID et un titre constitué d'une chaîne de caractères libres. Ensuite chaque chapitre sera constitué de plusieurs paragraphes et de sauts, les chapitres auront obligatoirement un identifiant unique, une page (... de début) normalisée et un titre. À l'intérieur d'un chapitre, il y aura des paragraphes contenant du texte et éventuellement un attribut suite qui contient un pointeur vers une URL. Un chapitre peut contenir également un saut de page.

Afin de tester la validation de documents avec la DTD, vous pouvez utiliser le validateur en ligne : https://xmlvalidation.com/. En intégrant la DTD et le document XML associé, vous pourrez vérifier le schéma du document.

4.3 Espaces de nommage (*namespaces*)

Les documents XML pouvant être liés à un schéma, ou encore une grammaire définissant la structure du document (voir § 4.4), il est nécessaire de différencier des documents similaires, mais avec deux définitions de schémas, voire d'identifier les éléments similaires à l'intérieur d'un même document. En effet, un élément contenu dans le document XML peut avoir différentes sémantiques en fonction du contexte d'utilisation et être associé à des schémas distincts.

Prenons l'exemple d'un élément <id> dans un document XML correspondant à un achat de produit. Cet élément pourrait en soit être associé à un élément père <client> soit à un élément père <produit>, comme le montre l'exemple de la figure **7**. Comment faire la différence sémantique de l'identifiant pour qu'il référence bien dans le premier cas l'identifiant du client, et le deuxième l'identifiant du produit ? L'espace de nommage permet de résoudre cette ambiguïté en associant à chaque élément sa définition. C'est un lieu abstrait, généralement une URI, auquel on associe l'espace

```
<commande xmlns="http://www.example.org/MonSchemaCommande">
  <id>Commande5678</id>
  <date>Tue, 22 Oct 2019 14:50:00 GMT</date>
  <client xmlns="http://www.example.org/MonSchemaClient">
    <id>Client25</id>
    <nom>Dupont</nom>
    <prenom>Jean</prenom>
  </client>
<produit xmlns="http://www.example.org/MonSchemaProduit">
    <id>Produit1234</id>
    <nom>Techniques de l'Ingénieur</nom>
    <type>Livre</type>
  </produit>
</commande>
```

Figure 7 – Description d'une commande avec un espace de nommage pour la commande, le client et le produit

```
<comm:commande xmlns:comm="http://www.example.org/MonSchemaCommande"
               xmlns:cli="http://www.example.org/MonSchemaClient"
               xmlns:prod="http://www.example.org/MonSchemaProduit">
  <comm:id>Commande5678</comm:id>
  <comm:date>Tue, 22 Oct 2019 14:50:00 GMT</comm:date>
  <cli:client>
    <cli:id>Client25</cli:id>
    <cli:nom>Dupont</cli:nom>
    <cli:prenom>Jean</cli:prenom>
  </cli:client>
  <prod:produit>
    <prod:id>Produit1234</prod:id>
    <prod:nom>Techniques de l'Ingénieur</prod:nom>
    <prod:type>Livre</prod:type>
  </prod:produit>
</comm:commande>
```

Figure 8 – Utilisation de plusieurs espaces de nommage

de l'ensemble des noms ou éléments qui le composent. Ce lieu abstrait permet ainsi d'enlever toute ambiguïté entre des éléments similaires dans un même document.

Pour le définir, il est possible d'associer l'attribut « xmlns » à un élément pour lui définir son espace de nommage. Tous les sous-éléments de celui-ci héritent automatiquement de cet espace de nommage, sauf s'il y a une surcharge. L'exemple de la figure **7** décrit une commande avec un espace de nommage pour la commande, le client et le produit (respectivement surlignés en orange, bleu et vert) avec l'URI où le schéma est défini, et de fait est unique.

Il est également possible de manipuler plusieurs espaces de nommage et de les préciser par élément pour être précis. Il est alors nécessaire de donner un nom à cet espace de nommage, puis d'utiliser ce nom suivi de « : » pour spécifier le nom de chaque élément (figure **8**).

L'espace de nommage est très utilisé pour définir des ontologies avec le formalisme RDF (voir § 5.2). Toutefois, on peut retrouver plusieurs *namespaces* communément utilisés par la communauté, le tableau **1** présente ceux abordés dans cet article.

4.4 Schémas XSD

À l'instar de la DTD, les schémas XSD (voir section Normes et standards) ont été proposés en 2001 par le W3C pour permettre de définir de manière précise la structuration d'un document XML. La version de 2004 (voir section Normes et standards) n'est pas une deuxième version, mais une version corrective intégrant les errata du *Working Group* du W3C sur les *DataType*. Elle est très souvent utilisée dans le cadre des services web afin de garantir la conformité des documents XML échangés entre deux applications.

Le formalisme XSD répond à des besoins de spécifications que la DTD ne peut pas satisfaire de fait de sa trop grande simplicité. Il permet ainsi de définir un typage rigoureux des différents éléments contenus dans le document et l'ensemble de sa structure.

L'espace de nommage du format XSD est : http://www.w3.org/2001/XMLSchema et doit être associé à chaque élément du schéma. Il est simplifié par « xs: ».

Il est à noter qu'un schéma XML est lui-même un document XML, il est ainsi manipulable de la même manière (par exemple, l'outil oXygen), que le document contenant les données (XML) ou la feuille de style (XSL).

Le schéma du document doit être défini par sa structure de base telle que présentée à la figure **9**. L'élément racine est l'élément « xs:schema » avec pour espace de nommage « xs » qui sera associé à chaque élément du schéma.

Heureusement, il n'est pas utile d'écrire un XML-Schema à la main. De nombreux outils, tels que « mherman », « xmlgrid », « xsd-generator » (https://www.freeformatter.com/xsd-generator.html) ou « devutilsonline » (https://devutilsonline.com/xsd-xml/generate-xsd-from-xml), existent sur le Web et permettent de générer celui-ci à partir d'un document

Tableau 1 – Namespaces couramment utilisés

Préfixe	Namespace	Usage
fn	http://www.w3.org/2005/xpath-functions	Fonctions et opération XPath
html	http://www.w3.org/1999/xhtml	Page Web XHTML
xml	http://www.w3.org/XML/1998/namespace	Standard XML
xs	http://www.w3.org/2001/XMLSchema	Schéma XSD
xsl	http://www.w3.org/1999/XSL/Transform	XSL stylesheet
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns	Faits RDF
soap	http://www.w3.org/2003/05/soap-envelope/	Protocole SOAP
svg	http://www.w3.org/2000/svg	Image SVG
math	https://www.w3.org/1998/Math/MathML/	Formules en MathML
xsi	http://www.w3.org/2001/XMLSchema-instance	Dump de Bases de données

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" >
   <!-- Déclaration du schéma -->
</xs:schema>
```

Figure 9 – Document générique pour XML-Schema

XML exemple (voir section « Logiciels »). Il sera toutefois nécessaire de relire le schéma généré pour vérifier qu'il ne soit pas trop contraignant (liste de valeurs strictes, séquence d'éléments plutôt que choix, etc.).

4.4.1 Types

Pour chaque type de contenu, il est alors nécessaire de définir chaque élément constituant le document XML. Pour cela, trois orientations sont possibles, soit l'élément est simple (contient des données dont le type est prédéfini), soit l'élément est complexe « x*s:complexType* » (besoin de définir une structure spécifique), ou n'importe quel type « x*s:anyType* ».

NB : Types simples XSD : xs:string, xs:duration, xs:datetime, xs:date, xs:time, xs:boolean, xs:anyURI, xs:float, xs:double, xs:decimal, etc.

À la figure **10**, nous pouvons constater deux exemples de schémas illustrant le document (voir § 3). L'élément « para » est de type simple avec du texte « *xs:string* », tandis que l'élément « saut » ne contient pas de données, mais un attribut « type » obligatoire (use= "required") dont la valeur est une chaîne de caractères sans espaces « xs:string ». Nous pouvons voir que ce dernier est ainsi de type complexe « xs:complexType » (même s'il ne contient qu'un seul attribut).

```
<xs:element name="para" type="xs:string"/>
<xs:element name="saut">
  <xs:complexType>
    <xs:attribute name="type" use="required" type="xs:string"/>
  </xs:complexType>
</xs:element>
```

Figure 10 – Structure d'un xs:element

```
<xs:simpleType name="entre0et100">
  <xs:restriction base="xs:integer">
    <xs:minInclusive value="0"/>
    <xs:maxInclusive value="100"/>
  </xs:restriction>
</xs:simpleType>
```

Figure 11 – Création de type avec une restriction

Il est également possible de dériver un type simple pour définir des restrictions. Les restrictions peuvent être des bornes (xs:minInclusive, xs:maxInclusive & exclusive), des motifs (xs:pattern), des longueurs (xs:length, xs:minLength, xs:maxLength), des listes de valeurs (xs:enumeration), etc.

Par **exemple**, le type « entre0et100 » définit un nouveau type de données, héritant des propriétés d'un entier « integer » dont les valeurs sont comprises entre 0 (min) et 100 (max) (figure **11**).

Nous pourrions également appliquer des expressions régulières (restriction de type « pattern ») sur des chaînes de caractères comme l'exemple de la figure **12** proposant la validation d'une adresse e-mail.

Il est également possible de créer des unions de contraintes pour combiner des ensembles de valeurs plus complexes avec xs:union.

4.4.2 Types complexes

Le type complexe permet ainsi de définir des structures très riches telles que des séquences d'éléments (xs:sequence) ou des choix (xs:choice). Le type complexe apparaît dès lors qu'un attribut

```
<xs:simpleType name="CodeIntern">
  <xs:restriction base="xs:string">
    <xs:pattern value="[^@]+@[^\.]+\..+"/>
  </xs:restriction>
</xs:simpleType>
```

Figure 12 – Exemple de restriction avec expression régulièrement pour adresse e-mail

```
<xs:element name="livre">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" maxOccurs="unbounded" ref="chapitre"/>
    </xs:sequence>
    <xs:attribute name="livreID" use="required" type="xs:string"/>
    <xs:attribute name="titre" use="required" type="xs:string"/>
  </xs:complexType>
</xs:element >
```

Figure 13 – Illustration de création du type complexe « livre » (figure 1)

```
<xs:element name="chapitre">
  <xs:complexType mixed="true">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="para"/>
      <xs:element ref="saut"/>
    </xs:choice>
    <xs:attribute name="chapID" use="required" type="xs:string"/>
    <xs:attribute name="page" use="required" type="xs:integer"/>
    <xs:attribute name="titre" use="required" type="xs:string"/>
  </xs:complexType>
</xs:element>
```

Figure 14 – Illustration de création de type complexe pour le « chapitre » (voir figure 1)

doit être associé à un élément ou que l'élément contient lui-même des sous-éléments. Le nombre d'occurrences d'un élément peut être défini (minOccurs et maxOccurs). Dans, l'exemple de la figure **13**, le schéma de l'élément « livre » précise qu'il est composé d'une séquence d'éléments « chapitre » (min 0, max non défini), le lien vers la définition du « chapitre » est donné par référence grâce à l'attribut « ref ». Il permet ainsi d'alléger la définition en séparant chaque type.

La définition de l'élément « *chapitre* », comme présenté ci-dessous, est composée d'un « choix » entre les éléments « para » et « saut » dont les types font référence à la définition précédente (montrant ainsi la composition de types). Grâce à ce choix, il est possible d'ajouter sans ordre prédéfini les deux balises. Il est à noter que l'élément « chapitre » de type « complexType » est associé à l'attribut « *mixed='true'* ». Cela permet à l'élément chapitre de contenir du texte entre les balises (ce qui était illustré entre les différents « para » et « saut » (figure **14**).

4.4.3 Contraintes

Comme dans le cas des DTD, il est possible de définir des contraintes d'unicité ou de références sur les attributs. Ainsi, grâce à un type « *xs:key* » ou « *xs:keyref* », nous pourrons faire référence aux éléments uniques constituant le document XML. Le schéma de la figure **15** permet d'illustrer cette définition.

La définition d'une clé se fait par rapport à la hiérarchie de l'élément sélectionné. Dans notre cas, nous souhaitons que l'identifiant du *chapitre* soit unique dans le *livre*. Ainsi, il est nécessaire de définir cette contrainte d'unicité au niveau de l'élément *livre*. L'exemple de la figure **15** illustre ce fait en précisant que la clé de l'élément *chapitre* fils de *livre* (via xs:selector), contient un identifiant unique dénommé par « *chapID* » (xs:field). Les chemins sont exprimés grâce à XPath (voir § 6.2). Il suffit ensuite d'ajouter la définition de l'attribut « chapID » dans l'élément « chapitre » pour

```
<xs:element name="livre">
        <xs:complexType mixed="true">
                ...
        </xs:complexType>
        <xs:key name="chapID">
           <xs:selector xpath="livre/chapitre"/>
           <xs:field xpath="@chapID"/>
        </xs:key>
</xs:element>

<xs:element name="chapitre">
   <xs:complexType mixed="true">
        ...
   <xs:attribute name="chapID" use="required" type="xs:string"/> ...
   </xs:complexType>
</xs:element>
```

Figure 15 – Illustration de type complexe avec identifiant (voir figure 4)

que celui-ci puisse être utilisé. Le « xs:key » ne sert que de définition de contrainte d'unicité (figure **16**).

Nous avons donc étendu les définitions de « chapitre » et « para ». Le premier (figure **15**) contient maintenant un attribut « chapID » unique obligatoire. Le second peut contenir un attribut « suite » (figure **16**) qui fait référence à un attribut « chapID » déjà défini.

Dans le cas où nous souhaitons appliquer une contrainte stricte sur la valeur d'un attribut, il est possible d'utiliser l'élément « xs:assertion » pour effectuer une assertion. Pour illustrer cela,

```
<xs:element name="livre">
   ...
   <xs:keyref name="suite" refer="chapID">
        <xs:selector xpath="livre/chapitre/para"/>
        <xs:field xpath="@suite"/>
   </xs:keyref>
</xs:element>
<xs:element name="para">
   <xs:complexType mixed="true">
         <xs:attribute name="suite" use="optional" type="xs:string"/>
   </xs:complexType>
</xs:element>
```

Figure 16 – Illustration de type complexe avec référence d'identifiant (voir figure 4)

```
<xs:element name="chapitre">
        <xs:complexType mixed="true">
            ...
            <xs:attribute name="page" use="required" type="xs:integer"/>
            <xs:assertion test="(@page mod 2) eq 1"/>
                <!-- fonction modulo : @page modulo 2 = 1 -->
          ...
        </xs:complexType>
</xs:element>
```

Figure 17 – Exemple de contrainte d'assertion avec formule

nous allons indiquer que la valeur de l'attribut « page » d'un « chapitre » commence toujours par une page impaire à l'aide d'une expression (figure **17**).

Il est également possible de faire une restriction avec une expression régulière avec l'élément « xs:pattern ». Vous trouverez un exemple à la figure **18**.

Et maintenant, il serait intéressant de pouvoir spécifier une restriction d'un ensemble de valeurs pour un attribut donné. Nous utiliserons alors l'élément « xs:restriction ». Prenons l'exemple de la balise « saut », il peut prendre les valeurs « page » ou « ligne », la restriction fait alors une énumération « *xs:enumeration* » des valeurs possibles (figure **19**).

Le schéma XML obtenu après agrégation des exemples donnés est complexe et très verbeux. C'est une des raisons qui fait que la DTD est parfois utilisée pour éviter une trop grande complexité. Toutefois, le schéma est bien plus précis et évite de nombreuses erreurs possibles lors de l'interopérabilité de services (services web notamment). Vous pourrez trouver une synthèse de la grammaire de XML-schema sur cette refCard : http://www.info.univ-angers.fr/pub/gh/refcards/XML_Schema_Data_Structures.pdf.

