Recommandations W3C et ACube

Normes de réalisation Internet d'un client Riche

Version 3.1 du 22/02/2010

Etat : Validé

SUIVI DES MODIFICATIONS

Version	Rédaction	Description	Vérification	Date
0.1	S. Péguet	Création du document pour plan		19/01/04
0.2	M. Aldana S. Péguet	Rédaction du document		01/03/04
1.0	S. Péguet	Version pour diffusion		16/03/04
2.0	A. Mazier	Corrections		06/04/04
3.0	S. Pitoiset	Ajout d'AJAX		23/10/06
3.1	G.Pasquereau	Actualisation		22/02/10

LISTE DE DIFFUSION

Organisation	Nom	Info	Commentaire	Validation
		☐	☐	☐

SOMMAIRE

TABLEAUX

FIGURES

DOCUMENTS DE REFERENCE

Version	Titre	Document
3.0	Normes de réalisation Internet d'un client léger	A3_NOR_NormesW3C Client Leger_3.0
1.0	Spécifications fonctionnelles techniques du framework JS client technique	A3_SPE_SFT FW JS Client Technique_1.0
1.1	Spécifications détaillées techniques du framework JS client ergonomique	A3_SPE_SDT FW JS Client Ergonomique_1.1
2.2	Dossier d'architecture	A3_NOR_Dossier Architecture_2.2

1 OBJET DU DOCUMENT

Ce document présente l'ensemble des normes de réalisation pour les applications Internet basées sur le client riche. Il a pour vocation d'aider l'équipe projet à déterminer rapidement et sans risque les solutions techniques pour la réalisation des interfaces graphiques sur client riche.

L'ensemble de ces normes a pour leitmotiv de suivre les recommandations du consortium W3C pour pouvoir garantir une conformité d'application des standards appliqués par les éditeurs de navigateur.

1.1 INTRODUCTION

L'introduction des principes du client riche (voir §1.2) induit la rédaction de nouvelles normes liées au standard XML en plus de celles incluses dans les normes de réalisation Internet d'un client léger. Ainsi, l'utilisation de document XML, dans les applicatifs pour une manipulation côté client, entraîne une possible mise en œuvre à la fois des langages XSD ou DTD pour la validation des documents XML mais aussi du langage DOM W3C XML (JavaScript) pour la manipulation côté client des arborescences d'information présentes dans les documents XML.

C'est pourquoi une norme de développement pour chacun de ces langages s'impose, afin de garantir à l'application :

- un comportement homogène et fiable en fonction du niveau de compatibilité des configurations clientes ciblées (OS, Navigateur),
- une bonne évolutivité,
- une bonne maintenabilité,
- une bonne exploitabilité.

Les normes présentées visent à permettre un développement industriel du site garantissant son bon fonctionnement sur l'ensemble des configurations clientes ciblées, tout en gardant une possible compatibilité avec d'autres navigateurs n'engageant cependant pas le support des projets applicatifs.

Ces normes visent aussi à définir un ensemble de bonnes pratiques de réalisation en définissant :

- le périmètre d'utilisation de chacun des langages liés au standard XML,
- les solutions à des problématiques apparus par retour d'expérience,
- un ensemble de préconisations induisant une cohérence d'ensemble préalable et nécessaire à la mise en place d'un framework ergonomique.

Le respect de ces normes doit conduire à :

- assurer le bon comportement de l'application pour chaque navigateur,
- diminuer la complexité des programmes,
- améliorer la lisibilité des programmes,
- diminuer le volume de tests de validation nécessaires aux différentes configurations clientes ciblées.

Et ainsi à atteindre les objectifs d'évolutivité, maintenabilité et d'exploitabilité de l'application dans le respect des contraintes de fiabilité, de vitesse de développement et de budget des projets.

1.2 PRINCIPES DU CLIENT RICHE

Contrairement aux principes du client léger associés à la mise en place d'une ergonomie simple, les principes du client riche permettent d'implémenter des fonctionnalités métier basées sur une ergonomie complexe. Pour situer les principes du client riche, cette partie explicite ces deux ergonomies en sachant que les technologies induites par l'utilisation des principes du client riche sont détaillées dans le dossier d'architecture (partie architecture du poste client).

1.2.1 ERGONOMIE SIMPLE

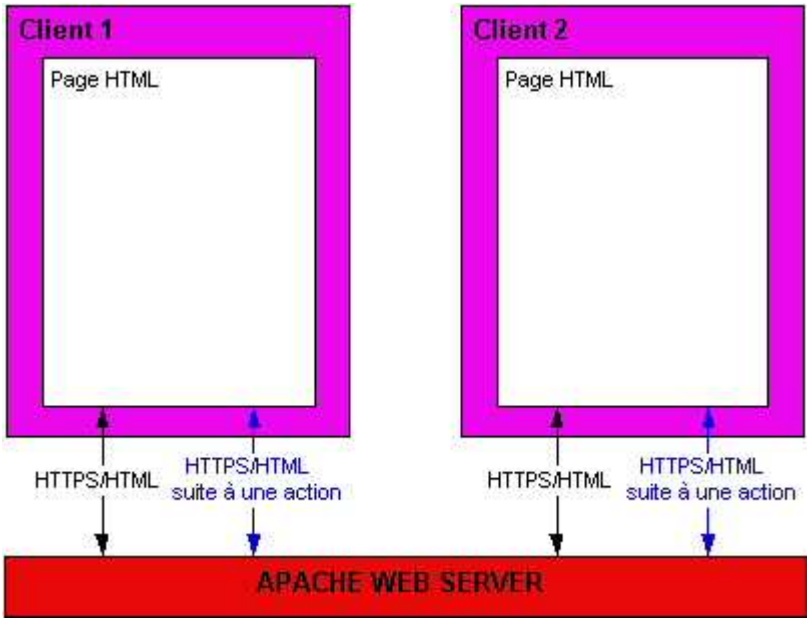

Figure 1 : ergonomie simple

Cette figure illustre le principe d'une ergonomie simple où chaque action sur une gestion métier correspond à une requête de page Web.

Ainsi, la mise à jour, la suppression, la recherche d'un élément dans une liste, la consultation d'une liste voire le tri sur cette liste correspondent chacune à une page Web dynamique générée par le serveur.

Cette ergonomie a pour principal inconvénient de ne pas prendre en compte dans la suite des actions du client la persistance de l'affichage. Par exemple, lors de l'ajout d'un élément d'une liste, il est nécessaire de rappeler l'affichage complet de la liste lorsque l'utilisateur désire revoir l'ensemble de cette liste après ajout. Même chose lors d'une mise à jour ou d'une suppression.

De plus, la recherche d'un élément ou le tri d'une liste quand les données sont déjà présentes sur le client nécessite l'appel d'un traitement serveur alors que le client est capable d'effectuer en local ce type de demande.

Il en résulte que pour ce type d'ergonomie, le serveur est sollicité à chaque action client, entraînant :

- une capacité de montée en charge plus réduite.
- un temps d'affichage plus long avec des effets « page blanche ».
- une plus grande sollicitation de la couche services pour des traitements que le client peut réaliser.
- Un nombre de données rapatriées plus important lors par exemple d'un rappel des informations lors d'une confirmation à fin de validation d'une action…
- Une limitation ergonomique du fait de la nécessité de découper l'ensemble des traitements par page.

L'avantage de ce type d'ergonomie est que le poste client n'effectue aucun traitement sur les données rapatriées et se contente d'effectuer une interprétation (X)HTML. Ceci implique que pour des pages dynamiques, aucune compétence supplémentaire (langage Javascript client) autres que celles nécessaires à la génération sur le serveur distant n'est requise.

1.2.2 ERGONOMIE COMPLEXE

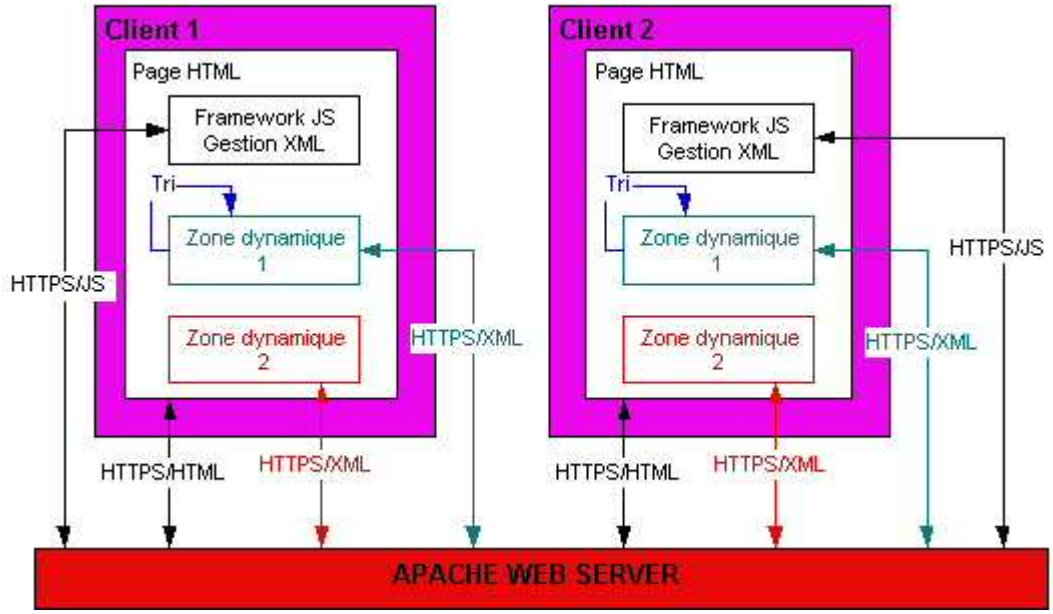

Figure 2 : ergonomie complexe

Contrairement à l'ergonomie décrite dans la partie précédente, une page Web statique est nécessaire pour gérer un métier. Ainsi, l'ensemble des résultats liés aux actions client est affiché ou désaffiché par le biais de zones dynamiques (tag <DIV>) ou par génération de contenu XHTML par le biais du DOM W3C HTML. Les données résultantes de ces actions sont appelées et montées en mémoire au format XML et converties au format (X)HTML par le client.

Ainsi, seules les données sont générées par le serveur permettant pour la gestion d'un métier de prendre en compte la persistance des données sur le client dans le cycle de vie des actions de l'utilisateur.

De plus, l'accès en mémoire sous forme d'arbre XML aux données rapatriées permet de pouvoir interagir avec celles-ci sur le client comme lors du tri d'une liste, de l'ajout d'un élément après confirmation du serveur dans l'arbre XML pour réafficher la liste des éléments…

Ce type d'ergonomie a donc pour avantages :

- une plus grande capacité de montée en charge du fait d'une sollicitation moindre des couches multi-canal et services.
- une souplesse ergonomique plus importante par l'utilisation de zones dynamiques et de génération de contenu par le client (proche des principes du client lourd).
- des flux d'échange de taille plus faible.
- utilisation optimum du cache client par l'utilisation d'objets statiques plus importante (.html et .js).

L'inconvénient principal de ce type d'ergonomie intervient dans l'augmentation de la complexité de programmation pour la prise en compte de flux XML pour générer par le client le code XHTML associé. Cette complexité est très fortement diminuée par l'usage des différentes couches du framework ergonomique.

1.3 LIMITATIONS

Ces normes sont constituées de sous ensembles liés aux possibilités des quatre différents langages XML, DTD, XSD et DOM W3C XML.

Ces sous ensembles permettent de couvrir l'ensemble des besoins liés aux principes de mise en œuvre du client riche explicités ci-dessus en sachant qu'elles nécessitent aussi le respect des normes explicitées dans les normes de réalisation Internet d'un client léger (XHTML/CSS/DOM W3C HTML).

L'utilisation d'une possibilité non décrite dans ce document d'un de ces langages est proscrite.

En l'absence de point précis dans ce document, les normes du W3C s'appliquent.

Toutefois, les technologies Internet étant en perpétuelle effervescence, ces normes peuvent s'avérer obsolètes (disparition d'un navigateur du marché) ou incomplètes (nouveau besoin ou évolution des technologies). Dans ce cas, le pôle d'architecture de DSI/PSI fera évoluer ces normes avant d'en engager les travaux de réalisation.

Enfin, ces normes s'appliquent au formalisme associé aux rapatriements de données d'une application WEB à la couche présentation d'une application WEB uniquement, et non à la couche génération de cette présentation.

Ceci signifie que les normes adressent le XML produit par une application et non pas la façon de le générer ; les DTD et XSD servant à valider le document XML rapatrié et le DOM W3C XML utile à la manipulation côté client de l'arborescence présente dans le document XML. La production de document XML côté serveur est couverte par un guide de développement côté serveur.

En cas de doute ou d'interrogation sur l'usage des technologies ou des normes dans un projet, le pôle d'architecture de DSI/PSI vous conseillera et orientera les choix à opérer au cas par cas.

1.4 CONTENU DU DOCUMENT

Le présent document est découpé en partie dont chacune décrit les possibilités de chacun des langages couverts en fonction d'un besoin donné, avec pour chacune de ces possibilités une illustration par un exemple d'implémentation.

De plus, pour faciliter la lecture de ces normes, une mise en valeur de l'information contenue est proposée pour distinguer l'information liée aux normes W3C et leurs déclinaisons pour les langages normalisés et les préconisations sur ces normes définies par le pôle d'architecture de DSI/PSI.

Ainsi, un lecteur connaissant déjà les normes W3C doit pouvoir accéder directement aux préconisations appliquées au sein des applicatifs du Ministère des Affaires Etrangères à l'aide de l'introduction de l'information de manière suivante :


Préconisation ou recommandation sur la norme du pôle d'architecture DSI/PSI

Les annexes du présent document offrent un liste de l'ensemble des directives des langages XML, DTD et XSD en sachant que le référencement associé au DOM W3C XML figure déjà dans les normes de réalisation Internet d'un client léger.

2 NORME DE DEVELOPPEMENT XML POUR CLIENT RICHE

2.1 INTRODUCTION

XML est un langage qui permet de décrire et d'échanger des documents structurés. Il permet de décrire la structure logique des documents à l'aide d'un système organisé de balises. L'objectif du W3C, initiateur de XML, est de définir un formalisme permettant d'échanger des documents complexes sur le Web, en dépassant les limites du HTML.

Les balises définies pour HTML, et la sémantique associée, sont prédéfinie par la norme. Il n'est pas possible de définir son propre système de balises, pour mieux décrire la structure de documents particuliers. Les balises définies pour HTML couvrent à la fois en mélange de genres des descriptions logique (div, center…) et sémantique (title…) des informations d'un document.

Le XML permet à une communauté d'utilisateurs de définir son propre système de balises pour marquer les constituants des documents qu'ils échangent, tout en utilisant des structures types (communauté des bibliothécaires…).

XML distingue deux classes de documents : les documents bien formés et les documents valides.

Un document est bien formé quand il obéit à la syntaxe XML stricte donnée ci-après.

Un document est valide, s'il est bien formé et s'il obéit en outre à une structure type définie explicitement dans une DTD (Document Type Definition) ou dans un Schéma XML

XML a donc été créé pour pouvoir structurer, stocker et transporter de l'information.

2.2 SYNTAXE XML

Tout document XML se compose :

- D'un prologue
- D'un arbre d'éléments
- De commentaires et d'instructions de traitement

Les documents XML utilisent une syntaxe simple et auto descriptive :

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<note>
<to>Steve</to>
<from>Miguel</from>
<heading>Rappel</heading>
<body>Réunion de travail demain</body>
</note>
```

La première ligne d'un document XML (prologue) sert à la déclaration XML. Elle définit la version du XML utilisé, l'encodage des caractères utilisés dans le document ainsi que le « standalone » précisant au parseur XML que pour l'ensemble des documents XML traités par le client, toutes les déclarations nécessaires au traitement y sont incluses ou pas (yes/no).

Dans le cadre des applicatifs du ministère des Affaires étrangères, la version du XML appliquée est le 1.0. Le jeu de caractères pratiqué est l'ISO-8859-1 (Latin-1/West European) contrairement à l'UTF-8 préconisé par le W3C du fait que les données contenues dans le document XML fournies au client sont ensuite intégrées à une page HTML. Le « standalone » est lui indiqué à « yes » lorsqu'aucun appel externe à une ressource externe (DTD, XSD, ou espace de noms non déclarés) n'est nécessaire.

La déclaration est obligatoire.

On distingue dans cet exemple la présence d'un élément racine <note>, de quatre éléments fils (to, from, heading et body) et de la dernière ligne fermant l'élément racine </note>.

2.2.1 BALISES FERMANTES

Tout élément XML doit avoir une balise fermante.

Il n'est pas possible qu'un élément XML soit non fermé sous-peine d'avoir une erreur de parsing immédiate. Cette erreur peut s'exprimer sous la forme d'une page d'erreur lors de la tentative d'ouverture dans un navigateur ou sous la forme d'une création d'objet d'erreur lors d'un accès via une API de type JDOM.

La seule exception de ce principe dans un document XML se trouve dans sa déclaration : elle ne possède pas de balise fermante car elle ne fait pas partie à proprement parler du document XML du fait que ce n'est pas un élément.

2.2.2 LA CASSE

Les balises XML sont toutes sensibles à la casse. Les caractères minuscules sont les seuls autorisés pour les élements.

Par exemple : la balise <To> est différente de la balise <to>. La balise ouvrante et fermante d'un élément XML doivent respecter exactement la même casse.

Exemples :

```
<Message>incorrecte</message>
<message>correct</message>
```

2.2.3 IMBRICATION DES ELEMENTS XML

Tous les éléments XML doivent être correctement imbriqués.

Exemples :