```
<xs:attribute name="page" use="required">
   <xs:simpleType>
      <xs:restriction base="xs:integer">
         <xs:pattern value="\d*[13579]"/>
      </xs:restriction>
   </xs:simpleType>
</xs:attribute>
```

Figure 18 – Exemple de contrainte de restriction avec "xs:pattern"

```
<xs:element name="saut">
   <xs:complexType>
      <xs:attribute name="type" use="required" type="xs:string">
         <xs:simpleType>
            <xs:restriction base="xs:string">
               <xs:enumeration value="page"/>
               <xs:enumeration value="ligne"/>
            </xs:restriction>
         </xs:simpleType>
      </xs:attribute>
   </xs:complexType>
</xs:element>
```

Figure 19 – Exemple de contrainte d'assertion par énumération

```
<?xml version="1.0" encoding="UTF-8"?>
<livre xmlns="http://www.montest.com/livre.xsd">
...
</livre >
```

Figure 20 – Association du schéma XML au document pour validation

4.4.4 Validation de documents

La validation de schéma repose sur la théorie des automates à base d'arbres [4]. Si la définition du nœud correspond au type de l'arbre requis par le schéma, l'automate passe alors dans l'état suivant et vérifie la série de nœuds présents dans le document XML. Le schéma XML étant strict, la vérification est complexe et doit ainsi vérifier l'ensemble des propriétés du nœud en cours et ses descendants. Si un seul état de l'automate est invalidé, le document est considéré comme invalide.

Dès lors qu'un schéma XSD est associé à un document XML, la vérification du document est automatiquement appliquée par l'application de lecture du document. Le schéma peut être intégré directement au document ou passé en référence comme le montre l'en-tête (section « document bien formé ») de la figure **20** avec le schéma spécifié à l'adresse du document XSD « http://www.montest.com/livre.xsd ».

Il est possible de tester la validation du schéma avec des outils en ligne, tels que : https://www.freeformatter.com/xml-validator-xsd.html. Afin de faciliter la lecture, chaque type d'élément est identifié séparément et référencé par l'attribut « type ». Pour le tester en ligne, vous pouvez copier le schéma complet (figure **21**) ainsi que le document XML référence (XML avec IDREF), puis coller les deux documents dans les formulaires correspondants de la page de freeformatter.com.

5. Usages du XML

5.1 DOM – *Document Object Model*

Afin de faciliter la visualisation et la manipulation de documents XML, le DOM ou *Document Object Model* (https://dom.spec.whatwg.org) est une interface de programmation normalisée par le W3C (https://www.w3.org/TR/DOM-Level-3-Core). Il permet ainsi d'appliquer des scripts ou programmes sur le document pour l'intégrer à une application. Ainsi, le document XML est représenté sous forme d'objets imbriqués donnant lieu à une arborescence facilitant la représentation.

Le DOM (Voir section Normes et standards, parties W3C et WHATWG) est communément représenté sous une forme arborescente

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="livre"><!------- Elément Livre -------->
   <xs:complexType>
      <xs:sequence>
         <xs:element minOccurs="0" maxOccurs="unbounded" ref="chapitre"/>
      </xs:sequence>
      <xs:attribute name="livreID" use="required" type="xs:string"/>
      <xs:attribute name="titre" use="required" type="xs:string"/>
   </xs:complexType>
   <xs:key name="chapID">
      <xs:selector xpath="livre/chapitre"/><xs:field xpath="@chapID"/>
   </xs:key>
   <xs:keyref name="suite" refer="chapID">
      <xs:selector xpath="livre/chapitre/para"/><xs:field xpath="@suite"/>
   </xs:keyref>
</xs:element>
<xs:element name="chapitre"><!------- Elément Chapitre -------->
   <xs:complexType mixed="true">
      <xs:choice minOccurs="0" maxOccurs="unbounded">
         <xs:element ref="para"/><xs:element ref="saut"/>
      </xs:choice>
      <xs:attribute name="chapID" use="required" type="xs:string"/>
      <xs:attribute name="page" use="required">
         <xs:simpleType><xs:restriction base="xs:integer">
            <xs:pattern value="\d*[13579]"/>
         </xs:restriction></xs:simpleType>
      </xs:attribute>
      <xs:attribute name="titre" use="required" type="xs:string"/>
   </xs:complexType>
</xs:element>
<xs:element name="para"><!------- Elément Para -------->
   <xs:complexType mixed="true">
      <xs:attribute name="suite" use="optional" type="xs:string"/>
   </xs:complexType>
</xs:element>
<xs:element name="saut"><!------- Elément Saut -------->
   <xs:complexType>
      <xs:attribute name="type" use="required">
         <xs:simpleType><xs:restriction base="xs:string">
            <xs:enumeration value="page"/><xs:enumeration value="ligne"/>
         </xs:restriction></xs:simpleType>
      </xs:attribute>
   </xs:complexType>
</xs:element>
</xs:schema>
```

Figure 21 – Exemple complet de XML schema

permettant de consulter l'imbrication des éléments et des valeurs qui le composent. L'outil en ligne *codebeautify* illustre cette représentation en direct. La figure **22** est extraite de l'interface et reprend notre exemple de livre.

Le DOM est présent dans la plupart des langages de programmation pour favoriser l'intégration et la manipulation de documents XML. Parmi ceux-ci, nous pouvons noter JAXP pour Java (Java API for XML Processing), Xerces ou libxml2 pour C++, xml.dom pour Python (voir section « Logiciels et outils »). Ces librairies intègrent l'importation de documents XML et l'exportation d'arbres DOM. Il est également possible d'appliquer des vérifications de schéma (voir § 4.4) ou des règles XLST (voir § 5.2.3).

Pour l'utilisation du DOM, il est important de comprendre la structure de ses objets :

– *Document* : le document XML est représenté à sa racine par cet élément. Toutes les fonctions sont appliquées à partir de cet élément ;
– *Element* : chaque élément/nœud utilise cette interface pour manipuler son contenu ;
– *NodeList* : un élément peut contenir plusieurs éléments fils, ils sont encapsulés dans ce NodeList ;
– *Attr* : les attributs des nœuds sont représentés par cet élément ;
– *Comment* : les commentaires ;
– *Text* : les valeurs textuelles sont stockées dans cet objet ;

De nombreuses fonctions sont applicables à chaque objet, nous pouvons noter particulièrement : *getElementById* et *getElementByName*

Figure 22 – Exemple de DOM pour l'exemple de la figure 4

qui permettent de retrouver un ou plusieurs éléments contenus dans le document. Comme nous le verrons au § 6.2, nous pouvons également évaluer des requêtes XPath avec la fonction « *evaluate* » au niveau du document, tout en spécifiant ce que l'on appelle « le nœud contextuel » (c'est-à-dire le nœud où doit être appliquée la requête).

5.2 Structuration documentaire

Il est à noter que XML dans sa structuration de base ne contient pas de sémantique, mais uniquement un formalisme. Pour introduire cette notion de sémantique, les *namespaces* et *schémas* (DTD ou XSD) permettent de définir une structuration spécifique dont le contenu peut être reconnu par des programmes dédiés, qui pourront dès lors exploiter le contenu en fonction de la sémantique associée à ce *namespace*. Les formalismes suivants illustrent plusieurs sémantiques connues sur le Web.

5.2.1 XML dans le HTML avec JS (AJAX)

Un des premiers usages du XML dans la sphère du Web au début des années 2000 a été de mettre à disposition des flux de données structurées à disposition depuis des serveurs. Ces données sont souvent initialement stockées dans des bases comme celles disponibles sur des serveurs LAMP (Linux / Apache / MySQL / PHP), puis transformées en XML grâce à des scripts (PHP par exemple). Ensuite, des scripts JavaScript chargent ces jeux de données dans le navigateur avec les méthodes de l'objet XMLHttpRequest et font un traitement local dans le navigateur sur la machine de l'usager final afin de permettre un affichage sélectif d'informations. Cette technique dite AJAX (*Asynchronous JavaScript + XML*) offre deux avantages certains : elle offre la possibilité de mettre à jour le contenu d'une page web sans avoir à la recharger et sans avoir recours à un langage de programmation sur le serveur pour réorganiser la mise en page. En effet, grâce à ce modèle, les contenus sont directement injectés dans les balises HTML et la manipulation des propriétés des feuilles de style

par le JavaScript également complète dynamiquement la mise à jour de la page (https://www.w3schools.com/xml/ajax_intro.asp).

NB : Pour plus de détails sur Ajax, vous pouvez consulter le cours de Garrett, Jesse James. "Ajax: A new approach to Web applications, 2005." *Adaptive Path Inc* (2005). Archivé à l'URL (accédé le 18 avril 2020) [7].

Dans l'exemple que nous proposons à la figure **23**, le code HTML propose un hyperlien (ligne 23) qui, s'il est cliqué, déclenche la fonction JavaScript showRSS(), cette dernière demande alors à charger et d'injecter le résultat de l'exécution du script PHP rssLoader.php qui va charger et mettre en forme les news RSS (voir § 5.2.4) qu'il récolte sur le site de *Techniques de l'Ingénieur*.

Ainsi, depuis la page HTML et grâce à AJAX, l'affichage et la mise à jour d'un flux de données XML peuvent se faire à la demande et sans recharger la page.

5.2.2 Du xHTML au HTML5

Le xHTML (https://www.w3.org/TR/xhtml1) est une extension du HTML pensée au début des années 2000 avec la volonté de trouver un équilibre fond/forme pour les documents hypertextes : allier la puissance structurelle du XML à la possibilité de partager et présenter facilement en HTML. Les domaines d'applications ont été centrés autour de projets de documentation pointus qui

```php
<ol>
<?php
$xml=simplexml_load_file("https://www.techniques-ingenieur.fr/rss/actualite.xml")
     or die("Les News RSS de TI ne sont pas accessibles");
print('<h2>'.$xml->channel->title.'</h2>');
foreach($xml->channel[0] as $item){
  if($item->title!=''){
    print('<li><a href="'.$item->link.'">'.$item->title.'</a></li>');
  }
}
?>
</ol>
```

```html
<html>
  <head>
    <meta charset="UTF-8">
    <script>
    function showRSS() {
      if (window.XMLHttpRequest) {
        xmlhttp=new XMLHttpRequest();
      } else {
        xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
      }
      xmlhttp.onreadystatechange=function() {
        if (this.readyState==4 && this.status==200) {
          document.getElementById("rssOutput").innerHTML=this.responseText;
        }
      }
      url_rss_TI="./rssLoader.php";
      xmlhttp.open("GET",url_rss_TI);
      xmlhttp.send();
    }
    </script>
</head>
<body>
  <a href="javascript:showRSS()">Cliquez pour (re)charger les news de TI</a>
  <div id="rssOutput">ça va s'afficher ici ...</div>
</body>
</html>
```

Figure 23 – XML avec JavaScript

```
<?xml versiWon="1.0" encoding="UTF-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd" >
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
   <head>
      <title>xHTML< /title>
   </head>
   <body><p>Let's define what xHTML is.</p> </body>
</html>
```

Figure 24 – Fichier xHTML associé à règle DTD du xHTML1 strict

```
<? xml version="1.0 " encoding="UTF-8" ? >
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:epub="http://www.idpf.org/2007/ops"><!---- namespace ePub ---->
   <head>
      <meta charset="utf-8"/>
      <title>TI : XML< /title>
      <link rel="stylesheet" type="text/css" href="../styles/stylesheet.css"/>
   </head>
   <body epub:type="bodymatter"> <!-- typage selon le namespace ePub --->
      <h1>Article sur XML</h1>
      <h2>Introduction</h2>
      <p>Cet article parle de XML, ici plus précisément de l'usage xHTML dans le cadre d'un eBook de type ePub</p>
   </body>
</html>
```

Figure 25 – Exemple de source d'un eBook au format ePub 2 en xHTML

nécessitaient à la fois de la précision structurelle et une possibilité d'affichage hypertexte comme peut l'être celle d'un navigateur. Le HTML (version 4) et le xHTML ont cohabité dans l'espace du Web pendant presque une décennie, sans forcément que l'usage du xHTML soit expliqué par des besoins spécifiques liés à la complexité documentaire d'un projet.

Sans aller jusqu'à la complexité du SGML, le xHTML a été pensé plus pour structurer les connaissances que pour les présenter. Pour qu'un document soit valide en xHTML, il doit avoir des déclarations de type – aussi nommées DOCTYPE – et d'encodage de caractères conformes aux spécifications du consortium (figure **24**).

Cependant, l'arrivée du Web dit « sémantique » va entraîner une perte de vitesse et une baisse de l'usage d'un xHTML jugé trop complexe et lourd à manipuler au profit de l'HTML5. Notons cependant que le xHTML peut parfaitement utiliser des vocabulaires de description avec l'usage des espaces de nommage et faire de la sémantique. Le HTML5 ne se cantonne plus à des fonctions d'affichage qui sont d'ailleurs déléguées aux feuilles de style CSS. Les balises HTML5 ont des fonctions de structuration et d'ingénierie de la connaissance. De plus, de nouveaux modes d'inclusion de triplets RDF comme les microdonnées et le RDFa, dont l'usage est largement encouragé par le consortium des moteurs de recherche, vont finir de rendre le xHTML obsolète pour les sites web grand public. En 2009, le xHTML2 est abandonné au profit du HTML5, cependant le HTML5 est sérialisable en XML ce qui explicite un xHTML5. Plus de détails sur la transition entre xHTML, HTML5 et xHTML5 sont disponibles sur : https://www.w3.org/2009/06/xhtml-faq.html

Il est cependant un domaine dans lequel le xHTML est encore largement utilisé : l'édition numérique. La deuxième mouture du format de livre numérique *electronic publication* ou ePub v2 a été pensée autour d'une structuration documentaire en xHTML compressé dans une archive et a été adoptée très largement en 2007 par l'*International Digital Publishing Forum* (qui a fusionné dans le W3C en 2017 : https://www.w3.org/2017/01/pressrelease-idpf-w3c-combination.html.en). Un ePub est une archive contenant un fichier d'index (en XML), des fichiers de média, les fichiers de mise en forme (CSS), un fichier de métadonnées (voir § 5.2.5) et les fichiers de documents au format xHTML.

Dans l'exemple de la figure **25** (représentant une page d'un ePub), nous remarquons que le document XML, encodé en utf-8 ne fait pas appel à une DTD mais convoque les espaces de nommage (voir § 4.3) du xHTML et celui propre au format ePub : l'*Open Publication Structure*.

Même si théoriquement l'ePub 3 appuyé sur le HTML5 a été mis sur le marché depuis 2011, les usages de l'industrie peinent à suivre ces évolutions et le xHTML est encore utilisé dans l'industrie du livre numérique et des liseuses. Cependant, son usage se fait quasi exclusivement sous la forme de chaînes éditoriales automatisées comme Sigil.