```
<b><i>incorrecte</b></i>
<b><i>correcte</i></b>
```

2.2.4 ELEMENT RACINE

Le premier et unique élément englobant le reste du document XML est l'élément racine.

Dans tous les documents XML, une seule paire de balise permet de définir l'élément racine, tous les autres éléments doivent être imbriqués dans cet élément.

Tous les éléments peuvent avoir des sous éléments (enfants). Les sous éléments doivent être correctement imbriqués dans les éléments parents.

Exemple :

```
<racine>
  <enfant>
    <petits_enfants>.....</petits_enfants>
  </enfant>
</racine>
```

2.2.5 LES ATTRIBUTS

Un élément peut contenir plusieurs attributs.

Exemple :

```
<prod id="33-657" media="paper"></prod>
```

« id » et « media »sont ici des attributs de la balise <prod>.

La valeur d'un attribut doit toujours être contenue dans des guillemets (simple cote ou double cote).

Les deux exemples ci-dessous sont valables tous les deux :

```
personne sexe="feminin">
<personne sexe='feminin'>
```

Si la valeur de l'attribut doit contenir des simples ou doubles cotes alors il faut observer l'un des deux formalismes ci-dessous :

```
<gangster nom='George "Shotgun" Ziegler'>
<gangster nom="George 'Shotgun' Ziegler">
```

Les valeurs des attributs doivent toujours être encadrées par des guillemets (que se soit simple cote ou double cote, les deux sont valables). Les caractères minuscules sont les seuls autorisés pour les attributs.

Les éléments XML peuvent avoir des attributs formés d'une paire nom/valeur comme en HTML en sachant que la valeur de l'attribut doit toujours être entre guillemets.

Instanciation incorrecte :

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<note date=12/11/2002>
<to>Steve</to>
<from>Miguel</from>
</note>
```

Instanciation correcte :

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<note date="12/11/2002">
<to>Steve</to>
```

```
<from>Miguel</from>
</note>
```

2.2.6 LES ESPACES

Dans la norme XML, les espaces ne sont pas tronqués à l'inverse du HTML.

Avec le HTML, une séquence comme celle-ci :

```
Bonjour              mon nom est Miguel,
```

Sera affichée ainsi:

```
Bonjour mon nom est Miguel,
```

Car le HTML supprime les espaces surnuméraires.

Les espaces sont définis par les caractères " ", "\t", "\r" et "\n" dans un document XML.

2.2.7 LES COMMENTAIRES EN XML

La syntaxe pour écrire des commentaires en XML est similaire à celle du HTML :

```
<!-- Ceci est un commentaire -->
```

Dans la mesure du possible, les documents XML seront commentés pour faciliter leurs lecture et maintenance. L'outil permettant de décommenter le code sera utilisé lors de la mise en production.

2.3 ELEMENTS

2.3.1 LES DOCUMENTS XML SONT EXTENSIBLES

Les documents XML peuvent être étendus pour contenir plus d'informations.

Reprenons l'exemple de document XML du début :

```
<note>
<to>Steve</to>
<from>Miguel</from>
<heading>Rappel</heading>
<body>Réunion de travail demain</body>
</note>
```

Imaginons que nous ayons développé une application qui utilise les informations contenues dans ce document XML.

Si nous ajoutons des éléments supplémentaires, comme ceci :

```
<note>
<date>2002-08-01</date>
<to>Steve</to>
<from>Miguel</from>
<heading>Rappel</heading>
<body>Réunion de travail demain</body>
```

```
</note>
```

Notre application basée sur les principes du client riche continue de fonctionner sans problème car les documents XML sont extensibles.

2.3.2 LES ELEMENTS XML SONT LIES

Les éléments XML sont liés comme parents et enfants. Pour comprendre la terminologie XML, il est nécessaire de savoir comment les relations entre les éléments XML sont nommées et comment le contenu de l'élément est décrit.

Voici par exemple la description d'un livre que l'on souhaite traduire sous forme d'un document XML :

Mon premier XML

Introduction au XML

- Qu'est ce que le HTML
- Qu'est ce que le XML

Syntaxe du XML

- Les éléments doivent avoir une balise fermante
- Les éléments doivent être proprement imbriqués

Voici le document XML correspondant :

```
<livre>
<titre>Mon premier XML</titre>
<prod id="33-657" media="papier"></prod>
<chapitre>Introduction au XML
<para>Qu'est ce que le HTML</para>
<para>Qu'est ce que le XML</para>
</chapitre>
<chapitre>Syntaxe du XML
<para>Les éléments doivent avoir une balise fermante</para>
<para>Les éléments doivent être proprement imbriqués</para>
</chapitre>
</livre>
```

« livre » est l'élément racine. « titre », « prod » et « chapitre » sont des éléments enfants de « livre ». « livre » est l'élément parent de « titre », « prod » et « chapitre ». « titre », « prod » et « chapitre » sont des éléments frères car ils ont le même parent.

2.3.3 LES ELEMENTS XML PEUVENT ETRE DE TYPES DIFFERENTS

Un élément XML est constitué d'une balise ouvrante, d'un contenu et d'une balise fermante.

Le contenu d'un élément XML peut être un élément, un contenu mixte, un contenu simple ou un contenu vide. Un élément peut également avoir des attributs.

Dans l'exemple précédent, « livre » a un contenu de type élément, car il contient d'autres éléments. « chapitre » a un contenu mixte car il contient à la fois du texte et d'autres éléments. « para » a un contenu simple (ou de type

texte) car il ne contient que du texte. « prod » a un contenu vide, car il ne contient pas d'information. Dans l'exemple précédent, seul l'élément « prod » a des attributs. L'attribut nommé « id » a la valeur « 33-657 » et l'attribut nommé « media » a la valeur « papier ».

Pour faciliter la mise en œuvre des principes du client riche, une limitation d'écriture des documents XML est appliquée :

- *Pas d'implémentation de contenu mixte.*
- *Pas d'utilisation d'attribut.*

Les seuls contenus autorisés sont les :

- *Contenu de type élément.*
- *Contenu de type texte.*
- *Contenu vide.*

Ces limitations permettent de faciliter la mise en oeuvre de la validation du document XML et l'interrogation de l'arborescence XML via DOM XML.

Ainsi le document XML en appliquant ces préconisations devient :

```
<livre>
<titre>Mon premier XML</titre>
<prod>
<id>33-657</id>
<media>papier</media>
</prod>
<chapitre>
<titre>Introduction au XML</titre>
<para>Qu'est ce que le HTML</para>
<para>Qu'est ce que le XML</para>
</chapitre>
<chapitre>
<titre>Syntaxe du XML</titre>
<para>Les éléments doivent avoir une balise fermante</para>
<para>Les éléments doivent être proprement imbriqués</para>
</chapitre>
</livre>
```

2.3.4 NOMS D'ELEMENTS

Les noms des éléments XML doivent suivre les règles suivantes :

- *Tous les noms peuvent être utilisés, il n'y a pas de mots réservés, mais il convient d'utiliser des noms descriptifs.*
- *Sans exagération, le nom des éléments peut être aussi long que souhaité.*
- *Les documents XML sont souvent extraits de bases de données. Il est alors souvent pertinent d'utiliser le nom des champs de la base pour créer le nom des éléments facilitant ainsi le « mapping » entre base de données et flux XML.*

- *Ils doivent être écrits en minuscules.*
- *Ils peuvent contenir des lettres, des numéros et d'autres caractères spéciaux.*
- *Ils ne doivent pas commencer avec un nombre ou un caractère de ponctuation.*
- *Ils ne peuvent pas contenir d'espaces.*
- *Les underscores sont préconisés pour faire des séparations.*
- *Il faut éviter d'utiliser « - » et « . », car le parser peut tenter de réaliser des soustractions ou considérer que le nom trouvé après le point est la méthode d'un objet.*
- *Il faut éviter l'utilisation du caractère « ; » car il est utilisé avec les namespaces (voir §2.3.6)..*
- *Il faut éviter l'utilisation des lettres accentuées ou non anglo-saxonnes comme « é, è, à ».*

2.3.5 ATTRIBUTS

Nous allons nous attarder sur un aspect important au sujet des attributs et des éléments. En effet, il est tout à fait concevable de stocker de l'information dans un attribut ou dans un élément.

Voici un exemple :

```
<personne sexe="feminin">
  <prenom>Anna</prenom>
  <nom>Smith</nom>
</personne>
```

Et le pendant en terme de représentation sous forme d'élément :

```
<personne>
  <sexe>feminin</sexe>
  <prenom>Anna</prenom>
  <nom>Smith</nom>
</personne>
```

Dans le premier exemple le sexe de la personne est un attribut de la balise personne. Dans le deuxième exemple le sexe est un élément.

Les attributs présentent les inconvénients suivants :

- *Un attribut ne peut contenir plusieurs informations. Dans l'exemple précédent, si nous avions représenté le nom de la personne sous forme d'attribut de la balise <personne> il n'aurait pas été possible par exemple que la personne est deux noms de famille.*
- *Les attributs ne décrivent aucunement une structure. Les éléments remplissent tout à fait cette fonction puisque se sont eux qui définissent l'arbre XML.*
- *La validation par DTD d'un document XML est beaucoup plus difficile à réaliser.*

Dans la philosophie du Framework ergonomique, l'information véhiculée via le XML (qui doit être transformée pour être présentée à l'utilisateur) doit apparaître essentiellement dans des éléments. Ainsi, les méthodes développées dans les bibliothèques JavaScript sont des méthodes de parcours d'arbre XML ainsi que des boucles sur les éléments. Aucune information contenue dans un attribut n'est retenue par ces méthodes de balayage.

Les attributs ne sont donc pas autorisés dans le cadre d'utilisation du Framework ergonomique.

2.3.6 LES NAMESPACES (ESPACES DE NOMS)

Les namespaces XML fournissent une méthode permettant d'éliminer les conflits de nom d'éléments. Les namespaces, ou domaines de noms XML ou encore espaces de noms XML, correspondent à des collections de noms uniques identifiées par une URI (Uniform Ressource Identifiers).

Une URI représente toutes les syntaxes identifiant une ressource internet. La ressource internet la plus utilisée est l'URL (Uniform Ressource Locators).

A noter que l'URI n'est utilisée que pour identifier le namespace et qu'elle n'est pas utilisée par le parser pour réaliser une quelconque opération. Son seul objectif est de donner un nom unique au namespace. Toutefois, il n'est pas rare que l'URI mentionnée pointe vers une véritable page web contenant des informations sur le namespace.

Le nom d'un namespace apparaît sous la forme d'un préfixe de namespaces et d'un nom local séparés par le caractère « : ». Le préfixe qui pointe sur une URI, sélectionne le namespace. Cette technique garantit ainsi l'unicité des identificateurs dans le document XML.

Un namespace est déclaré par l'attribut xmlns. La syntaxe est la suivante :

```
<element xmlns:prefixe=URI>
```

Le Framework ergonomique ne prend par en compte les namespaces puisque le document XML à traiter par les bibliothèques du FrameWork ne comporte pas de conflit d'élément. Cela peut toutefois arriver dans le cas où une même page charge deux documents dont l'élément racine est identique. Mais dans ce cas, ces deux documents seront parsés dans des instances différentes de composants et ne pourront donc pas entrer en conflit.

2.3.7 GESTION DES RETOURS CHARIOT (CR/LF)

Dans une application Windows, une nouvelle ligne dans un texte est généralement indiquée par une paire de CR/LF (carriage return / line feed).

Dans une application Unix, une nouvelle ligne est indiquée par le caractère LF.

Dans une application Macintosh, seulement le caractère CR est utilisé pour marquer une nouvelle ligne.

En XML, l'ensemble des caractères CR/LF sont convertis en LF. Ainsi, seul le caractère LF marque une nouvelle ligne.

Dans le cadre d'utilisation du Framework ergonomique, la gestion de ces caractères lors du parcours d'une arborescence en DOM W3C XML pose des problèmes car sur certains parsers présents sur certains navigateurs, il sont identifiés comme un nœud à part entière de l'arborescence. Ainsi, pour faciliter toute mise en œuvre des principes du client riche, l'utilisation des retours chariots dans tout fichier XML est proscrite dans l'arborescence XML seul un retour chariot en fin d'en-tête est autorisé.

Ainsi en reprenant l'exemple de document XML du début, le fichier XML devient :

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<note><to>Steve</to><from>Miguel</from><heading>Rappel</heading><body>Réunion de
travail demain</body></note>
```

Il n'est donc pas conseillé de visualiser un document XML sur un éditeur de texte rendant la lecture de d'une telle arborescence XML difficile. Il est préférable de le visualiser dans un éditeur spécialisé (Eclipse) affichant la structure d'un document XML présent dans un fichier.

2.3.8 SECTION CDATA

Toute implémentation encadrée dans une section CDATA est ignorée par les parsers XML.

Si par exemple un élément XML doit contenir des caractère tels que '<', '>', '&' comme du code de programme quelconque, cet élément peut être définit comme une section CDATA.

Une section CDATA démarre avec la balise "<![CDATA[" et se termine avec la balise "]]>".

Exemple :

```
<script>
<![CDATA[
function matchwo(a,b)
{
if (a < b && a < 0) then
   {
   return 1;
   }
else
   {
   return 0;
   }
}
]]>
</script>
```

2.4 OUTILS XML

L'editeur XML préconisé pour manipuler le XML est Oxygen XML Editor (http://www.oxygenxml.com).

2.5 VALIDATION D'UN DOCUMENT XML

2.5.1 INTRODUCTION

Les documents valides obéissent à une structure type prédéfinie. La définition de document type (Document Type Definition ou DTD) est le mécanisme traditionnel par lequel de telles structures sont définies. Le Schéma XML (XML Schema Definition ou XSD) est un autre moyen plus précis (et conforme lui-même à la syntaxe XML) que la DTD pour le faire.

Produire des documents valides présente un double intérêt :

- L'auteur évite de réinventer des structures déjà disponibles, validées par de nombreux utilisateurs (DTD ou XSD sont réutilisables)
- Une feuille de style XSL se définit toujours par référence à la structure des documents à formater

Voici un exemple de document XML bien formé faisant référence à un DTD qui par conséquent est considéré comme valide :

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<!DOCTYPE note SYSTEM "NoteInterne.dtd">
<note date="12/11/99">
        <to>skyce</to>
        <from>Ben</from>
        <heading>Rappel</heading>
        <body>Ne pas oublier la réunion aujourd'hui !</body>
</note>
```

2.5.2 VALIDATION XML PAR DTD

Un document XML est valide s'il est associé à une définition de type de document (DTD) et s'il respecte les contraintes qui y sont définies. La définition de type de document doit apparaître avant le premier élément du document. Le nom suivant le mot DOCTYPE dans la définition de type de document doit correspondre au nom de l'élément racine.

Voici un exemple de document XML validé par une DTD :

```
<!DOCTYPE exemple SYSTEM "exemple.dtd">
<exemple>Ceci est un document XML</exemple>
```

2.5.2.1 REGLES DTD SUR LES ELEMENTS XML

Pour que le document ci-dessus soit valide, il faut que la DTD exemple.dtd soit définie comme suit :

```
<!ELEMENT exemple (#PCDATA)>
```

La traduction de ce premier exemple de DTD est qu'un document ne peut comporter qu'un seul élément racine **exemple** et que ce dernier ne peut contenir que du texte.

Le document suivant est invalidé par cette DTD car l'élément racine n'est pas conforme :

```
<!DOCTYPE exemple SYSTEM "exemple.dtd">
<text>Ceci est un document XML</text>
```

Une fois que la règle est établie pour l'élément racine, il convient de définir la suite des règles entre les éléments du document, constituant ainsi la règle de structuration du document XML complet.

Un type d'élément a un contenu d'élément lorsque les éléments de ce type ne doivent comporter que des éléments fils (aucune donnée caractère), éventuellement séparés par des espaces.

Voici un exemple de DTD dans laquelle on définit que l'élément racine **XXX** doit précisément comporter un élément **AAA**, suivi d'un élément **BBB**. Les éléments **AAA** et **BBB** peuvent contenir du texte, mais aucun autre élément :

```
<!ELEMENT XXX (AAA , BBB)>
<!ELEMENT AAA (#PCDATA)>
<!ELEMENT BBB (#PCDATA)>
```

Ci-dessous nous avons deux exemples de document XML validés par cette DTD :

```
<!DOCTYPE XXX SYSTEM "exemple.dtd">
<XXX>
<AAA>Début</AAA>
<BBB>Fin</BBB>
</XXX>
```

```
<!DOCTYPE XXX SYSTEM "exemple.dtd">
<XXX><AAA/><BBB/></XXX>
```

Ci-dessous quelques exemples de document XML non valides avec cette DTD :

L'élément **BBB** est absent :

```
<!DOCTYPE XXX SYSTEM "exemple.dtd">
<XXX><AAA/>___</XXX>
```

L'élément **BBB** doit suivre l'élément **AAA** :

```
<!DOCTYPE XXX SYSTEM "exemple.dtd">
<XXX><BBB/><AAA/></XXX>
```

L'élément racine **XXX** ne peut contenir qu'un élément **BBB** :

```
<!DOCTYPE XXX SYSTEM "exemple.dtd">
<XXX><AAA/><BBB/><BBB/></XXX>
```

(voir Annexe A : Référence DTD pour plus de détails)

2.5.2.2 REGLES DTD SUR LES ATTRIBUTS DES ELEMENTS XML

Les attributs servent à associer des paires « nom/valeur » aux éléments. Les spécifications d'attributs ne peuvent figurer que dans les balises de début et les balises d'éléments vides. La déclaration commence par ATTLIST, suivent après le nom de l'élément auquel les attributs sont rattachés et, enfin, la définition des attributs individuels.

Voici un exemple de DTD (fichier exemple.dtd) illustrant ce principe :