5.2.3 XSLT

XSLT (*Extensible Stylesheet Language Transformations* v 3.0 : https://www.w3.org/TR/2017/REC-xslt-30-20170608/) est un langage formalisé en XML à base de règles permettant de définir une transformation d'un format XML vers tout autre format. Ce langage est très utile pour pouvoir définir un nouveau format de présentation pour un document respectant une certaine structure ou un schéma.

L'espace de nommage lié à XSLT est « *xsl:* » et est utilisé pour définir une feuille XSL (figure **26**).

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="3.0" > ...
</xsl >
```

Figure 26 – Structure générique pour un document XSLT

```
<xsl:template match="/">
...
</xsl:template>
```

Figure 27 – Structure générique pour la définition d'une règle *« template »*

La définition d'une feuille XSL repose sur un ensemble de règles qui vont être appliquées sur le contenu du document XML. À chaque règle correspond un modèle composé d'un élément « *xsl:template* » avec un attribut « match » pour définir les conditions d'application de la règle, et d'un contenu pour la transformation. L'exemple de la figure **27** présente le modèle avec le « match » appliqué à la racine du document XML à transformer.

La définition de ce modèle repose sur deux concepts majeurs :
1/ La définition d'un parcours dans le document pour chaque règle.
2/ La définition de la transformation pour chaque règle de transformation.

Le parcours du document repose sur la définition d'un chemin par étapes successives comme on pourrait exprimer un chemin dans une hiérarchie de fichiers. Ce chemin repose sur la définition du nom des nœuds, ou du motif, parcouru par la règle ainsi que des filtres qui peuvent leur être associés. Le langage associé à cette expression de chemins est le langage XPath que nous détaillerons au § 6.2.

La transformation donne la structure de la sortie ; cela peut être un nouveau document XML ou tout autre formalisme, mais également l'extraction des valeurs du document XML lié au *nœud contextuel*, ainsi que les appels à d'autres règles de la feuille XSL. Un nœud contextuel est un élément du document XML en cours de traitement dans la règle de transformation.

L'exemple de la figure **28** donne une règle qui s'applique sur la balise « livre » (match= « livre »). Pour cet élément contextuel, l'élément « livre » trouvé dans le document XML source, la transformation va produire en sortie le formalisme xHTML. Ainsi, l'application de cette règle donnera en sortie les balises nécessaires à l'affichage du document dans un navigateur web.

```
<xsl:template match="livre">
    <html>
      <head>
          <title><xsl:value-of select="@titre"/></title>
      </head>
      <body>
      <h1 >
            Titre : "<xsl:value-of select="@titre"/>"
      </h1>
          <xsl:apply-templates select="./chapitre"/>
      </body>
    </html>
</xsl:template>
```

Figure 28 – Exemple de règle de transformation du livre en HTML

Nous pouvons constater également que le titre du livre est extrait de notre nœud contextuel (l'élément « livre ») grâce à la séquence « *xsl:value-of* » avec l'expression de chemin « select » qui va extraire l'attribut « titre » pour mettre sa valeur dans la balise de sortie « title ». Par ailleurs, dans le corps du document de sortie « body », le titre est également ajouté dans une balise « <h1> » et un appel à d'autres règles est appliqué avec l'élément « *xsl:apply-templates* ». Celui-ci s'applique sur chacun des éléments fils de l'élément « livre ».

Pour fonctionner, le moteur de règles XSLT charge l'ensemble des règles définies dans la feuille XSL. Ensuite, il va charger le document XML et lors de sa lecture il va appliquer les règles en suivant la logique suivante :
1/ Débuter avec la règle « / » pour la racine du document.
2/ Dans le corps de chaque règle, définir les éléments qui doivent déclencher une règle (ne pas définir la règle à appeler, seulement les éléments à traiter).

Ainsi, la règle précédente est appelée lorsque la règle racine fait appel (via « *xsl:apply-templates* ») à son élément fils : « <livre> ». Cette règle fera ensuite appel aux éléments fils, chacun des éléments « <chapitre> ». La règle de la figure **29** serait alors appelée par XSLT à partir du moment où l'attribut « *match* » contient la définition des éléments fils de « livre ».

Nous pouvons voir que les informations de l'élément « chapitre » sont utilisées pour afficher les informations dans la balise XHTML « <h1> » (identifiant, titre, page et sa position par rapport à l'élément « titre ») et que les paragraphes sont intégrés soit par le texte intégré, soit via appel d'une nouvelle règle (appliquée sur les éléments « <para> » qui composent le chapitre).

Il est possible de surcharger une règle sur un élément dans le cas où certaines particularités seraient applicables. Il est ainsi possible de rajouter des filtres ou n'importe quelle expression XPath (voir § 6.2) pour raffiner la règle. Dans le cas de litige avec plusieurs règles applicables sur un même élément comme dans l'exemple de la figure **30** combiné avec celui de la figure **29**, la règle la plus précise est appliquée, ici le filtre sur le chapitre. Il est également possible de définir une priorité sur la règle avec l'attribut « *priority* ».

La difficulté de l'apprentissage de XSLT réside dans l'application des règles et son enchaînement. Ainsi, il n'est pas forcément évident de constater que la règle « chapitre » (figure **29**) sera appliquée à chaque élément « chapitre » grâce à l'appel « *apply-template* » de la règle « livre » (figure **28**). C'est la manière de procéder de XSLT pour les transformations.

Lorsque l'on souhaite appliquer un aspect plus procédural à l'appel de règle, des structures plus classiques sont disponibles :

– « *xsl:if* » pour les structures conditionnelles ;
– « *xsl:for-each* » pour les boucles ;
– « *xsl:choose* » pour des structures conditionnelles énumératives ;
– « *xsl:variable* » pour définir des variables internes à la règle.

NB : Il est à noter que la structure « *xsl:for-each* » se comporte de manière similaire à « *xsl:template* ». Le « *for-each* » a une approche programmative tandis que « *template* » applique une règle. Le résultat est similaire, mais le contexte du document XML est dans ce cas différent.

Pour tester XLST, il est possible d'utiliser l'outil en ligne proposé par w3schools (https://www.w3schools.com/xml/tryxslt.asp?xmlfile=cdcatalog&xsltfile=cdcatalog_if). Remplacez le document XML de gauche par notre exemple complet (figure **4**), et dans le texte de droite, vous y mettrez la structure générique de XSLT (figure **26**), la règle pour l'élément « livre » (figure **28**) et la dernière règle avec les boucles et les conditions (figure **29**). Cela produit le document XSLT de la figure **31**.

Ainsi, grâce aux documents XSLT, il est possible de transformer n'importe quel document XML dans un nouveau format. Cela peut servir à formater l'information en fonction du support nécessaire en sortie sans avoir à définir la transformation d'un format

```
<xsl:template match="chapitre">
   <h2 ID="{@chapID}">
      Chapitre <xsl:value-of select="position()"/> :
      <xsl:value-of select="@titre"/> - page <xsl:value-of select="@page"/>
   </h2 >
   <xsl:apply-templates />
</xsl:template >
```

Figure 29 – Exemple de règle pour les chapitres

```
<xsl:template match="chapitre[1]">
   <h2 ID="{@chapID}">
      Premier chapitre :
      <xsl:value-of select="@titre"/> - page<xsl:value-of select="@page"/>
   </h2 >
   <xsl:apply-templates/>
</xsl:template >
```

Figure 30 – Exemple de surcharge de règle

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="livre">
   <html>
      <head>
         <title><xsl:value-of select="@titre"/></title>
      </head>
      <body>
         <h1>Titre : "<xsl:value-of select="@titre"/>"</h1>
         <xsl:apply-templates select="./chapitre"/>
      </body>
   </html>
</xsl:template>

<xsl:template match="chapitre">
   <h2 ID="{@chapID}">
         Chapitre <xsl:value-of select="position()"/> :
      <xsl:value-of select="@titre"/> - page<xsl:value-of select="@page"/>
   </h2>
   <xsl:for-each select = "*">
      <xsl:choose>
         <xsl:when test = "name(.) = 'para'">
            <p><xsl:value-of select="text()"/></p>
         </xsl:when>
         <xsl:when test = "name(.) = 'saut'">
            <hr/>
         </xsl:when>
         <xsl:otherwise>
            <xsl:value-of select="text()"/>
         </xsl:otherwise>
      </xsl:choose>
   </xsl:for-each>
   <a href="#{@chapID}">Début du chapitre</a><hr/>
</xsl:template>
</xsl:stylesheet>
```

Figure 31 – Exemple complet de document XSLT

à un autre (PDF vers Word par exemple), le format XML sert de pivot.

La documentation XSLT 3.0 propose de nombreux éléments pour définir les règles de transformation (https://www.w3.org/TR/xslt-30/). Notamment, nous y trouverons le « *xsl:streaming* » qui a été introduit pour permettre le traitement de flux XML. Ce flux XML n'est pas un document complet puisqu'il n'a pas de fin, le document XSLT correspondant définit ainsi la manière de traiter ce flux avec des événements.

Tout langage de programmation intègre une librairie permettant d'intégrer XSLT pour transformer un document XML dans le format de destination. En ligne de commande, vous pouvez également utiliser xsltproc2 (http://xmlsoft.org/XSLT/xsltproc2.html, voir Section logiciels et outils du Pour en savoir plus).

5.2.4 Dialectes XML

De nombreux formats ont été proposés avec un schéma XSD pour standardiser certains types de données. Ces formats sont des *dialectes* conformes aux règles syntaxiques XML et les documents sont donc manipulables avec les interfaces de programmation standard XML. Ainsi, les systèmes deviennent interopérables, car ils partagent la même définition de documents pour un contexte donné (image vectorielle, math, musique, etc.). Nous allons donner ici un aperçu des différents formats existants avec leurs buts et références.

Flux RSS

Le RSS est un format d'échange d'informations produites par des sites web apparu en 1999 à l'initiative de Dave Winer, sous un formalisme XML. On peut retrouver deux formats principaux avec RSS (https://www.cyber.harvard.edu/rss/rss.html#roadmap) et Atom (https://www.tools.ietf.org/html/rfc4287). Ce format est utilisé particulièrement dans les podcasts (mise à jour de contenus multimédias) ou notifications d'articles de presse en ligne. RSS n'étant pas un standard, il n'est pas associé à un espace de nommage et de fait n'a pas de préfixe avec un *namespace*.

L'exemple de la figure **32** montre le contenu d'un flux RSS, et son rendu visuel simple, produits par un site web (example.org) contenant 2 items. La notion de « flux » est une vision du document XML par effet de mises à jour de celui-ci. Ainsi, l'apparition de nouveaux items, ajoutés au début du document XML, sera détectée par le lecteur de flux RSS pour produire un effet de notification à l'utilisateur.

OOXML et OpenDocument

Office Open XML (OOXML – http://www.officeopenxml.com/) est une norme créée par Microsoft visant l'interopérabilité des environnements et permettant de faciliter l'intégration dans différents logiciels de bureautique (docx, xlsx, pptx) (figure **33**).

OpenDocument (https://www.libreoffice.org/discover/what-is-opendocument/) est son pendant libre avec LibreOffice (Apache / The Document Foundation). Les suites bureautiques intégrées LibreOffice et Apache OpenOffice sont capables de lire ce type de formats.

SVG

Le Scalable Vector Graphics (SVG – https://www.w3.org/Graphics/SVG/) est un format permettant de représenter des images vectorisées. Il permet la spécification de figures et l'intégration d'images (ou flux multimédia avec SMIL – https://www.w3.org/TR/REC-smil/). Il est reconnu par la plupart des navigateurs web et logiciels graphiques.

L'exemple de la figure **34** illustre l'écriture d'un texte incurvé colorisé et vectorisé.

Vous pouvez le tester et créer vos propres images ici :

https://www.w3schools.com/code/tryit.asp?filename=G98BPY6I43ZZ

MathML

MathML (https://www.w3.org/Math/) permet de formaliser des formules et équations mathématiques pour pouvoir les afficher, les combiner, les calculer, etc. Le rendu visuel est ainsi calculé automatiquement

```
<?xml version="1.0" encoding="UTF-8"?>
<rss version="2.0">
   <channel>
      <title>Mon site Web</title>
      <description>Exemple de flux RSS v2.0
         produisant des informations</description>
      <lastBuildDate>Tue, 22 Oct 2019 14:50:00
                     GMT</lastBuildDate>
      <link>http://www.example.org</link>
      <item>
            <title>Dernier élément</title>
            <description>Actualité récente</description>
            <pubDate>Tue, 22 Oct 2019 14:50:00
                     GMT</pubDate>
            <link>http://www.example.org/actu52</link>
      </item>
      <item>
            <title>Avant dernier élément</title>
            <description>Cette actualité est apparue
                     juste avant</description>
            <pubDate>Tue, 21 Oct 2019 19:00:30
                     GMT</pubDate>
            <link>http://www.example.org/actu51</link>
      </item>
   </channel>
</rss>
```

Rendu visuel

Dernier élément
Tue, 22 Oct 2019 14:50:00 GMT

Actualité récente

Read More

Avant dernier élément
Tue, 21 Oct 2019 19:00:30 GMT

Cette actualité est apparue juste avant

Read More

Figure 32 – Contenu d'un flux RSS et son rendu visuel

```
...
<w:body>
   <w:p w:rsidR="00CF2FC0" w:rsidRDefault="004A0488">
      <w:r><w:t>Titre du document</w:t></w:r>
   </w:p>
   <w:p w:rsidR="004A0488" w:rsidRDefault="004A0488"
w:rsidP="004A0488">
   <w:r><w:t>Le contenu…</w:t></w:r>
   <w:bookmarkStart w:id="0" w:name="_GoBack"/>
      <w:bookmarkEnd w:id="0"/>
   </w:p>
   <w:sectPr w:rsidR="004A0488" w:rsidRDefault="00ED3940">
      <w:pgSz w:w="11900" w:h="16840"/>
      <w:pgMar w:top="1417" w:right="1417" w:bottom="1417"
w:left="1417"
       w:header="708" w:footer="708" w:gutter="0"/>
      <w:cols w:space="708"/>
      <w:docGrid w:linePitch="360"/>
   </w:sectPr>
</w:body>
...
```

Rendu visuel

Titre du document
Le contenu...

Figure 33 – Office Open XML et son rendu visuel

```
<svg xmlns="http://www.w3.org/2000/svg"
      xmlns:xlink="http://www.w3.org/1999/xlink">
      <defs>
            <path id="myTextPath"
                  d="M110,20 a1,1 1 1,0 60,0"/>
      </defs>
      <text x="0" y="100"
            style="font-family: Arial;
                  font-size : 15;
                  stroke : #000000;
                  fill : #00ff00;">
            <textPath xlink:href="#myTextPath">
                  Image en SVG
            </textPath>
      </text>
</svg>
```

Figure 34 – Écriture d'un texte incurvé colorisé et vectorisé en SVG

(https://www.tutorialspoint.com/online_mathml_editor.php) et peut être utilisé ou converti dans différentes applications compatibles (Web, doc, TeX, etc.). L'extension MathJax (https://www.docs.mathjax.org/en/latest/basic/mathjax.html) permet de pousser le concept de formule de manière dynamique et interactive.

L'exemple de la figure **35** donne la formule de l'écart-type. Vous pouvez constater la présence d'une racine carrée (msqrt), d'une fraction (mfrac), d'une somme (&#x2211), du carré (msup), etc.