```
<!ELEMENT attributes (#PCDATA)>
<!ATTLIST attributes
      aaa CDATA #REQUIRED
      bbb CDATA #IMPLIED>
```

Un attribut de type CDATA peut contenir n'importe quel caractère, s'il respecte les contraintes de forme. L'attribut obligatoire doit toujours être présent, tandis que l'attribut implicite est facultatif.

Voici des exemples de document XML valides par cette DTD :

L'attribut de type CDATA peut contenir n'importe quel caractère respectant les contraintes de forme :

```
<!DOCTYPE attributes SYSTEM "exemple.dtd">
<attributes aaa="#d1" bbb="*~*">
   Texte
</attributes>
```

L'ordre des attributs n'a pas d'importance :

```
<!DOCTYPE attributes SYSTEM "exemple.dtd">
<attributes bbb="$25" aaa="13%">
   Texte
</attributes>
```

L'attribut **bbb** peut être omis car il est implicite:

```
<!DOCTYPE attributes SYSTEM "exemple.dtd">
<attributes aaa="#d1" />
```

Voici maintenant un document non-valide par cette même DTD :

```
<!DOCTYPE attributes SYSTEM "exemple.dtd">
<attributes ___ bbb="X24"/>
```

(voir Annexe A : Référence DTD pour plus de détail)

2.5.2.3 LES ENTITIES

Le block ENTITY dans une DTD permet de définir une variable se référant à une chaîne de caractères. Lors du parsing du document XML, la référence, à une ENTITY dans ce document, fait apparaître la chaîne de caractères définie dans la DTD en lieu et place de la référence. La référence à une ENTITY dans un document XML se fait de la manière suivante :

```
&ENTITY;
```

Voici un exemple de XML dans laquelle deux ENTITY « auteur » et « copyright » sont définies dans une déclaration interne :

```
<?xml version="1.0" standalone="yes" ?>
<!DOCTYPE auteur [
<!ENTITY redacteur "Steve Peguet">
<!ENTITY copyright "Copyright MAE">
]>
<auteur>&redacteur;&copyright;</auteur>
```

Ces déclarations peuvent être externalisées et figurent dans ce cas dans le DTD de la manière suivante:

```
<?xml version="1.0" standalone="no" ?>
<!DOCTYPE auteur [
<!ENTITY redacteur SYSTEM "http://localhost/entities/entities.xml">
<!ENTITY copyright SYSTEM "http://localhost/entities/entities.xml">
<auteur>&redacteur; &copyright;</auteur>
```

(voir Annexe A : Référence DTD pour plus de détails)

2.5.3VALIDATION XML PAR XSD

Le but d'un schéma XML est de définir la façon valide de construire des blocs de données XML, comme une DTD, mais de manière plus souple, plus précise et respectueuse de la syntaxe XML.

Un schéma XML :

- Définit les éléments qui peuvent apparaître dans un document.
- Définit les attributs qui peuvent apparaître dans un document.
- Définit quels éléments sont les éléments fils.
- Définit l'ordre des éléments fils.
- Définit le nombre d'éléments fils.
- Définit si un élément est vide ou peut inclure du texte.

- Définit les types de données pour les éléments et les attributs.
- Définit les valeurs fixées par défaut pour les éléments et les attributs.

Les schéma XML sont les successeurs des DTD notamment car :

- Les XSD sont extensibles.
- Les XSD sont plus riches et plus pratiques.
- Les XSD sont écrits en XML.
- Les XSD supportent les types de données.
- Les XSD supportent les « namespace ».

Ainsi, les raisons pour lesquelles on peut être amené à utiliser les schémas XML sont :

- Les XSD supportent les types de données :
  - Il est plus facile de décrire les contenus autorisés des documents.
  - Il est plus facile de valider des données.
  - Il est plus facile de travailler avec des données venant de base de données.
  - Il est plus facile de déterminer des restrictions sur les données.
  - Il est plus facile de déterminer des formats de données.
  - Il est plus facile de convertir des données de types différents.
- Les XSD utilisent la syntaxe XML :
  - Pas besoin d'apprendre un nouveau langage.
  - On peut utiliser un éditeur XML pour éditer les XSD.
  - On peut utiliser un parseur XML pour visualiser un XSD.
  - On peut manipuler un XSD avec DOM ou Xpath.
  - On peut transformer un XSD avec XSL.

Les XSD uniformisent la représentation de l'information. En effet, comme XSD permet de mettre en conformité la représentation de l'information XML avec un standard universel. Ainsi une date comme 1999-03-11 peut être interpréter comme le 11 Mars 1999 ou le 3 Novembre 1999. Avec un XML validé par un schéma, on sait que cette date est le 11 Mars 1999.

- Les XSD sont extensibles :
  - Réutilisation d'un schéma dans un autre.
  - Création de type de données propres dérivées des types standards.
  - On peut faire référence à de multiples schémas à partir du même document.

A partir d'un premier exemple :

Soit le document « note.xml » suivant :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
<to>Steve</to>
<from>Miguel</from>
<heading>Rappel</heading>
<body>Réunion de travail demain</body>
</note>
```

Voici le document XSD « note.xsd » décrivant les éléments du document XML « note.xml » :

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://localhost"
```

```
xmlns="http://localhost"
elementFormDefault="qualified">
<xs:element name="note">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="to" type="xs:string"/>
        <xs:element name="from" type="xs:string"/>
        <xs:element name="heading" type="xs:string"/>
        <xs:element name="body" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
</xs:element>
</xs:schema>
```

L'élément « note » est décrit comme étant de type « complex type » car il contient d'autres éléments. Les autres éléments (« to », « from », « heading », « body ») sont décrits comme étant des types simples « simple type » car ils ne contiennent pas d'autres éléments.

Afin de faire référence à un XSD, un document XML doit comporter certaines informations.

Voici le fichier « note.xml », correctement complété :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note
xmlns="http://localhost"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="noteInterne.xsd">
<to>Steve</to>
<from>Miguel</from>
<heading>Rappel</heading>
<body>Réunion de travail demain</body>
</note>
```

Décortiquons un peu les différentes informations de l'en-tête du fichier XML :

Le code suivant :

```
xmlns="http://localhost"
```

spécifie le « namespace » par défaut. Cette déclaration indique au validateur de schéma que tous les éléments utilisés dans le document XML vient du « namespace » indiqué.

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

indique au validateur de schéma que l'attribut « schemaLocation » que nous utilisons se trouve dans le « namespace » « XMLSchema-instance ».

```
xsi:schemaLocation="noteInterne.xsd"
```

indique le chemin du schéma utilisé pour ce document XML.

2.5.3.1 L'ELEMENT SCHEMA

L'élément ***schema*** encadre une définition de schéma en se comportant comme un élément racine.

```
<?xml version="1.0"?>
```

```
<xs:schema>
...
</xs:schema>
```

L'élément ***schema*** peut contenir un certain nombre d'attributs. Dans les exemples donnés ci-dessous, nous déclarons un élément *schema* de la manière suivante :

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://localhost"
xmlns="http://localhost"
elementFormDefault="qualified">
...
...
</xs:schema>
```

Détaillons ces différentes informations :

```
xmlns:xs="http://www.w3.org/201/XMLSchema"
```

Cette déclaration indique que les éléments, les types de données utilisés dans ce schéma (« schema », « element », « complexType », « sequence », « string », « boolean », etc…) viennent du « namespace » indiqué. Elle précise aussi que tous les éléments et les types de données qui viennent de ce « namespace » doivent être préfixés avec « xs : ».

```
targetNamespace="http://localhost"
```

indique que les éléments définis par ce schéma (« note », « to », « from », « heading », « body ») viennent du « namespace » http://localhost/.

```
xmlns=" http://localhost"
```

indique que le « namespace » par défaut est " http://localhost".

```
elementFormDefault="qualified"
```

indique que chaque élément utilisé par l'instance XML de ce document doit être qualifié par un « namespace ».

2.5.3.2 LES ELEMENTS SIMPLES (SIMPLE ELEMENT)

Un élément simple est un élément XML ne contenant que du texte. Il ne peut pas contenir d'autres éléments ou attributs.

En revanche, la restriction "texte seulement" est assez trompeuse dans le sens où un texte peut être de types très différents. Cela peut être un de ceux qui sont inclus dans la définition des schémas XML (« boolean », « string », « date », etc…) ou cela peut également être un type personnalisé.

2.5.3.3 LES ELEMENTS COMPLEXES (COMPLEX ELEMENT)

Un élément complexe est un élément XML contenant d'autres éléments et/ou des attributs.

Il y a quatre types d'éléments complexes :

- Les éléments vides.
- Les éléments qui ne contiennent que d'autres éléments.
- Les éléments qui ne contiennent que du texte.
- Les éléments qui contiennent à la fois d'autres éléments et du texte.

2.5.3.4 L'ELEMENT ELEMENT

L'élément ***element*** se réfère à un type d'élément qui peut apparaître à l'intérieur de la portée de l'élément *ElementType* nommé.