MusicXML, MEI

Les deux formats MusicXML (https://www.musicxml.com) et MEI (https://www.music-encoding.org) permettent de définir une notation pour partitions musicales. Alors que MusicXML est très strict et place le rendu visuel en avant, MEI s'intéresse à la notation et le rendu visuel est laissé à l'interprétation de l'application. Bien que MusicXML reste un format plus répandu, des transformations automatiques avec XSLT2.0 existent. On peut retrouver des corpus libres d'accès contenant des transcriptions de partitions telles que NEUMA (http://www.neuma.huma-num.fr/home/). De nombreux lecteurs MusicXML existent tels que Finale (https://www.finalemusic.com/), Sibelius (https://www.avid.com/sibelius) ou Music21 en python (http://www.mit.edu/music21/).

L'exemple de la figure **36** correspond aux deux formats de l'Agnus Dei de Palestrina, ainsi que son rendu visuel.

5.2.5 RDF et RDFs : vers la gestion des connaissances

Le XML est donc un puissant outil de structuration de données et de documents de tous types. À la fin du XX^{e} siècle, l'enjeu de la documentation des corpus présents sur le Web est devenu crucial.

```
<math display="block">
  <mi>&#x3C3;</mi>
  <mo>=</mo>
  <msqrt>
    <mfrac>
      <mn>1</mn><mi>N</mi>
    </mfrac>
    <munderover>
      <mo>&#x2211;</mo>
      <mrow class="MJX-TeXAtom-ORD">
        <mi>i</mi><mo>=</mo><mn>1</mn>
      </mrow>
      <mi>N</mi>
    </munderover>
    <mo stretchy="false">(</mo>
    <msub>
      <mi>x</mi><mi>i</mi>
    </msub>
    <mo>&#x2212;</mo>
    <mi>&#x3BC;</mi>
    <msup>
      <mo stretchy="false">)</mo>
      <mn>2</mn>
    </msup>
  </msqrt>
</math>
```

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$$

Figure 35 – Écriture de la formule de l'écart-type en MathML

5.2.5.1 Dublin Core : des données sur les données

Des initiatives, comme le « Dublin Core Metadata Initiative » (1995), ont tenté de relever le défi à la fois conceptuellement et techniquement. L'idée principale de cet effort de normalisation était d'uniformiser – en plus des données – les métadonnées et leurs usages au sein du monde entier afin d'éviter que le Web ne se transforme en tour de Babel informationnelle. L'idée initiale était de proposer pour une ressource (ou sujet), d'en définir les propriétés (ou prédicats) au moyen d'objets (valeur de la propriété) (tableau **2**).

Originellement avec le Dublin Core, chaque page ressource (x)HTML pouvait être décrite dans son en-tête au moyen de 13 puis 15 propriétés qui en définissent le titre, le ou les auteurs, les sujets traités, la date de publication, la langue (figure **37**), etc. Il est à noter également qu'il était possible de définir un identifiant et une relation à une autre ressource, comme ici l'identifiant ISSN de la revue Techniques de l'Ingénieur. Ce vocabulaire s'est ensuite étendu avec le Dublin Core qualifié dans lequel chacun des éléments peut être spécifié. La BnF précise également que le Dublin Core est indépendant des formats d'encodage et de stockage de l'information (Dublin Core qualifié : https://www.bnf.fr/fr/dublin-core#bnf--dublin-core-qualifi-qualified-dublin-core-). Il peut donc s'implémenter dans les formes (x)HTML, mais aussi comme descripteur de données en XML et ses dialectes.

5.2.5.2 Concept de RDF : la notion de triplet

Pour implémenter un système générique de description de données ou métadonnées qui soit indépendant du support, un modèle de description en graphe a été pensé : le **Resource Description Framework (RDF)**. Ce modèle, initialement décorrélé de toute implémentation technique fonctionne sur un système grammatical simple : le triplet. Le triplet est la base du Web dit « sémantique », il s'agit conceptuellement de l'association d'un *sujet* et d'un *objet* au travers d'un *prédicat*. L'association (*sujet*, *prédicat*, *objet*) peut se comprendre, pour être triviale comme une phrase grammaticalement simple : sujet, verbe, complément.

Pour reprendre l'exemple du Dublin Core, il est possible de décrire la phrase suivante : « L'article 'The Semantic Web' a été écrit par Tim Berner Lee » par deux valeurs littérales de type chaîne de caractères reliées par une URL descriptive (tableau **3**).

Dans le tableau **4**, l'article 'The Semantic Web' est représenté par son URL officielle, la relation d'attribution de la paternité, elle aussi est signifiée par l'URL Dublin Core. Enfin Tim Berner Lee est représenté par l'URL de sa notice d'autorité de l'ISNI.

En utilisant des autorités d'identification comme l'ISNI ou VIAF (personnes) et DOI (identifiant d'objets numériques, souvent utilisé en recherche) il est possible d'éviter toute ambiguïté sur l'article décrit et sur la personne (tableau **5**).

5.2.5.3 RDF : le langage de description en XML

Turtle et N3

L'instanciation du concept de RDF peut se faire sous différentes formes. Des fichiers plats comme le turtle (*Terse RDF Triple Language*), sous-ensemble de la Notation 3 (N3), vont sérialiser le RDF de manière simple, compréhensible et compacte, sans utiliser de XML. L'exemple précédent pourra être sérialisé ainsi de plusieurs manières en Turtle :

```
<http://www.jstor.org/stable/26059207>
<http://purl.org/dc/elements/1.1/creator>
<http://www.isni.org/0000000124514311>
```

MusicXML

```
…
<part id= "P1">
   <measure number="1" width="389">
      <print page-number="1" >
         <system-layout>
            <system-margins>
               <left-margin>60</left-margin>
               <right-margin>0</right-margin>
            </system-margins>
            <top-system-distance>180
               </top-system-distance>
         </system-layout>
         <measure-numbering>system
            </measure-numbering>
      </print>
      <attributes>
         <divisions>2</divisions>
         <key>
            <fifths>0</fifths>
            <mode>major</mode>
         </key>
         <time>
            <beats>4</beats>
            <beat-type>2</beat-type>
         </time>
         <clef>
            <sign>G</sign>
            <line >2< /line >
         </clef>
      </attributes>
      <sound tempo="120"/>
…
```

MEI

```
…
<section xml:id="m-46">
   <sb xml:id="m-48"/>
      <measure xml:id="m-47" label="1" n="1">
         <staff xml:id="m-49" n="1">
            <layer xml:id="m-50" n= "1">
               <note xml:id="m-51" dur="1"
                     dur.ges="1024p" oct="4"
                     pname="g" pnum="67"
                     stem.dir="up"/>
               <note xml:id="m-52" dur="1"
                     dur.ges="1024p" oct="5"
                     pname="d" pnum="74"
                     stem.dir="down"/>
            </layer>
         </staff>
…
```

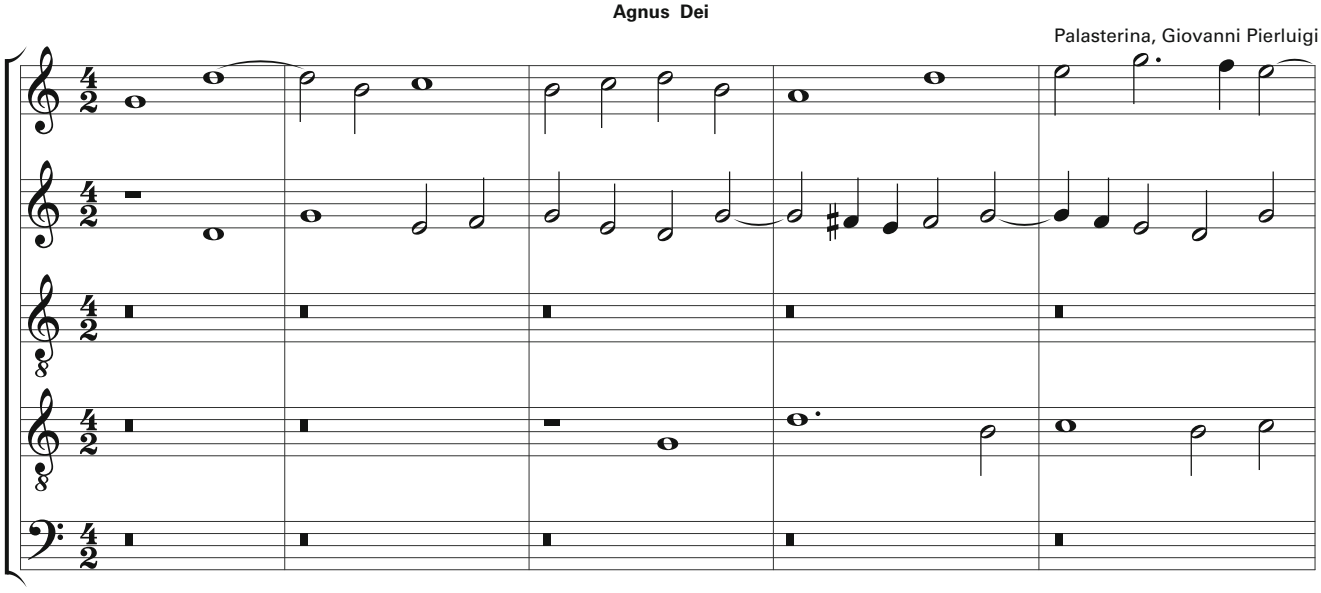


Figure 36 – Rendu visuel de notation musicale avec MusicXML / MEI

Tableau 2 – Les éléments de description du Dublin Core simple

Élément	Description
Title	Nom de la ressource
Creator	Auteur(s) de la ressource
Subject	Descripteurs de la ressource
Description	Présentation du contenu de la ressource (résumé, table des matières, représentation graphique du contenu, texte libre)
Publisher	Nom de l'entité morale ou physique responsable de la publication
Contributor	Personne(s) morale(s) ou physique(s) ayant contribué à la ressource : traducteur, graphiste...
Date	Date de création ou de mise à disposition de la ressource
Type	Nature de la ressource
Format	Encapsulation technique de la ressource présentée
Identifier	Identifiant unique de la ressource
Source	Lien à une autre ressource de référence
Language	Langue de la ressource
Relation	Lien à une autre ressource sans qu'il s'agisse d'une source de référence
Coverage	Couverture géographique ou temporelle d'une ressource
Rights	Droits associés à la ressource d'usage et/ou de diffusion)

ou encore en convoquant un vocabulaire, une URI et un littéral :

```
@prefix dc: <http://purl.org/dc/elements/1.1/> .
<www.jstor.org/stable/26059207> dc:creator "Tim Berners Lee" .
```

RDF-XML

Le concept de RDF va trouver également une instanciation en XML : le XML RDF ou plus simplement le RDF, ce qui explique la confusion qui existe entre le concept et sa possible sérialisation XML. La figure **38** présente l'exemple RDF du tableau **4** du § 5.2.5.2 en XML. Et la figure **39** présente l'écriture en XML du tableau **5**.

5.2.5.4 RDF : la gestion des connaissances

L'usage des langages de description trouve une place importante dans les grandes organisations pour la gestion des archives (SAE) et pour la gestion informatisée des documents électroniques (GED). Des langages spécifiques sont créés pour la gestion des terminologies, des connaissances.

5.2.5.5 SKOS : le langage des thésaurus

Les thésaurus permettent de documenter et d'indexer des ressources en les alignant sur un vocabulaire de description. Ils sont souvent modélisés en *Simple Knowledge Organization System* (SKOS) qui est une implémentation du RDF orientée documentation (https://www.w3.org/TR/swbp-skos-core-spec/). Ce langage permet de maintenir un vocabulaire de description avec des relations de synonymie, de hiérarchie et de champ sémantique. Le SKOS permet donc de relier et de faire correspondre des concepts issus de systèmes d'information différents, possiblement en langues différentes. Il est à noter que l'association *American Computer Machinery* ACM a utilisé le langage SKOS comme modèle pour sérialiser l'« *ACM Computing Classification System* », leur thésaurus qui est la référence dans les milieux scientifiques informatiques. L'ACM CCS a permis de gérer les connaissances contenues dans l'immense bibliothèque numérique de revues et d'actes de conférences de l'ACM entre 1998 et 2012. Ce système a été remplacé par une ontologie depuis 2012 (voir § 5.2.5.6).

Dans l'exemple de la figure **40**, on s'inspire des éléments du thésaurus spécialisé en archéologie « Pactols » pour décrire le concept de « critique d'Art » en utilisant la syntaxe RDF et plus spécifiquement l'espace de nommage SKOS. Le terme concept de « critique d'Art » est représenté ici, selon le système d'identification pérenne Ark (*Archival Resource Key* – https://www.fr.wikipedia.org/wiki/Archival_Resource_Key), par l'URI <https://www.ark.frantiq.fr/ark:/26678/pcrt8e84y5qvKb>. Ce terme a pour définition « Art de juger les œuvres artistiques », qui a pour synonymes « Fortune critique » et « Ekphrasis ». On note cependant que le terme « officiel » dans ce thésaurus est bien « Critique d'Art ». Ce terme peut être généralisé

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
   <head profile="http://dublincore.org/documents/2008/08/04/dc-html/">
      <title>Notre article sur le XML</title>
      <meta http-equiv="Content-type"content="text/html; charset=utf-8" />
      <!-- Appel au schéma de description dublincore -->
      <link rel="schema.DC" href="http://purl.org/dc/elements/1.1/" />
      <link rel="schema.DCTERMS" href="http://purl.org/dc/terms/" />
      <meta name="DC.title" lang="fr" content="Notre article sur le XML" />
      <meta name="DC.date" scheme="DCTERMS.W3CDTF" content="2020-04-01" />
      <meta name="DC.subject" lang="fr" content="XML, document, Format" />
      <meta name="DC.description" lang="fr" content="Article TI sur le XML" />
      <link rel="DC.source" href="urn:ISSN:0245-9612" />
   </head>
   <body>...</body>
</html>
```

Figure 37 – Code source d'une page xHTML auto-décrite par le Dublin Core simple

Tableau 3 – Description d'une phrase en RDF		
Sujet	Prédicat	Objet
« The Semantic Web »	<http://www.purl.org/dc/elements/1.1/creator>	« Tim Berner Lee »

Tableau 4 – Représentation RDF		
Sujet	Prédicat	Objet
<http://www.jstor.org/stable/26059207>	<http://www.purl.org/dc/elements/1.1/creator>	<http://www.isni.org/0000000124514311>

Tableau 5 – Représentation RDF sans ambiguïté		
Sujet	Prédicat	Objet
urn:doi:10.1.1.115.9584	dc:creator	urn:ISNI:0000000124514311

(balise <skos:broader>) par le concept « art » ici représenté par la ressource <https://www.ark.frantiq.fr/ark:/26678/pcrttq9gp2ZMu> du thésaurus Pactols.

NB : Archéologie Pactols, SKOS : https://www.w3.org/TR/swbp-skos-core-spec/.
La présentation du thésaurus Pactols proposé par le groupe FRANTIQ (Fédération et ressources sur l'Antiquité) du CNRS : https://www.masa.hypotheses.org/pactols

Le rendu va être réalisé en intégrant le code SKOS dans le convertisseur visuel Skosplay (http://www.labs.sparna.fr/skos-play/).

5.2.5.6 RDFs : le langage des taxonomies

Les taxonomies, ou taxinomies, sont des hiérarchies conceptuelles qui peuvent être modélisées en RDF Schema ou RDFs (https://www.w3.org/TR/rdf-schema/), une spécification conceptuelle du RDF. Il s'agit donc d'un vocabulaire pour décrire des vocabulaires de descriptions... Le RDFs peut se sérialiser en XML ou en turtle et propose des classes, des sous-classes et des propriétés. Le RDFs pourra ainsi décrire les possibles propriétés des contenus décrits dans des fichiers RDF sous la forme de « classes » (*Class*) et de « sous-classes » (*SubClassOf*). Le RDFs mobilise aussi des éléments de vocabulaires RDF comme des « propriétés » (*Property*) qui seront des instances de classes. On trouve plusieurs types d'attributs triviaux comme « *label* » et « *comment* », ou beaucoup moins triviaux comme « *range* » et « *domain* ». Ces derniers vont permettre de définir les règles de transitivité entre les propriétés des classes en cas d'appartenances multiples et définir les inférences possibles. Dans l'exemple de la figure **41**, nous avons généré un graphe pour faciliter la compréhension (RDF Grapher : http://www.ldf.fi/service/rdf-grapher).

Dans la figure **41**, la taxonomie décrit qu'une *Voiture*, une *Personne* et un *Vehicule* sont des classes. Un conducteur est une propriété de la classe *Personne* qui est aussi du domaine de la classe *Vehicule*. Un conducteur est donc une *Personne* et fait également partie des propriétés (est lié). *Range* et *domain* servent à restreindre l'ensemble des ressources pouvant avoir une propriété donnée (*domain* de la propriété) et l'ensemble des valeurs valables pour une propriété (*range*). Une propriété peut avoir autant de valeurs pour « rdfs:domain » que nécessaire, mais pas plus d'une valeur pour « rdfs:range » (https://www.w3.org/TR/rdf-schema/#ch_domainrange). Un conducteur pourrait être lié à plusieurs instances de la classe *Voiture*, mais à une seule instance de la classe *Personne*.

On peut donc y voir une manière d'établir des cardinalités.

Une extension du RDFs, le Ontology Web Language (OWL – https://www.w3.org/TR/owl2-overview/) permet de modéliser des systèmes complexes, avec plus d'expressivité que le RDFs (Voir section Normes et standard, partie W3C). Ce langage est utilisé principalement dans les systèmes de pointe, complexes à modéliser comme dans le domaine médical ou de l'industrie. L'outil (graphique) (figure **42**) dédié à la modélisation des ontologies est protégé et développé par le *Center for Biomedical Informatics Research* de l'université de Stanford (Protégé : https://www.protege.stanford.edu/).

La partie 6 de cet article sera consacrée à la manipulation et à l'interrogation des structures XML, nous étudierons le langage de requête orienté « graphe » spécifique au RDF appelé SparQL.

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:dc="http://purl.org/dc/elements/1.1/">
   <rdf:Description rdf:about="http://www.jstor.org/stable/26059207">
      <dc:title>The Semantic Web</dc:title>
      <dc:creator>Tim Berners Lee</dc:creator>
   </rdf:Description>
</rdf:RDF>
```

Figure 38 – RDF XML

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:dc="http://purl.org/dc/elements/1.1/">
   <rdf:Description rdf:about="http://www.jstor.org/stable/26059207">
      <dc:title>The Semantic Web</dc:title>
      <dc:creator rdf:resource="http://www.isni.org/0000000124514311"/>
   </rdf:Description>
</rdf:RDF>
```

Figure 39 – RDF XML avec ISNI

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:skos="http://www.w3.org/2004/02/skos/core#">
  <skos:Concept rdf:about="https://ark.frantiq.fr/ark:/26678/pcrt8e84y5qvKb">
    <skos:broader rdf:resource="https://ark.frantiq.fr/ark:/26678
        /pcrttq9gp2ZMuc"/>
      <skos:prefLabel>Critique d'Art</skos:prefLabel>
    <skos:altLabel>Fortune Critique</skos:altLabel>
    <skos:altLabel>Ekphrasis</skos:altLabel>
    <skos:inScheme rdf:resource="https://pactols.frantiq.fr/opentheso/"/>
    <skos:definition>Art de juger les oeuvres artistiques</skos:definition>
  </skos:Concept>
</rdf:RDF>
```

Exemple de (micro) thésaurus en SKOS

- **Critique d'Art**
  EP : *Fortune Critique*
  EP : *Ekphrasis*
  DEF : Art de juger oeuvres artistiques
  TG : **https://ark.frantiq.fr/ark:/26678/pcrttq9gp2ZMuc**
  TT : **https://ark.frantiq.fr/ark:/26678/pcrttq9gp2ZMuc**
- ~~*Ekphrasis*~~
  EM : **Critique d'Art**
- ~~*Fortune Critique*~~
  EM : **Critique d'Art**

Figure 40 – Exemple de (micro)thésaurus en SKOS

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
  <rdfs:Class rdf:ID="Personne">
      <rdfs:label>Classe personne</rdfs:label>
  </rdfs:Class>
  <rdfs:Class rdf:ID="Vehicule">
      <rdfs:label>Classe vehicule</rdfs:label>
  </rdfs:Class>
  <rdfs:Class rdf:ID="Voiture">
    <rdfs:comment>Classe voiture</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Vehicule"/>
  </rdfs:Class>
  <rdf:Property rdf:ID="conducteur">
    <rdfs:domain rdf:resource="#Vehicule"/>
    <rdfs:range rdf:resource="#Personne"/>
    <rdfs:comment>Propriété de Voiture et de Personne</rdfs:comment>
  </rdf:Property>
</rdf:RDF>
```

Figure 41 – Exemple description de taxonomie en RDF schema (RDFs)

5.2.5.7 Les autres...

La liste des formalismes XML est longue. On peut citer : RelaxNG (description de documents XML – https://www.relaxng.org/), DockBook/TEI (représentation de texte – https://tei-c.org/), OAI (Open Archives Initiative – https://www.openarchives.org/), KML (données géographiques – https://developers.google.com/kml/documentation/?csw=1).

Une liste non officielle est disponible sur Wikipédia :

https://en.wikipedia.org/wiki/List_of_XML_markup_languages

5.3 Partage de données

5.3.1 Services web, SOAP et WSDL

Afin de rendre les systèmes d'information interopérables et consistants, les services web définissent un formalisme XML permettant d'échanger des messages et des données de manière sécurisée et formatée. Pour cela, nous nous reposerons sur le protocole d'accès à un service SOAP, et WSDL pour faire la découverte de services web.

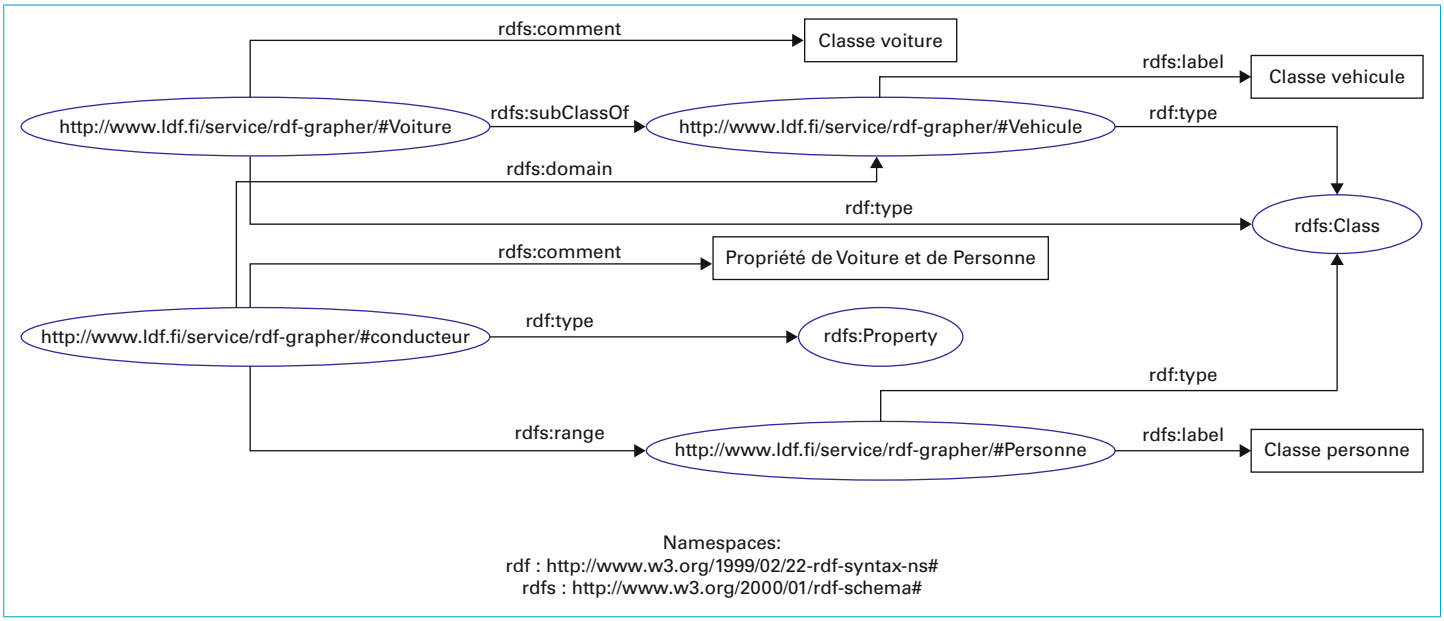


Figure 42 – Modélisation des ontologies par Protégé

SOAP (Simple Object Access Protocol – http://www.w3.org/TR/SOAP/https://xml.developpez.com/soap.php) est donc un formalisme pour décrire les méthodes d'appels de fonctions à distance (RPC). Il décrit les fonctions disponibles, leurs paramètres ainsi que leur typage, les contraintes associées à l'appel, etc. Ce formalisme est intégré à différents langages de programmation tels que Java ou .NET. Il est donc possible de générer un serveur web avec différents appels de services directement sur les fonctions implémentées dans une classe Java. Pour cela, il est nécessaire d'installer le serveur Axis (http://www.axis.apache.org/axis/java/user-guide.html) qui permet l'intégration de votre programme dans cet environnement web.

Comme exemple de service web, nous pouvons citer les plateformes de réservation de billets (avions, spectacles, sport, etc.). Prenons les avions, chaque compagnie aérienne gère sa flotte et l'ensemble des vols et réservations. Toutefois, la plateforme de réservation ne gère aucun vol, elle n'est qu'un intermédiaire pour trouver des places disponibles parmi toutes les compagnies aériennes. Le service web intervient dans le fait que la plateforme communique via ces services dédiés à la consultation/réservation pour trouver tous les vols éligibles. Chaque compagnie offre ainsi un service web sur lequel il faut échanger des documents XML valides pour effectuer la transaction.

WSDL (*Web Services Description Language*) de son côté est un formalisme XML pour fournir les spécifications du service web pour que les messages envoyés soient au format XML requis. Il suffit alors de publier le document WSDL sur un serveur UDDI (*Universal Description Discovery and Integration*). Celui-ci joue le rôle d'annuaire de services pour retrouver les documents WSDL correspondant aux critères choisis, ici « compagnie aérienne : consultation & réservation ».

La figure **43** illustre l'échange de messages effectué dans une architecture services web. Chaque service web correspond à un programme développé chez le fournisseur, celui-ci est alors enregistré dans le serveur Axis, pour être interrogé, mais également transformé en document XML WSDL pour décrire les paramètres du service. Le serveur UDDI reçoit le document WSDL, *via* une requête SOAP. De son côté, le client peut chercher les services qui l'intéressent dans le serveur UDDI, *via* requête SOAP. Lorsqu'il trouve le service concerné, il génère un client SOAP grâce au descripteur WSDL. Le client SOAP peut alors envoyer des messages SOAP avec les bons paramètres et recevoir les messages de réponses en SOAP.

5.3.2 Fichiers de données (dump de base)

Les bases de données offrent l'opportunité d'effectuer des extractions de tout ou partie d'une base en divers formats. Ces extractions, aussi appelées « dumps », vont permettre de travailler sur les contenus de la base de manière asynchrone, ou encore d'en faire une sauvegarde. Pour des sauvegardes de tables, les formats d'exports qui viennent à l'esprit sont le SQL ou le CSV, mais il est également possible d'exporter une table, une vue ou une base complète au format XML. Évidemment, cette méthode testée sous mysql avec mysqldump en ligne de commande et phpMyAdmin pour une surcouche web, génère des fichiers extrêmement volumineux par rapport à leur équivalent en SQL.

ex : la commande mysqldump, usage :

```
mysqldump --xml -u user -p password mabase > dump.xml
```

Dans l'exemple de la figure **44**, la commande mysqldump avec l'argument --xml va générer une sauvegarde de toute la base « mabase » au format XML avec une racine de type <database>. La redirection de sortie standard permettra de générer un fichier nommé « dump.xml », qui va archiver le résultat de de la requête.

La figure **45** présente la table archivée (capture de la vue « concepteur » de phpMyAdmin) et ses deux contenus.

Mysqldump a généré un code XML pour la base et pour chaque table on trouvera :

– un objet de type <table_structure> qui servira à décrire les colonnes de la table et à en définir les clés (transcription des informations de la base) ;

– un objet de type <table_data> qui contiendra des lignes de la table ;

– les contenus eux-mêmes sous la forme de lignes <raw>, elles-mêmes composées de champs <field>.

Il est à noter en ligne 2 (figure **44**) qu'un schéma W3C est utilisé pour assurer l'universalité de l'export (voir § 4).

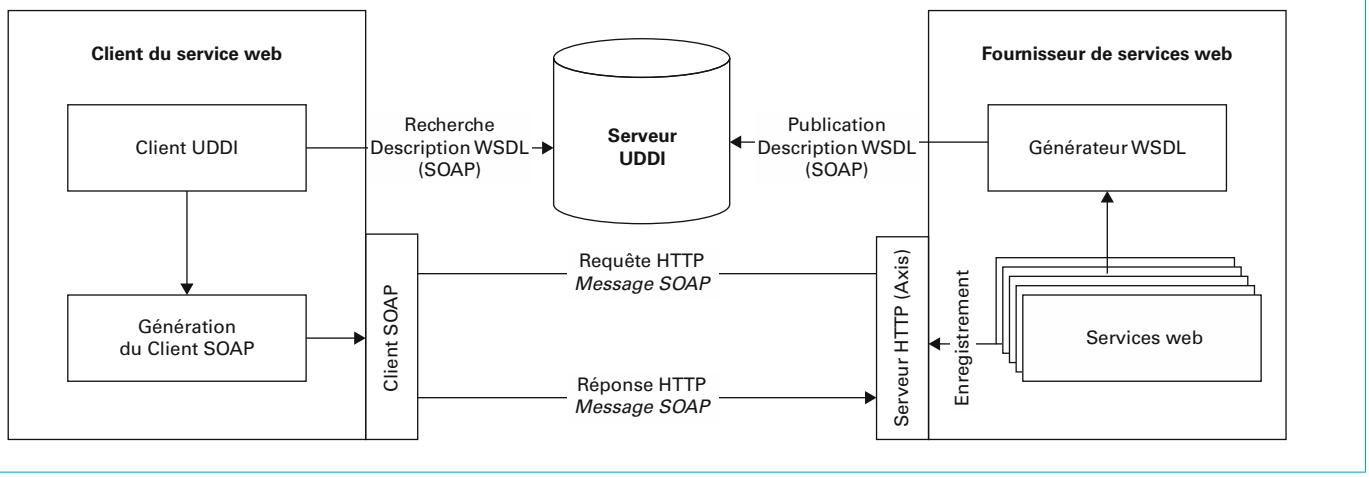


Figure 43 – Architecture des services web (SOAP/WSDL/UDDI)