```
<element type="Type Element"
       [minOccurs="{0 | 1}"]
       [maxOccurs="{1 | *}"]>
</element>
```

L'attribut *type* indique le nom d'un élément *ElementType* défini dans le schéma ou un autre schéma spécifié par un espace de noms fourni. La valeur donnée doit correspondre à l'attribut *name* de l'élément *ElementType*. Le type peut inclure un préfixe d'espace de noms.

L'attribut *minOccurs* détermine si l'élément est requis au moins une fois (1) ou ne l'est pas (0).

L'attribut *maxOccurs* détermine si l'élément doit apparaître au maximum une fois ou un nombre de fois illimité.

Les attributs *minOccurs* et *maxOccurs* ont la valeur 1 par défaut. Un élément sans attribut spécifié utilise ces valeurs par défaut et par conséquent doit apparaître une seule fois dans le modèle de contenu.

L'élément *element* peut posséder un élément parent *ElementType* ou *group*, mais aucun élément enfant.

Le déclarations d'élément *ElementType* peuvent contraindre le contenu et les attributs qui apparaissent dans les éléments de type nommé en se référant à d'autres déclarations de types d'élément ou d'attribut.

(voir Annexe B : Référence XSD pour plus de détails)

2.5.3.5 L'ELEMENT ATTRIBUTE

L'élément *attribute* permet de représenter un attribut XML dans une définition de schéma.

Cet élément possède plusieurs attributs destinés à définir l'attribut d'un élément.

(voir Annexe B : Référence XSD pour plus de détails)

2.5.3.6 L'ELEMENT ATTRIBUTEGROUP

L'élément *attributeGroup* permet de regrouper la définition de plusieurs attributs XML. L'élément *attributeGroup* possède deux attributs destinés à identifier l'élément et à se référer à un groupe d'attributs XML.

(voir Annexe B : Référence XSD pour plus de détails)

2.5.3.7 L'ELEMENT COMPLEXTYPE

L'élément *complexType* définit un type de données complexe pour des éléments XML.

L'élément *complexType* possède plusieurs attributs destinés à définir les caractérstiques du type de données complexe.

(voir Annexe B : Référence XSD pour plus de détails)

2.5.3.8 L'ELEMENT SIMPLETYPE

L'élément *simpleType* définit un type de données simple pour des éléments XML. L'élément *simpleType* possède plusieurs attributs destinés à définir les caractéristiques du type de données simple.

(voir Annexe B : Référence XSD pour plus de détails)

2.5.3.9 L'ELEMENT GROUP

L'élément group permet de définir un groupe d'éléments et d'y faire référence dans un schéma XML. L'élément *group* possède plusieurs attributs destinés à définir les caractéristiques du groupe d'éléments.

(voir Annexe B : Référence XSD pour plus de détails)

2.5.3.10 L'ELEMENT ANY

L'élément *any* représente n'importe quel élément dans un schéma XML. L'attribut *any* possède plusieurs attributs destinés à définir précisément l'élément XML.

(voir Annexe B : Référence XSD pour plus de détail)

2.5.3.11 L'ELEMENT ANYATTRIBUTE

L'élément anyAttribute représente n'importe quel attribut dans un schéma XML. L'élément anyAttribute possède plusieurs attributs destinés à définir précisément l'attribut XML.

(voir Annexe B : Référence XSD pour plus de détails)

2.5.3.12 L'ELEMENT KEY

L'élément *key* permet de définir un élément clé dans une structure XML. L'élément *key* possède plusieurs attributs destinés à l'identifier et à préciser son nom.

(voir Annexe B : Référence XSD pour plus de détails)

2.5.4 PRECONISATIONS POUR LA VALIDATION XML PAR LE CLIENT

La validation des fichiers de configuration XML côté serveur devra s'effectuer par DTD. La validation des échanges de flux XML inter-application ou avec un partenaire devra s'effectuer par XSD. Pour éviter une surqualité, la validation des flux XML ne se fera pas côté client que ce soit par DTD ou XSD.

3 NORME DE DEVELOPPEMENT DOM W3C XML (JAVASCRIPT)

3.1 INTRODUCTION

Le JavaScript offre une bibliothèque d'APIs permettant d'accéder et de manipuler un document XML en montant son arborescence en mémoire (DOM W3C XML ou Document Object Model). Il est ainsi possible avec cette bibliothèque de :

- Créer tout document XML.
- Naviguer dans la structure d'un document XML.
- Ajouter, modifier, supprimer et interroger des éléments du document.

Bien que JavaScript offre de telles fonctionnalités de bas niveau, il est préconisé d'utiliser les méthodes de haut niveau fournies par le Framework ergonomique avant toute méthode native JavaScript.

Si par essence le framework ergonomique ne peut pas couvrir l'ensemble des besoins, l'utilisation des APIs JavaScript est subordonnée à la présente norme et aux normes JavaScript (définies dans le document Normes de développement client léger). Le pôle d'architecture de DSI/PSI devra être tenu informé de l'usage qui en est fait.

3.2 PRINCIPES D'UNE ARBORESCENCE DOM

Nous avons donc bien à l'esprit qu'un parser peut être utilisé pour charger un document XML en mémoire. Une fois ce document présent dans la mémoire, l'information qu'il contient peut être lue et manipulée via ces APIs DOM W3C XML.

Le DOM représente une vue sous forme d'arbre du document XML. En DOM, le **documentElement** représente la racine du document XML. Cet élément peut avoir un ou plusieurs **childNodes** qui représentent les différentes branches de l'arbre XML à partir de ce nœud racine.

Le **Node Interface Model** des APIs DOM W3C XML sert par la suite à accéder individuellement à un nœud de l'arbre. Par exemple, la propriété childNodes du documentElement peut être utilisée avec une boucle « for » de programmation afin d'énumérer chaque nœud individuellement.

Ces APIs comportent donc toutes les fonctions et méthodes nécessaires pour :

- Accéder à chaque nœud de l'arbre XML ainsi qu'à la valeur de son attribut.
- Insérer et effacer des nœuds.
- Convertir un document XML en un autre document XML.

Le tableau ci-dessous donne la liste des différents types de nœud rencontrés dans un document XML et utilisables par les APIs DOM W3C XML :

Type de nœud	Exemple
Document type	<**!DOCTYPE** note SYSTEM "noteInterne.dtd">

Processing instruction	<**?xml** version="1.0"?>
Element	<**note** date="12/11/2002">Rendez-vous demain avec Miguel</**note**>
Attribute	**date**="12/11/2002"
Text	**Rendez-vous demain avec Miguel**

Tableau 1 : types de noeuds traités par les APIs DOM W3C XML

3.3 LES PRINCIPAUX OBJETS ET APIS ASSOCIEES EN JAVASCRIPT

3.3.1 NODE TYPES

Les nœuds d'un arbre XML sont de différents types. Ci-dessous, sous forme de tableau, nous donnons la liste de tous les types possibles de nœuds et la valeur retournée par les propriétés **nodeType**, **nodeTypeString**, **nodeName** et **nodeValue** d'un nœud.

nodeType	nodeTypeString	nodeName	nodeValue
1	element	tagName	null
2	attribute	name	value
3	text	#text	content of node
4	cdatasection	#cdata-section	content of node
5	entityreference	entity reference name	null
6	entity	entity name	null
7	Processinginstruction	target	content of node
8	comment	#comment	comment text
9	document	#document	null
10	documenttype	doctype name	null
11	documentfragment	#document fragment	null
12	notation	notation name	null

Tableau 2 : valeurs retournées par les propriétés nodeType, nodeTypeString, nodeName et nodeValue

Ci-dessous nous retrouvons de nouveau les différents types de nœuds mais avec la valeur de la constante qui les définit :

NodeType	Nom de la constante
1	ELEMENT_NODE
2	ATTRIBUTE_NODE
3	TEXT_NODE
4	CDATA_SECTION_NODE
5	ENTITY_REFERENCE_NODE

6	ENTITY_NODE
7	PROCESSING_INSTRUCTION_NODE
8	COMMENT_NODE
9	DOCUMENT_NODE
10	DOCUMENT_TYPE_NODE
11	DOCUMENT_FRAGMENT_NODE
12	NOTATION_NODE

Tableau 3 : constantes JavaScript liées à la propriété nodeType

Dans le cadre d'utilisation du Framework ergonomique, seule la propriété « nodeType » est utile pour gérer des règles techniques suivant le type de nœud (pas d'implémentation de la propriété « nodeTypeString »). L'utilisation des constantes pour tester les valeurs renvoyées par la propriété « nodeType » est préconisée pour faciliter la lecture du code.

Les propriétés « nodeName » et « nodeValue » seront largement utilisées pour interroger le nom de l'élément et obtenir les informations contenues dans cet élément.

3.3.2 NODE OBJECT

L'objet **node** de l'API représente un nœud de l'arbre XML. Un nœud peut être de type élément ou tout autre type de nœud précédemment cité. Cet objet des APIs DOM W3C XML est doté des propriétés et méthodes suivantes :

Propriété	Description
attributes	Renvoie un objet NamedNodeMap contenant l'ensemble des attributs du nœud
childNodes	Renvoie un objet NodeList contenant l'ensemble des nœuds fils du nœud
firstChild	Renvoie le premier nœud présent dans le nœud
lastChild	Renvoie le dernier nœud présent dans le nœud
nextSibling	Renvoie le nœud voisin suivant. Deux nœuds sont voisins (ou frères) s'ils possèdent le même nœud parent.
nodeName	Renvoie la propriété nodeName dépendant du type du nœud
nodeType	Renvoie la propriété nodeType dépendant du type du nœud sous forme de nombre
nodeValue	Renvoie ou modifie la valeur du nœud en fonction du type de nœud
ownerDocument	Renvoie le nœud racine du document XML
parentNode	Renvoie le nœud parent
previousSibling	Renvoie le nœud voisin précédent. Deux nœuds sont voisins (ou frères) s'ils possèdent le même nœud parent.

Tableau 4 : propriétés de l'objet Node

Dans le cadre d'utilisation du Framework ergonomique, les propriétés « nextSibling » et « previousSibling » ne seront pas utilisées. Il est préférable de parcourir l'arborescence à l'aide d'un objet collection « NodeList », en utilisant la propriété « childNodes » du nœud parent pour limiter les risques d'erreur, et d'utiliser ainsi des boucles « for » plus performantes (que des parcours par condition « while »).

Méthode	Description
appendChild(newChild)	Ajoute le noeud newChild à la fin des noeuds fils présents dans le noeud
cloneNode(boolean)	Renvoie un nœud cloné du nœud. Si le booléen est spécifié à « true », le nœud cloné contient aussi tous les nœuds fils présents dans le nœud.
hasChildNodes()	Renvoie « true » si le nœud possède des nœuds fils.
insertBefore(newNode,refNode)	Ajoute un nouveau nœud (« newNode ») avant le nœud existant « refNode ».
removeChild(nodeName)	Supprime le nœud spécifié par son nodeName.
replaceChild(newNode,oldNode)	Remplace le nœud « oldNode » par le nœud « newNode ».

Tableau 5 : méthodes de l'objet Node

Dans le cadre d'utilisation du Framework ergonomique, les seules méthodes à utiliser pour manipuler l'arborescence d'un document XML sont « appendChild » pour l'ajout et « removeChild » pour la suppression. La méthode « hasChildNodes » sera utilisée pour tester la présence ou non d'élément attendu.

Les méthodes « insertBefore » et « replaceChild » sont proscrites du fait de différences d'implémentation ou de comportement suivant le navigateur. Des APIs basées sur les méthodes autorisées citées ci-dessus sont proposées dans le framework technique pour les remplacer à iso-périmètre (voir document des spécifications fonctionnelles techniques du framework JS client technique).

3.3.3 NODELIST OBJECT

L'objet nodeList représente un nœud de l'arbre ainsi que tout ces nœuds fils sous la forme d'un arbre. Ci-dessous les propriétés et méthodes de l'objet :

Propriété	Description
length	Renvoie le nombre de noeuds présent dans la collection nodeList (utile lors d'un parcours d'arborescence)

Tableau 6 : propriété de l'objet NodeList

Méthode	Description
item	Renvoie un nœud spécifique présent dans la collection nodeList

Tableau 7 : méthode de l'objet nodeList

Dans le cadre d'utilisation du Framework ergonomique, la méthode « item » n'est pas recommandée pour récupérer un nœud de l'objet nodeList car cet objet est en fait une collection. Il est préférable de l'interroger comme un tableau nodeList[].

3.3.4 DOCUMENT OBJECT

La propriété **documentElement** de l'objet Document (DOM) représente l'élément racine de l'arbre XML du document. Tous les autres nœuds de l'arbre sont des nœuds fils de l'objet Document. Cet objet Node est donc requis dans tous les documents XML.

Ci-dessous la liste des propriétés et méthodes de l'objet Document :

Propriété	Description
documentElement	Renvoie l'élément racine du document XML
doctype	Renvoie le DTD ou le Schéma du document XML
implementation	Renvoie l'objet d'implémentation pour le document particulier (DOM)

Tableau 8 : propriétés de l'objet Document

Méthode	Description
createAttribute(attributeName)	Crée un nœud Attribut avec le nom spécifié
createCDATASection(text)	Crée une section CDATA contenant le texte spécifié
createComment(text)	Crée un nœud commentaire contenant le texte spécifié
createDocumentFragment()	Crée un objet documentFragment vide
createElement(tagName)	Crée un élément avec un « tagName » spécifié
createEntityReference(referenceName)	Crée une référence ENTITY avec le nom spécifié
createProcessingInstruction(target,text)	Crée un noeud de processing instruction contenant la target et le texte spécifié.
createTextNode(text)	Crée un noeud de type texte contenant le texte spécifié
getElementsByTagName(tagName)	Renvoie le noeud désiré en fonction du critère demandé (nom de l'élément) et l'ensemble de ces noeuds fils sous la forme d'un objet nodeList.

Tableau 9 : méthodes de l'objet documentElement

3.3.5 ATTR OBJECT

L'objet Attr renvoie un attribut d'un objet élément sous la forme d'un nœud de type attribut.

Cet objet possède les propriétés suivantes :

Propriété	Description
name	Renvoie ou modifie le nom de l'attribut
specified	Renvoie un booléen pour indiquer si l'attribut est renseigné ou non
value	Renvoie ou modifie la valeur de l'attribut

Tableau 10 : propriétés de l'objet Attr

Dans le cadre d'utilisation du Framework ergonomique, la gestion par attributs des informations dans un document XML n'est pas autorisée. Elle est remplacée par une gestion par élément du document XML. Les APIs JavaScript correspondantes ne sont donc pas autorisées.

3.3.6 TEXT OBJECT

L'objet text représente le nœud texte inclus dans un nœud élément.

Méthode	Description
splitText(number)	Split du texte au caractère spécifié et renvoie du reste du texte

Tableau 11 : méthode de l'objet Text

Dans le cadre d'utilisation du Framework ergonomique, l'objet Text est proscrit. Il est préférable d'utiliser l'objet Node et d'interroger sa valeur à l'aide de la propriété « nodeValue ». Cette propriété renvoie le texte de l'élément et met à disposition l'ensemble des fonctions standards du JavaScript pour la manipulation d'objet « String » (dont une méthode pour réaliser le split).

3.3.7 COMMENT OBJECT

L'objet comment représente un nœud de type commentaire. Le nœud comment ne comporte pas de propriété **nodeName** mais seulement la propriété **nodeValue** pour interroger le texte du commentaire de l'objet comment.

4 NORME DE DEVELOPPEMENT AJAX

4.1 INTRODUCTION

AJAX offre une classe XMLHttpRequest permettant de faire des requêtes HTTP afin de récupérer des données au format XML qui pourront être intégrées à un document. Il n'est ainsi plus nécessaire de recharger la totalité du document.

Créé par Microsoft pour Internet Explorer, l'objet XMLHttpRequest a été adopté par les navigateurs Mozilla, Konqueror, Safari et récemment Opéra. Bien que largement implémentée dans les navigateurs récents, cette technologie n'est pas un standard du W3C , lequel propose des fonctionnalités similaires à travers la recommandation Document Object Model (DOM) Level 3 Load and Save Specification.

Bien qu'AJAX offre de telles fonctionnalités de bas niveau, il est préconisé d'utiliser les méthodes de haut niveau fournies par la classe XMLObject du Framework ergonomique avant toute méthode native AJAX.

Si par essence le framework ergonomique ne peut pas couvrir l'ensemble des besoins, l'utilisation des APIs AJAX est subordonnée à la présente norme et aux normes JavaScript (définies dans le document Normes de développement client léger). Le pôle d'architecture de DSI/PSI devra être tenu informé de l'usage qui en est fait.

4.2 PRINCIPES D'AJAX

L'objet XMLHttpRequest s'utilise dans une architecture de type client/serveur. Le navigateur avec son moteur Javascript va faire office de client. Du côté du serveur, n'importe quelle application délivrant du XML à travers HTTP fait l'affaire. La communication entre les deux peut se faire suivant deux modes : synchrone ou asynchrone. Dans le premier cas, les traitements suivant une requête ne sont exécutés que lorsque celle-ci est terminée. Dans le second cas, les traitements sont exécutés sans attendre son résultat. C'est ce dernier cas qui est intéressant pour créer des applications interactives et dynamiques. Détaillons son mode de fonctionnement :

- L'objet XMLHttpRequest est créé. Un gestionnaire de réponse lui est associé.
- Il est alors utilisé pour créer et effectuer une requête HTTP.
- Sans attendre le résultat, le reste des instructions est exécuté. Les instructions déclenchées par une réponse du serveur seront exécutées par le gestionnaire défini plus haut dès que le navigateur aura reçu une réponse.

4.3 L'OBJET XMLHTTPREQUEST

4.3.1 PROPRIETES DE L'OBJET XMLHTTPREQUEST

Propriété	Description

onreadystatechange	Gestionnaire d'événements pour les changements d'état. Il faut assigner une fonction à cette propriété pour effectuer des traitements sur les données renvoyées après la requête
readyState	statut de l'objet
responseText	Réponse sous forme de chaîne de caractères
responseXML	Réponse sous forme d'objet DOM