```
<?xml version='1.0' encoding='utf-8'?>
<mysqldump xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<database name="ouvrages">
  <table_structure name="auteurs">
    <field Field="id" Type="int(11)" Null="NO" Key="PRI"
           Extra="auto_increment" />
    <field Field="nom" Type="varchar(45)" Null="NO" />
    <field Field="prenom" Type="varchar(45)" Null="NO" />
    <field Field="annee_naissance" Type="year(4)" Null="NO" />
    <field Field="annee_mort" Type="year(4)" Null="NO" />
    <key Table="auteurs" Non_unique="0" Key_name="PRIMARY"
         Seq_in_index="1" Column_name="id" />
  </table_structure>
  <table_data name="auteurs">
    <row>
      <field name="id">1</field>
      <field name="nom">Sartre</field>
      <field name="prenom">Jean-Paul</field>
      <field name="annee_naissance">1905</field>
      <field name="annee_mort">1959</field>
    </row>
    <row>
      <field name="id">2</field>
      <field name="nom">Clavel</field>
      <field name="prenom">Bernard</field>
      <field name="annee_naissance">1923</field>
      <field name="annee_mort">2010</field>
    </row>
  </table_data>
</database>
</mysqldump>
```

Figure 44 – Utilisation de la commande mysqldump

5.3.3 API de partage / export

Pour accéder et manipuler les contenus d'une base de données (SQL ou NoSQL), il n'est pas obligatoire de s'y connecter. Certaines applications en ligne autorisent, outre l'usage d'une interface homme-machine (IHM), la possibilité de sélectionner et d'exporter des contenus dans divers formats dont le XML. C'est ce que l'on appelle une interface de programmation applicative ou API. Une API va être requêtée de manière automatisée par un programme ou un script – souvent avec des identifiants de connexion comme dans le cas des services des GAFAM – pour exploiter des données en temps réel, au fur et à mesure de leur intégration.

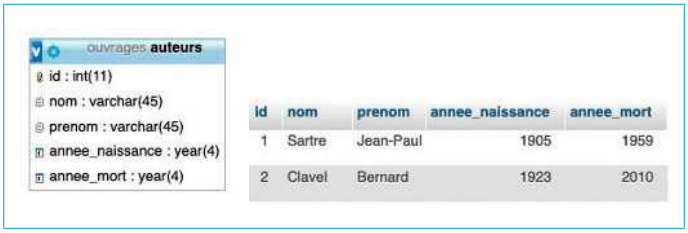


Figure 45 – Capture de la vue « conception » de phpMyAdmin

Ce genre de services aux données fortement structurées (XML) est particulièrement utilisé dans le cadre des serveurs de données du Web sémantique comme les SparQL endpoints avec le format spécifique XML RDF.

6. Manipulation de collections XML

Depuis la création de XML, le volume de documents a augmenté de manière significative.

De fait, il est nécessaire de stocker, de gérer et de manipuler des collections de documents XML dans une base de données dédiée avec un langage d'interrogation spécifique à ce format. Nous allons voir dans cette partie les solutions existantes, mais également le langage spécifique nommé XPath.

6.1 Bases de données XML

Les bases de données XML ont pour but de stocker et manipuler des collections de documents XML volumineuses. Plusieurs solutions logicielles existent sur le marché telles que BaseX (http://www.basex.org/), eXist (http://www.exist-db.org), DB2 avec IBM pureXML (https://www.ibm.com/analytics/db2), Oracle XML DB (https://www.oracle.com/fr/database/technologies/appdev/xmldb.html) ou encore Microsoft XML Data (https://www.docs.microsoft.com/en-us/sql/relational-databases/xml/xml-data-sql-server). La particularité de ces bases de données, à l'instar du format XML, est de gérer à la fois la structure des documents et les données en elles-mêmes. En effet, contrairement à une base de données relationnelle, la structure du document est bien plus complexe et requiert un stockage adapté. La notion de collections remplace celle de relations (ou tables) dans une base de données traditionnelle.

Par la suite, nous utiliserons BaseX pour effectuer les manipulations sur notre document XML. Vous pouvez pour cela télécharger l'exécutable jar (vérifiez que vous avez installé Java sur votre machine).

Pour débuter, créez un premier document en prenant notre exemple du chapitre 3 dans un fichier que vous placez dans un répertoire « basex » que vous pouvez créer dans votre répertoire « $HOME/basex/ ». Ensuite, lancez l'exécutable BaseX dans lequel vous pouvez créer une collection, ici « basex » (symbole en forme d'étoile) en précisant le dossier que nous venons de créer (figure **46**).

Nous allons nous servir par la suite du champ « Find... » pour effectuer les requêtes XPath détaillées ci-dessous.

BaseX a l'avantage de gérer des répertoires de fichiers contenant des documents XML et de les indexer pour améliorer le temps d'exécution des requêtes. Vous pouvez également l'utiliser en structure client-serveur (plutôt qu'une interface comme ici) dans le cadre d'une application. Vous pouvez vous référer à la documentation de BaseX pour son intégration.

6.2 XPath

XPath est un langage pour manipuler des collections de documents XML en définissant une navigation à travers la structure arborescente de ceux-ci. Comme nous l'avons vu dans les parties précédentes, le langage XPath est utilisé pour définir des contraintes avec XSD ou les présentations avec XSLT ou DOM. Il permet de sélectionner un contenu grâce à une syntaxe précise en se basant sur la structure du document XML.

Contrairement aux autres formats que nous avons vus jusqu'ici, XPath n'est pas exprimé en XML. Le principe de XPath est de définir le chemin à parcourir dans le document à l'instar d'une arborescence de fichier (/A/B/C). XPath est donc un langage d'interrogation dont le résultat est une séquence de nœuds présents dans le document XML ; les textes et les attributs restent des nœuds dans un arbre DOM.

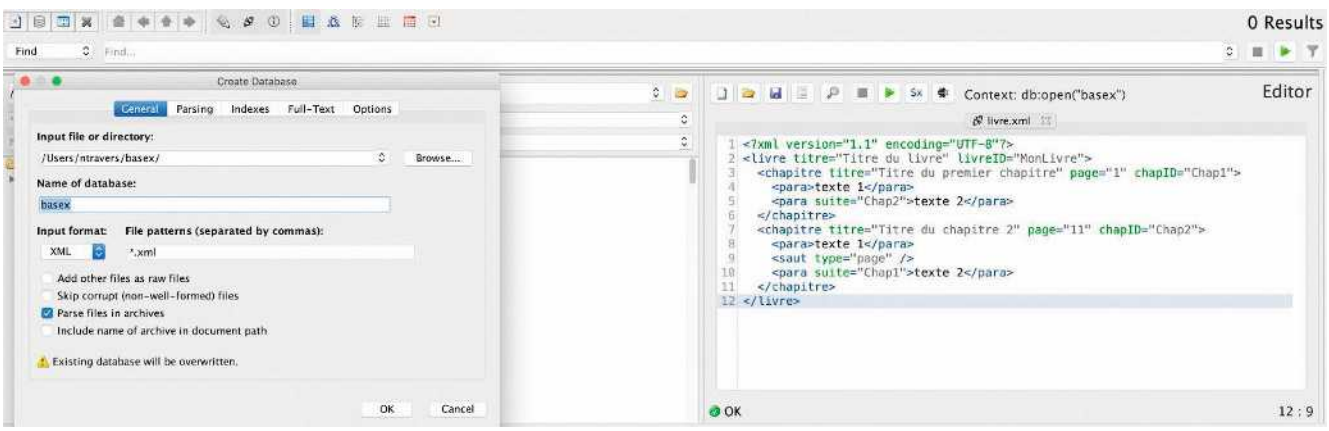


Figure 46 – Copie d'écran de BaseX

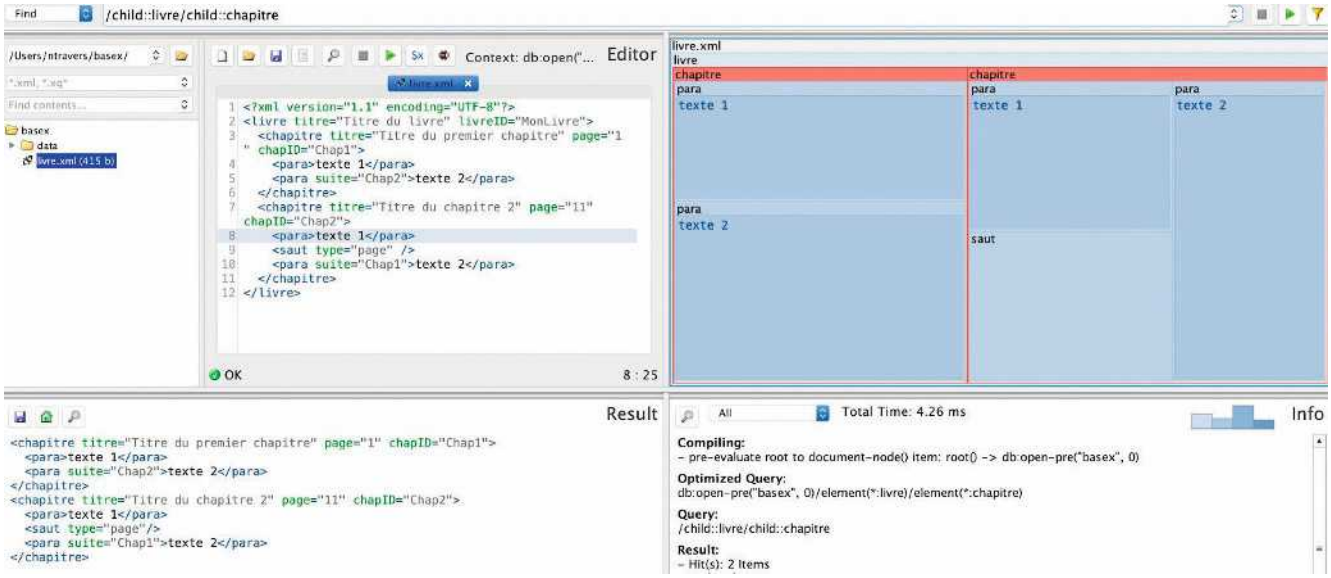


Figure 47 – Copie d'écran de requête XPath avec BaseX

Afin de se déplacer dans le document, nous exprimons chaque étape de navigation à l'aide d'un « / », associé à un axe de navigation (fils, père, frère, etc.), suivi de « :: » et du nom de l'élément concerné (appelé QName : le QName est composé du nom du nœud considéré et éventuellement de son *namespace*),

exemple :
/child::livre.

Nous choisissons ici tous les éléments fils « child:: » de la racine dont le nom de l'élément est « livre ». Étant donné que le document commence à la racine « / », le premier et seul fils possible de cette racine est bien dans notre cas, l'élément « livre ». Vous pouvez le tester sur BaseX et constater la sélection de l'ensemble du document XML, puisque nous avons sélectionné le nœud « livre ».

Si nous souhaitons maintenant choisir les « chapitres », comme fils de « livre », il suffit d'effectuer la requête : /child::livre/child::chapitre. Le résultat de la requête est cette fois-ci un ensemble de nœuds contenant 2 éléments « chapitre », comme le montre la copie d'écran de la figure **47** (éléments en rouge en haut à droite, ou séquence de documents XML « chapitre » en bas à gauche).

Il est également possible de ne pas préciser de nom de nœud avec « * » et ainsi de sélectionner tous ceux considérés par l'axe choisi.

Exemple :
/child::livre/child::chapitre/child::*

6.2.1 Axes de navigation

Il existe différents axes de navigation pour effectuer une sélection dans le document XML. Chaque axe est suivi d'un « :: » pour le séparer du nom de l'élément considéré. Le tableau **6** présente un récapitulatif des axes et des exemples de requêtes que vous pouvez tester.

Vous pouvez remarquer qu'une syntaxe simplifiée est proposée pour faciliter l'écriture des requêtes XPath pour les axes classiques (child, descendant, parent, etc.).

Il est donc important de connaître à chaque étape quels sont les nœuds ciblés (appelés nœuds contextuels) sur lesquels est appliqué un nouveau chemin. Les exemples de « following-sibling » et « preceding-sibling » sont particuliers puisqu'ils s'appliquent en première étape à l'ensemble de tous les éléments « para » (descendants de la racine), à ceux-ci s'applique l'axe « xx-sibling » qui va sélectionner l'ensemble des éléments de n'importe quel nom « * » frères. De fait, tous ceux qui ont un élément frère précédent (resp. suivants) sont les derniers éléments de leur nœud père (resp. premiers). C'est pour cela que nous ne verrons pas apparaître dans le résultat le dernier nœud (resp. premier).

6.2.2 Filtres

Une requête XPath peut intégrer des notions de filtres sur les nœuds sélectionnés comme dans une requête SQL traditionnelle. Un nœud contextuel sera sélectionné si au moins un nœud valide le prédicat du filtre.

Pour définir un filtre, l'élément contextuel est suivi de crochets « [ ] » au sein desquels seront spécifiés : une nouvelle requête XPath, un prédicat ou encore une fonction.

Exemple :
//para[position() = 2] qui sélectionne tous les éléments « para » en deuxième position par rapport à leurs frères. Vous pourrez remarquer que l'on sélectionne bien le « para » du chapitre 1, mais également celui du chapitre 2 qui est en troisième position (après le « saut »), mais en deuxième dans la liste des « para ». Vous pouvez tester la requête suivante : //chapitre/*[position() = 2], qui sélectionne les deuxièmes éléments fils de « chapitre ».

Le tableau **7** donne une liste de prédicats utilisables pour filtrer un ensemble de documents avec des exemples associés et leur simplification.

On peut combiner le tout avec des requêtes plus complexes :

//chapitre[@chapID='Chap1']/para[@suite] ou //chapitre[saut/@type='page']/para[@suite]

Tableau 6 – Récapitulatif des axes et des exemples de requêtes possibles

Axe	Fonction	Exemple	Simplification
child	Enfant direct	/child::livre/child::chapitre	/livre/chapitre
attribute	Attribut d'un élément	/livre/chapitre/attribute::titre	/livre/chapitre/@titre
descendant	Nœuds descendants quelle que soit la profondeur	/descendant::para	//chapitre
parent	Parent direct	//para/parent::*	//para/..
ancestor	Nœuds ancêtres quelle que soit la profondeur	//para/ancestor::*	
preceding-sibling	Nœuds frères précédents (du même parent)	//para/preceding-sibling::*	
following-sibling	Nœuds frères suivants (du même parent)	//para/following-sibling::*	
preceding	Tous les nœuds précédents dans l'arbre	//para/preceding::para	
following	Tous les nœuds suivants dans l'arbre	//para/following::para	
self	Nœud contextuel courant	//para/self::*	//para/.

Tableau 7 – Liste des prédicats utilisables pour filtrer un ensemble de documents

Filtre	Fonction	Exemple	Simplification
position()	Position d'un nœud dans la liste contextuelle	//para[position() = 2]	//para[2]
name()	Retourne le nom de l'élément	//chapitre/*[name()!="saut"]	
xpath	Appliqué au nœud contextuel	//*[attribute::type] //*[child::saut]	//*[@type] //*[saut]
prédicat	Filtre sur valeur	//chapitre[@page> 10]	
count()	Compte le nombre de nœuds	//chapitre[count(*) > 2]	

6.2.3 XPath 2.0

La version XPath 2.0 (https://www.w3.org/TR/xpath20/), publiée en 2010, intègre des changements de manipulation lors de l'exécution des requêtes. Comme :

– le résultat d'une requête XPath 1.0 est une séquence non ordonnée de nœuds. Pour XPath 2.0, la séquence produite est, quant à elle, ordonnée ;

– manipulation des nombres avec différents types. En effet, XPath 1.0 n'utilisait que les nombres flottants, maintenant il intègre également les types : xsd:integer, xsd:decimal et xsd:double ;

– dans la continuité du précédent, XPath 2.0 permet l'intégration des différents types de données XML : dates (et affiliés), URIs, etc. ainsi que les fonctions associées ;

– l'axe « self:: » ou « . » désigne en XPath 1.0 l'élément contextuel (nœud courant). En XPath 2.0, il peut maintenant désigner n'importe quel objet (valeur, attribut, etc.) ;

– l'axe « namespace » n'existe plus en XPath 2.0, il est remplacé par la fonction « namespace-uri » applicable à un nœud ;

– XPath 2.0 est utilisé pour XSLT 2.0. Il intègre alors des notions de *quantifiers* (some/every), ensemblistes (intersect/except) pour la manipulation de séquences de nœuds.

6.2.4 XQuery

Comme nous avons pu le voir, XPath est un langage permettant de manipuler une collection de documents pour en extraire une séquence de nœuds. Toutefois, dans le cadre de bases de données XML, il est nécessaire de faire des requêtes plus complètes pour intégrer des opérations de jointure, filtres complexes, production de documents, etc. Pour cela, le langage de requêtes XQuery (XQuery 3.0 : https://www.w3.org/TR/2014/REC-xquery-30-20140408/) a été proposé par le W3C (v3 en 2014).

Ce langage encapsule le langage XPath, et permet de définir des variables à manipuler dans l'ensemble de la requête. Il se décompose en différentes clauses appelées requêtes FLWOR (prononcé « *flower* ») pour : *For Let Order by Where Return*. Ainsi :

– *For* : affectation d'une variable à une requête XPath pour un traitement séquentiel de chaque nœud sélectionné ;

– *Let* : affectation d'une variable à une requête XPath pour produire une séquence entière ;

– *Where* : prédicats de filtrages sur les variables de la requête ;

– *Order by* : tri du résultat ;

– *Return* : production du document de sortie.

La requête ci-dessous illustre le langage avec des requêtes simples (peut être testée dans BaseX) :

```
FOR $i in db:open("basex")//chapitre
WHERE $i/saut/@type = 'page'
        AND $i/para/@suite = 'Chap1'
RETURN <chap>{string($i/@titre)}</chap>
```

Cette requête va donc prendre chaque document de la base de données « basex » et sélectionner chacun des chapitres. Pour chaque chapitre, il vérifie s'il existe un élément « saut » avec pour attribut « type » de valeur 'page' et un « para » avec attribut « suite » de valeur 'Chap1'. Si un chapitre répond à ce critère (motif arborescent), le résultat produit est un nouveau document de la sorte :

```
<chap>Titre du chapitre 2</chap>
```

XQuery est donc une version étendue de XPath permettant de faire des manipulations plus complexes avec une grande expressivité. Ce langage reste destiné à des experts dans la manipulation de documents XML, car XPath est plus accessible et répond à une grande majorité des expressions requises par une application manipulant des documents XML.

Depuis la version 3.0, d'autres clauses ont été rajoutées, en voici un extrait :

– *Group by* : groupement de nœuds par clés produisant une séquence de tuples. Il est équivalent à la combinaison de « let » et de « for » ;

– *Tumbling/Sliding* window : permet de faire des fenêtres temporelles sur des flux de données XML. Le contenu de la séquence dépend des paramètres de la fenêtre ;

– *Try/Catch* : traitement des erreurs (types, valeurs, etc.) ;

– *Switch/Case* : traitement différencié en fonction de valeurs d'une variable.

6.3 SparQL : le requêtage RDF

Le langage de requête orienté « graphe » spécifique au RDF se nomme le SPARQL Protocol and RDF Query Language (SparQL – https://www.w3.org/TR/rdf-sparql-query/). Ce langage est dédié à la sémantique et pas uniquement aux relations comme le SQL traditionnel.

Il est utilisé sur des plateformes dédiées nommées SParQL EndPoints comme Virtuoso (http://www.vos.openlinksw.com/owiki/wiki/VOS/VOSRDF) qui permettent de requêter les bases de triplets (*triple stores* RDF). Ces serveurs de requêtes trouvent leur intérêt dans la manipulation de gros volumes de données, spécifiquement dans la culture (DPpedia, Wikidata) ou la recherche avec la plateforme de recherche du CNRS en sciences humaines et sociales Isidore (EndPoint Isidore : https://www.isidore.science/sparql) ou en informatique (EndPoint DBLP : http://www.dblp.rkbexplorer.com/sparql/).

Ainsi, avec des interfaces dédiées (EndPoints), il est possible de consulter des corpus massifs de données encodés en RDF, de construire des requêtes complexes en convoquant les vocabulaires spécifiques qui les structurent. Il est aussi possible de sélectionner comme en SQL des types de données ou d'invoquer des clauses, des restrictions, de limiter les résultats ou d'en ordonner l'affichage. De plus, il est également envisageable de restructurer les informations sélectionnées au moyen d'autres vocabulaires et de transcoder le résultat en divers formats : utiliser d'autres schémas ou espaces de nommages, puis inscrire le résultat sous forme de fichier JSON, CSV, XML (voir documentation SPARQL https://www.w3.org/TR/sparql11-overview/#sparql11-results) ou encore tout simplement d'autres modèles RDF avec la fonction « construct ». Certains SparQL EndPoint proposent également des visualisations plus ou moins complexes allant du graphe au clustering pour afficher les résultats.

Une requête SPARQL comprend, dans l'ordre :
1/ Les déclarations de préfixe, pour abréger les URI (commande PREFIX).
2/ Une définition de l'ensemble de données : quels graphes RDF sont interrogés (commande FROM).
3/ Une clause de résultat, identifiant les informations à retourner de la requête (commandes SELECT, ASK, DESCRIBE, CONSTRUCT).
4/ Le modèle de requête, qui précise ce qu'il faut rechercher dans l'ensemble de données.
5/ Les options de réorganisation des résultats de la requête (commandes FILTER, ORDER BY, LIMIT...).

La requête de la figure **48** est passée au SparQL EndPoint Virtuoso de DBpedia. Elle commence par convoquer le vocabulaire des propriétés de DBpedia et de créer un préfixe 'prop'. Ensuite, elle n'utilise pas la fonction « SELECT », qui existe par ailleurs, mais « ASK » (https://www.w3.org/TR/rdf-sparql-query/#QueryForms) pour effectuer un test booléen sur un raisonnement. On indique que l'on cherche les informations dans les urls des graphes RDF <http://www.dbpedia.org/resource/Amazon_River> et <http://www.dbpedia.org/resource/Nile> les propriétés 'length' qui seront stockées respectivement dans les variables ?amazone et ?nil. Le résultat attendu est donc un booléen, ici « true ».

Le deuxième exemple de requête convoque l'ontologie DBpedia puis utilise la fonction « SELECT » pour chercher dans la ressource « Île-de-France » les villes, la population et le descriptif – en français – des villes d'Ile-de-France de moins de 100 habitants (figure **49**). Le résultat sera affiché par ordre croissant de population sous la forme d'un tableau dont la première ligne est présentée après le code (figure **49**).

7. Conclusion et ouverture

XML est donc un format de structuration et de stockage de données polyvalent qui occupe une place d'importance dans les mondes de la programmation, de la documentation et de l'édition. Il dispose de ses propres modèles de présentation et de requêtage : c'est donc un outil complet pour le stockage et l'échange de

```
PREFIX prop: <http://dbpedia.org/property/>
ASK
{
   <http://dbpedia.org/resource/Amazon_River> prop:length ?amazone .
   <http://dbpedia.org/resource/Nile> prop:length ?nil .
   FILTER(?amazone > ?nil) .
}
```

Figure 48 – Premier exemple de requête SparQL : « L'Amazone est-il plus long que le Nil ? »

```
PREFIX db-owl: <http://dbpedia.org/ontology/>
SELECT ?ville ?population ?description
WHERE {
    ?ville db-owl:region <http://fr.dbpedia.org/resource/Île-de-France> .
    ?ville db-owl:populationTotal ?population .
    ?ville  dbpedia-owl:abstract ?description .
    FILTER (?population < 100 && lang(?description) = "fr" ) .
}
ORDER BY ?population
```

Ville	population	description
http://fr.dbpedia.org/resource/Theuville_(Val-d'Oise)	« 25 »^^<http://www.w3.org/2001/XMLSchema#nonNegativeInteger>	« Theuville est une commune du Val-d'Oise située dans le Vexin français, à environ 45 km au nord-ouest de Paris. Ses habitants sont appelés les Theuvillois et les Theuvilloises. » @fr

Figure 49 – Deuxième exemple de requête SparQL : « Décrivons en français les villes d'île de France de moins de 100 habitants »

données structurées. Le XML aurait pu être « le » langage de référence pour les échanges de flux entre systèmes d'information. Les nouvelles problématiques liées au Big Data ou 5V (Valeur, Volume, Vélocité, Variété et Véracité) bousculent la donne. Étant catégorisé dans les langages du *Not only SQL (NoSQL)*, il s'intègre parfaitement dans les architectures ne nécessitant pas l'usage de bases de données relationnelles, bien qu'il puisse aussi en assurer certains aspects. Cependant, des formats plus légers comme le CSV et le JSON, plus simples, plus souples et moins volumineux lui sont souvent préférés pour la gestion des bases de données NoSQL.

Dans les milieux de la documentation, de l'édition et de l'archivistique, la rigidité structurelle du XML fait sa force : les industries de pointe continuent à dépendre de formats robustes à forte granularité, y compris en lien avec des référentiels d'autorité. Mais pour le tout-venant de l'industrie des données, les formats plus légers et plus souples tendent à détrôner le XML. L'ingénieur soucieux de choisir un format pour l'échange de données ou la création de flux devra se poser la question du choix du format, des contraintes de granularités de contenus, des principes d'autorités et de schémas liés, de pérennité du format, mais aussi du volume des données. Donc, dans des situations de modélisation de données complexes en recherche et développement par exemple (médecine, documentation d'entreprise, techniques de pointe...), le XML et sa galaxie ont encore de beaux jours devant eux. En revanche, dans les clusters d'objets connectés dont les données sont remontées en temps réel, dans la téléphonie mobile, les systèmes de recommandation commerciaux et autres « Big Data », c'est vers les données faiblement structurées que le choix s'orienterait. Dans l'écosystème du Web, même s'il existe encore des poches de résistance en faveur du xHTML sémantique, hautement qualifié, il est probable que le HTML5 et son usage préconisé par les industries des moteurs de recherche vont le rendre obsolète en matière d'usages.

XML et son écosystème

par **Gérald KEMBELLEC**

Chargé de recherche au Centre Historique Allemand, Département des Humanités Numériques
Maître de conférences au CNAM en détachement
Laboratoire Dicen-IdF (EA 7339) / Thématique(s) de recherche : Data, médiation, valorisation
Paris, France

et **Nicolas TRAVERS**

Enseignant-Chercheur au Léonard de Vinci Research Center, Pôle Universitaire
Maître de conférences au CNAM en détachement
Laboratoire DVRC / Pôle Universitaire, Paris, France
Chercheur associé au laboratoire CEDRIC / CNAM, Paris, France

Sources bibliographiques

[1] OTLET (P.). – *Traité de documentation : le livre sur le livre, théorie et pratique*, Bruxelles, Editions Mundaneum, p. 238, point 243.54.e) (1934).

[2] BUSH (V.). – *« As we may think », The atlantic monthly*, 176-1, p. 101-108 (1945).

[3] BERNERS-LEE (T.) et FISCHETTI (M.). – *Weaving the Web: the past, present and future of the World Wide Web by its inventor*, Londres, Texere, p. 45-46 (2000).

[4] ABITEBOUL (S.), MANOLESCU (I.), RIGAUX (P.) et et al. – *Web data management.* Cambridge University Press, p. 72.-92 (2011).

Normes et standards

ISO

ISO 8879 1986 ISO	1986	ISO *Traitement de l'information – Systèmes bureautiques – Langage normalisé de balisage généralisé (SGML)*
ISO/IEC 29500-1	2016	ISO *Technologies de l'information – Description des documents et langages de traitement – Formats de fichier "Office Open XML" – Partie 1 : Principes essentiels et référence de langage de balisage*
ISO 15836-2	2019	ISO *Information et documentation – L'ensemble des éléments de métadonnées Dublin Core*

W3C

W3C	1998	(éditeurs. Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler et François Yergeau) *Extensible Markup Language (XML) 1.0*, https://www.w3.org/TR/1998/REC-xml-19980210.
W3C	2006	(éditeurs. Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, François Yergeau et John Cowan), *Extensible Markup Language (XML) 1.1* (Second Edition), https://www.w3.org/TR/2006/REC-xml11-20060816/
W3C	2008	(éditeurs. Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler et François Yergeau) *Extensible Markup Language (XML) 1.0 (Fifth Edition)* *XML 1.1* https://www.w3.org/TR/2008/REC-xml-20081126
W3C	2012	(éditeurs. Shudi Gao, C.M. Sperberg-McQueen, Henry S. Thompson, Noah Mendelsohn, David Beech et Murray Maloney), *XML Schema, Part 1*. https://www.w3.org/TR/xmlschema11-1
W3C	2004	(éditeurs. Paul V. Biron et Ashok Malhotra), *XML Schema, Part 2*, https://www.w3.org/TR/xmlschema-2
W3C	2014	(éditeurs. Dan Brickley et R.V. Guha, Google), *RDF Schema 1.1*, https://www.w3.org/TR/rdf-schema/#ch_domainrange
W3C	2012	(éditeur. W3C OWL Working Group), *OWL 2 Web Ontology Language Document Overview (Second Edition)*, https://www.w3.org/TR/owl2-overview/
W3C	2000	(éditeurs. Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen, Satish Thatte, Dave Winer, UserLand), *Simple Object Access Protocol (SOAP) 1.1*, http://www.w3.org/TR/SOAP/
W3C	2013	(éditeur. The W3C SPARQL Working Group), *SPARQL 1.1 Overview*, https://www.w3.org/TR/sparql11-overview/
W3C	2014	(éditeurs. Jonathan Robie, Don Chamberlin, Michael Dyck, John Snelson), *XQuery 3.0: An XML Query Language*, https://www.w3.org/TR/2014/REC-xquery-30-20140408/
W3C	2010	(éditeurs. Anders Berglund, Scott Boag, Don Chamberlin, Mary

		F. Fernández, Michael Kay, Jonathan Robie, Jérôme Siméon), XML Path Language (XPath) 2.0 (Second Edition), https://www.w3.org/TR/xpath20/
W3C	2018	(Steven Pemberton, Daniel Austin, Jonny Axelsson, Tantek Çelik,Doug Dominiak, Herman Elenbaas, Beth Epperson, Masayasu Ishikawa, Shin'ichi Matsui, Shane McCarron, Ann Navarro, Subramanian Peruvemba, Rob Relyea, Sebastian Schnitzenbaumer, Peter Stark), *XHTML™ 1.0 The Extensible HyperText Markup Language (Second Edition). A Reformulation of HTML 4 in XML 1.0,* https://www.w3.org/TR/xhtml1
W3C	2017	(éditeur Michael Kay), *XSL Transformations (XSLT) Version 3.0,* https://www.w3.org/TR/xslt-30/
W3C	2018	(éditeurs Amelia Bellamy-Royds, Tavmjong Bah, Chris Lilley, Dirk Schulze, Eric Willigers), *Scalable Vector Graphics (SVG) 2,* https://www.w3.org/TR/SVG2/
W3C	2008	(éditeurs Dick Bulterman, Jack Jansen, Pablo Cesar, Sjoerd Mullender, Eric Hyche, Marisa DeMeglio, Julien Quint, Hiroshi Kawamura, Daniel Weck, Xabiel García Pañeda, David Melendi, Samuel Cruz-Lara, Marcin Hanclik, Daniel F. Zucker, Thierry Michel, *Synchronized Multimedia Integration Language (SMIL 3.0),* https://www.w3.org/TR/SMIL3/
W3C	2014	(éditeurs David Carlisle, Patrick Ion, Robert Miner), *Mathematical Markup Language (MathML) Version 3.0 2nd Edition,* https://www.w3.org/TR/MathML3/
W3C	2005	(éditeurs Alistair Miles, Dan Brickley), *Vocabulaire SKOS de base (fr),* https://www.w3.org/TR/swbp-skos-core-spec/,
W3C	2012	(éditeur W3C OWL Working Group), OWL 2 Web Ontology Language Document Overview (Second Edition), https://www.w3.org/TR/owl2-overview/,

WHAT-WG

WHATG	2021	DOM Living Standard https://www.dom.spec.whatwg.org

ECMA

ECMA-376	2020	Microsoft, *Office Open XML file formats*, 5th edition, https://www.ecma-international.org/publications-and-standards/standards/ecma-376/, 2016 voir aussi *Office Open XML – What is OOXML?,* http://www.officeopenxml.com/, et ISO/IEC 29500-1:2016

IETF

F. Yergeau, Request for Comments: 3629, *UTF-8, a transformation format of ISO 10646,* 2003,

https://www.tools.ietf.org/html/rfc3629#page-B4

(éditeurs M. Nottingham, et R. Sayre) Request for Comments : 4287, *The Atom Syndication Format,* https://www.tools.ietf.org/html/rfc4287

DCMI

(éditeurs DCMI Usage Board) *Dublin Core Metadata Iinitiative Terms*, https://www.dublincore.org/specifications/dublin-core/dcmi-terms/, 2020. voir aussi ISO 15836-2:2019

Logiciels et outils

Éditeurs génériques

JEdit est un éditeur multiplateformes sous licence GNU GPL écrit en Java qui autorise l'édition de fichiers XML, permet de générer des DTD à partir d'un modèle XML.

Voir https://www.sourceforge.net/projects/jedit/files/jedit/

Emacs est un éditeur multiplateformes sous licence GNU GPL avec beaucoup d'extensions,

Voir https://www.gnu.org/software/emacs/

Notepad++ est un éditeur pour Microsoft sous licence GNU GPL écrit en C++

Voir https://www.notepad-plus-plus.org/

Bibliothèques de programmation spécialisées

Java API for XML Processing (JAXP) est l'interface de programmation Oracle permettant la création, la manipulation et le traitement de fichiers XML en DOM (ou SAX ou StAX,) et la transformation transformer avec XSL.

Voir https://www.docs.oracle.com/javase/tutorial/jaxp/

Xerces est un ensemble de bibliothèques logicielles l'Apache Software Foundation pour la lecture et le traitement XML DOM et SAX en C++, Perl et Java.

Voir http://www.xerces.apache.org/

Libxml2 est un analyseur XML en C (et langages dérivés) libre disponible sous la licence du MIT. Il contient également ensemble d'outils annexes élaborés pour le traitement XML.

Voir http://www.xmlsoft.org/

xml.dom est une bibliothèque Python pour accéder et modifier les documents XML en DOM

Voir https://www.docs.python.org/fr/3/library/xml.dom.html

Éditeurs spécialisés

Oxygen est un éditeur XML commercial de la société Syncro Soft

Voir https://www.oxygenxml.com/

XML Spy est un éditeur XML commercial de la société Altova

Voir https://www.altova.com/xmlspy-xml-editor

SKOS Play est un outil en ligne et gratuit de la société Sparna qui permet de représenter et de visualiser des thésaurus, des taxonomies ou des vocabulaires contrôlés exprimés en SKOS.

Voir https://www.skos-play.sparna.fr/play/

Protégé est un éditeur graphique d'ontologie gratuit proposé par le Center for Biomedical Informatics Research de l'université de Stanford.

Voir https://www.protege.stanford.edu/

Bases de données

BaseX est une base de données XML Open Source et multiplateformes dotée d'un moteur XQuery compatible avec les mises à jour du W3C.

Voir https://www.basex.org/

EXist est une base de données de documents XML, Open Source et multiplateformes, dotée d'un moteur XQuery qui respecte également les règles du W3C.

Voir http://www.exist-db.org/

pureXML est la technologie de stockage XML native liée au serveur de données commercial IBM DB2

Voir https://www.ibm.com/analytics/db2

Et http://www.redbooks.ibm.com/abstracts/sg247315.html

MySQL est un système de gestion de base de données relationnelles sous double licence GPL et Oracle.

Voir https://www.mysql.com/fr/

Oracle XML DB est un module de stockage et d'extraction XML fourni dans le cadre d'Oracle Database.

Voir https://www.oracle.com/fr/database/technologies/appdev/xmldb.html

XML Data est le module XML de la base de données Microsoft SQL Server

Voir https://www.docs.microsoft.com/en-us/sql/relational-databases/xml/xml-data-sql-server

Virtuoso Universal Server de la société OpenLink Software est un intergiciel et un moteur de base de données hybride sous licence GPL2 et propriétaire. Il combine les fonctionnalités d'un système traditionnel de gestion de base de données relationnelle, d'une base de données relationnelle objet (SGBDR), d'une base de données RDF XML, d'un serveur d'application web et d'un serveur de fichiers dans un seul système.

Voir http://www.vos.openlinksw.com/owiki/wiki/VOS/

Voir aussi https://www.github.com/openlink/virtuoso-opensource

Validation, parcours et transformation de fichiers XML

XMLLint, outil en ligne de commande pour la validation et l'analyse de fichiers XML

Voir http://www.xmlsoft.org/xmllint.html

Voir aussi : https://www.tutorialspoint.com/unix_commands/xmllint.htm

xsltproc est un outil en ligne de commande permettant d'appliquer des feuilles de style XSLT à des documents XML. Il fait partie de libxslt, la bibliothèque C XSLT pour GNOME

Voir http://www.xmlsoft.org/XSLT/xsltproc2.html

Générateurs et validateurs en ligne de Schémas XML

mherman, outil PHP qui génère des schémas en RELAX NG, DTD: XML 1.0 DTD ou XSD depuis du XML copié/collé.

Voir http://www.xml.mherman.org

xmlgrid, outil qui génère des XML Schema/XSD depuis un fichier XML, une URL ou du XML copié/collé

Voir http://www.xmlgrid.net/xml2xsd.html

freeformatter est un service en ligne privé gratuit qui comprend plusieurs validateurs, encodeurs et décodeurs XML. xsd-generator génère un XSD à partir de XML copié/collé ou uploadé. Le générateur autorise un parmettrage assez fin d'entrées et de sorties.

Voir https://www.freeformatter.com/xsd-generator.html

et https://www.freeformatter.com/xml-validator-xsd.html

devutilsonline, https://devutilsonline.com/xsd-xml/generate-xsd-from-xml

codebeautify est un outil gratuit en ligne qui permet de réaliser des visualisations de corpus de données à partir de formes structurées, par exemple des arbres à partir de sources XML.

Voir https://www.codebeautify.org/xmlviewer

Serveur SOAP

Axis est une solution logicielle de la fondation Apache Software qui facilite le développement JAVA de services Web SOAP.

Voir http://axis.apache.org/axis/ et http://www.axis.apache.org/axis/java/user-guide.html

TECHNIQUES DE L'INGÉNIEUR

GAGNEZ DU TEMPS ET SÉCURISEZ VOS PROJETS EN UTILISANT UNE SOURCE ACTUALISÉE ET FIABLE

Techniques de l'Ingénieur propose la plus importante collection documentaire technique et scientifique en français !

Grâce à vos droits d'accès, retrouvez l'ensemble des **articles et fiches pratiques de votre offre, leurs compléments et mises à jour,** et bénéficiez des **services inclus.**

RÉDIGÉE ET VALIDÉE PAR DES EXPERTS	MISE À JOUR PERMANENTE	100 % COMPATIBLE SUR TOUS SUPPORTS NUMÉRIQUES	SERVICES INCLUS DANS CHAQUE OFFRE

- **+ de 350 000 utilisateurs**
- **+ de 10 000 articles de référence**
- **+ de 80 offres**
- **15 domaines d'expertise**

- **Automatique - Robotique**
- **Biomédical - Pharma**
- **Construction et travaux publics**
- **Électronique - Photonique**
- **Énergies**
- **Environnement - Sécurité**
- **Génie industriel**
- **Ingénierie des transports**
- **Innovation**
- **Matériaux**
- **Mécanique**
- **Mesures - Analyses**
- **Procédés chimie - Bio - Agro**
- **Sciences fondamentales**
- **Technologies de l'information**

Pour des offres toujours plus adaptées à votre métier, découvrez les offres dédiées à votre secteur d'activité

Depuis plus de 70 ans, Techniques de l'Ingénieur est la source d'informations de référence des bureaux d'études, de la R&D et de l'innovation.

www.techniques-ingenieur.fr

CONTACT : Tél. : + 33 (0)1 53 35 20 20 - Fax : +33 (0)1 53 26 79 18 - E-mail : infos.clients@teching.com

TECHNIQUES DE L'INGÉNIEUR

LES AVANTAGES ET SERVICES compris dans les offres Techniques de l'Ingénieur

ACCÈS

Accès illimité aux articles en HTML
Enrichis et mis à jour pendant toute la durée de la souscription

Téléchargement des articles au format PDF
Pour un usage en toute liberté

Consultation sur tous les supports numériques
Des contenus optimisés pour ordinateurs, tablettes et mobiles

SERVICES ET OUTILS PRATIQUES

Questions aux experts*
Les meilleurs experts techniques et scientifiques vous répondent

Articles Découverte
La possibilité de consulter des articles en dehors de votre offre

Dictionnaire technique multilingue
45 000 termes en français, anglais, espagnol et allemand

Archives
Technologies anciennes et versions antérieures des articles

Impression à la demande
Commandez les éditions papier de vos ressources documentaires

Alertes actualisations
Recevez par email toutes les nouveautés de vos ressources documentaires

*Questions aux experts est un service réservé aux entreprises, non proposé dans les offres écoles, universités ou pour tout autre organisme de formation.

ILS NOUS FONT CONFIANCE

AgroParisTech · AIRBUS · cea · CESI Enseignement Supérieur et Formation Professionnelle · cnes Centre National d'Études Spatiales · CSTB le futur en construction · ENGIE · GE · L'ORÉAL · Leroy-Somer · École Polytechnique Université Paris-Saclay · SAFRAN Aerospace Defence Security · Sagemcom · SAINT-GOBAIN · Schneider Electric · THALES · TOTAL · Valeo · VEOLIA · Wallonie · SPW Service public de Wallonie

www.techniques-ingenieur.fr

CONTACT : Tél. : + 33 (0)1 53 35 20 20 - Fax : +33 (0)1 53 26 79 18 - E-mail : infos.clients@teching.com

TECHNIQUES DE L'INGÉNIEUR

GAGNEZ DU TEMPS ET SÉCURISEZ VOS PROJETS EN UTILISANT UNE SOURCE ACTUALISÉE ET FIABLE

Techniques de l'Ingénieur propose la plus importante collection documentaire technique et scientifique en français !

Grâce à vos droits d'accès, retrouvez l'ensemble des **articles et fiches pratiques de votre offre, leurs compléments et mises à jour,** et bénéficiez des **services inclus.**

RÉDIGÉE ET VALIDÉE PAR DES EXPERTS	MISE À JOUR PERMANENTE	100 % COMPATIBLE SUR TOUS SUPPORTS NUMÉRIQUES	SERVICES INCLUS DANS CHAQUE OFFRE

- **+ de 350 000 utilisateurs**
- **+ de 10 000 articles de référence**
- **+ de 80 offres**
- **15 domaines d'expertise**

- **Automatique - Robotique**
- **Biomédical - Pharma**
- **Construction et travaux publics**
- **Électronique - Photonique**
- **Énergies**
- **Environnement - Sécurité**
- **Génie industriel**
- **Ingénierie des transports**
- **Innovation**
- **Matériaux**
- **Mécanique**
- **Mesures - Analyses**
- **Procédés chimie - Bio - Agro**
- **Sciences fondamentales**
- **Technologies de l'information**

Pour des offres toujours plus adaptées à votre métier, découvrez les offres dédiées à votre secteur d'activité

Depuis plus de 70 ans, Techniques de l'Ingénieur est la source d'informations de référence des bureaux d'études, de la R&D et de l'innovation.

www.techniques-ingenieur.fr

CONTACT : Tél. : + 33 (0)1 53 35 20 20 - Fax : +33 (0)1 53 26 79 18 - E-mail : infos.clients@teching.com

TECHNIQUES DE L'INGÉNIEUR

LES AVANTAGES ET SERVICES compris dans les offres Techniques de l'Ingénieur

ACCÈS

Accès illimité aux articles en HTML
Enrichis et mis à jour pendant toute la durée de la souscription

Téléchargement des articles au format PDF
Pour un usage en toute liberté

Consultation sur tous les supports numériques
Des contenus optimisés pour ordinateurs, tablettes et mobiles

SERVICES ET OUTILS PRATIQUES

Questions aux experts*
Les meilleurs experts techniques et scientifiques vous répondent

Articles Découverte
La possibilité de consulter des articles en dehors de votre offre

Dictionnaire technique multilingue
45 000 termes en français, anglais, espagnol et allemand

Archives
Technologies anciennes et versions antérieures des articles

Impression à la demande
Commandez les éditions papier de vos ressources documentaires

Alertes actualisations
Recevez par email toutes les nouveautés de vos ressources documentaires

*Questions aux experts est un service réservé aux entreprises, non proposé dans les offres écoles, universités ou pour tout autre organisme de formation.

ILS NOUS FONT CONFIANCE

AgroParisTech · AIRBUS · cea · CESI Enseignement Supérieur et Formation Professionnelle · cnes Centre National d'Études Spatiales · CSTB le futur en construction · ENGIE · GE · L'ORÉAL · Leroy-Somer · École Polytechnique Université Paris-Saclay · SAFRAN Aerospace Defence Security · Sagemcom · SAINT-GOBAIN · Schneider Electric · THALES · TOTAL · Valeo · VEOLIA · Wallonie · SPW Service public de Wallonie

www.techniques-ingenieur.fr

CONTACT : Tél. : + 33 (0)1 53 35 20 20 - Fax : +33 (0)1 53 26 79 18 - E-mail : infos.clients@teching.com