status	code numérique de réponse du serveur HTTP
statusText	message accompagnant le code de réponse

Tableau 12 : propriétés de l'objet XMLHttpRequest

Les code possibles pour le statut de l'objet sont :

- 0 = non initialisé ;
- 1 = ouverture. La méthode open() a été appelée avec succès ;
- 2 = envoyé. La méthode send() a été appelée avec succès ;
- 3 = en train de recevoir. Des données sont en train d'être transférées, mais le transfert n'est pas terminé ;
- 4 = terminé. Les données sont chargées.

4.3.2 METHODES DE L'OBJET XMLHTTPREQUEST

Méthode	Description
abort ()	Abandonne la requête
getAllResponseHeaders ()	Renvoie l'ensemble de l'entête de la réponse sous forme de chaîne de caractères
getResponseHeader (champEntete)	Renvoie la valeur d'un champ d'entête HTTP.
open (method,URL,asyncFlag, userName,password)	Prépare une requête en indiquant la méthode, l'URL, la drapeau de synchronisation, le nom d'utilisateur et le mot de passe
send (contenu)	Effectue la requête, éventuellement en envoyant les données
setRequestHeader (champ,valeur)	Assigne une valeur à un champ d'entête HTTP qui sera envoyé lors de la requête

Tableau 13 : méthodes de l'objet XMLHttpRequest

ANNEXE A : REFERENCE DTD

Block ELEMENT

Voici les différentes possibilités de déclaration de règles concernant le block ELEMENT :

<!ELEMENT element-name ANY> : element can contain any combination of parsable data.

<!ELEMENT element-name ANY> : element is empty

<!ELEMENT element-name (#PCDATA)> : element with only character data.

<!ELEMENT element-name (child-element-name,child-element-name,.....)> : Elements with one or more children are defined with the name of the children elements inside parentheses

<!ELEMENT element-name (child-name)> : Declaring only one occurrence of the same element.

<!ELEMENT element-name (child-name+)> : Declaring minimum one occurrence of the same element

<!ELEMENT element-name (child-name*)> : Declaring zero or more occurrences of the same element

<!ELEMENT element-name (child-name?)> : Declaring zero or one occurrences of the same element

<!ELEMENT element-name (child-name1| child-name1)> : Declaring either/or content

<!ELEMENT element-name (#PCDATA|child-name1)*> : Declaring mixed content

Block ATTLIST

La déclaration du block ATTLIST permet de définir les règles concernant les attributs d'éléments XML :

<!ATTLIST element-name attribute-name **attribute-type default-value**>

La déclaration **attribute-type** peut avoir les valeurs suivantes :

Value	Explanation
CDATA	The value is character data
(en1|en2|..)	The value must be one from an enumerated list
ID	The value is a unique id
IDREF	The value is the id of another element
IDREFS	The value is a list of other ids
NMTOKEN	The value is a valid XML name
NMTOKENS	The value is a list of valid XML names
ENTITY	The value is an entity
ENTITIES	The value is a list of entities
NOTATION	The value is a name of a notation
xml:	The value is a predefined xml value

Tableau 1: valeurs de attribute-type pour un block ATTLIST

La déclaration **default-value** peut avoir les valeurs suivantes :

Value	Explanation
value	The default value of the attribute
#REQUIRED	The attribute value must be included in the element
#IMPLIED	The attribute does not have to be included
#FIXED value	The attribute value is fixed

Tableau 2: valeurs de default-value pour un block ATTLIST

Block ENTITY

Déclarations des blocks ENTITY concernant les caractères spéciaux pré-définis :

Internal Entity Declaration

<!ENTITY entity-name "entity-value">

External Entity Declaration

<!ENTITY entity-name SYSTEM "URI/URL">

ANNEXE B : REFERENCE XSD

Eléments du langage XSD

La colonne « Inclusion stricte » représente les éléments du langage pour lesquels l'élément peut être uniquement inclus.

Nom de l'élément	Description	Inclusion stricte	Attributs	
schema	Un schéma XML commence par l'ouverture d'un élément *schema* destiné à accueillir la définition des composants d'un document XML.		***attributeFormDefault***	indique si les attributs XML doivent être qualifiés par un espace de noms.
			blockDefault	empêche, par défaut, l'utilisation de types dérivés dans des éléments attendant le type de base.
			elementFormDefault	indique si les éléments XML doivent être qualifiés par un espace de noms.
			finalDefault	empêche, par défaut, la dérivation de type par restriction, extension ou les deux.
			id	précise un identificateur unique pour le schéma.
			targetNamespace	indique un espace de noms cible pour tout élément étranger au vocabulaire de schéma XML.
			version	indique un numéro de version.
			xml:lang	indique le langage dans lequel est conçu le document.
include	L'élément *include* permet d'inclure un schéma XML d'un même espace de noms dans un autre schéma.	schema	***id***	précise un identificateur unique pour l'élément.
			schemaLocation	spécifie une adresse URI pointant vers un schéma XML.
import	L'élément *import* permet d'importer un schéma XML avec un espace de noms différent dans un autre schéma.	schema	***id***	précise un identificateur unique pour l'élément.
			names	spécifie l'espace de noms du schéma XML.

Nom de l'élément	Description	Inclusion stricte	Attributs	
	différent dans un autre schéma.		***schemaLocation***	spécifie une adresse URI pointant vers un schéma XML.
redefine	L'élément *redefine* permet d'mporter et de redéfinir les déclarations d'un schéma XML pour un même espace de noms.	schema	***id***	précise un identificateur unique pour l'élément.
			schemaLocation	spécifie une adresse URI pointant vers un schéma XML à rédéfinir.
element	L'élément *element* permet de représenter un élément XML dans une définition de schéma.	all choice schema sequence	***abstract***	provoque l'abstraction (*true*) de l'élément XML, devant être remplacé par un autre élément.
			block	spécifie une valeur de blocage du type dans des éléments attendant le type de base.
			default	précise une valeur par défaut pour l'élément.
			final	empêche la dérivation de type par restriction, extension ou les deux.
			fixed	empêche une dérivation par restriction du type de l'élément.
			form	indique si l'élément XML doit être ou non qualifié par un espace de noms.
			id	précise un identificateur unique pour l'élément.
			maxOccurs	précise le nombre d'occurrences maximum de l'élément. Par défaut, ce nombre est égal à 1.
			minOccurs	précise le nombre d'occurrences minimum de l'élément. Par défaut, ce nombre est égal à 1.
			name	indique le nom de l'élément XML.
			nillable	signifie qu'un élément peut être valide (*true*) lorsqu'il est nul, s'il est porteur d'un attribut qualifié d'espace de noms *xsd:nil*.

Nom de l'élément	Description	Inclusion stricte	Attributs	
			ref	spécifie une référence à un autre élément de schéma.
			substitutionGroup	définit un élément pour lequel l'élément peut se substituer.
			type	fournit le type de données accepté par l'élément.
group	L'élément *group* permet de définir un groupe d'éléments et d'y faire référence dans un schéma XML.	choice complexType redefine schema sequence	***id***	précise un identificateur unique pour le groupe.
			maxOccurs	précise le nombre d'occurrences maximum du groupe. Par défaut, ce nombre est égal à 1.
			minOccurs	précise le nombre d'occurrences minimum du groupe. Par défaut, ce nombre est égal à 1.
			name	indique le nom du groupe.
			ref	indique une référence à un groupe d'attributs.
any	L'élément *any* représente n'importe quel élément dans un schéma XML.	choice sequence	***id***	précise un identificateur unique pour l'élément.
			maxOccurs	précise le nombre d'occurrences maximum de l'élément. Par défaut, ce nombre est égal à 1.
			minOccurs	précise le nombre d'occurrences minimum de l'élément. Par défaut, ce nombre est égal à 1.
			namespace	spécifie un ou plusieurs espaces de noms.
			processContents	précise le type de processus de contenu.
attribute	L'élément *attribute* permet de représenter un attribut XML dans une définition de schéma.	attributeGroup complexType extension schema	***default***	Précise une valeur par défaut pour l'attribut
			fixed	empêche une dérivation par restriction du type de l'attribut.
			form	indique si l'attribut XML doit être ou non qualifié par un espace de noms.
			id	précise un identificateur unique pour l'attribut.
			name	indique le nom de l'attribut XML.
			ref	spécifie une référence à un autre élément de schéma.
			type	fournit le type de données accepté par l'attribut.

Nom de l'élément	Description	Inclusion stricte	Attributs	
			use	indique comment l'attribut doit apparaître.
attributeGroup	L'élément *attributeGroup* permet de regrouper la définition de plusieurs attributs XML.	attributeGroup complexType extension redefine schema	***id***	précise un identificateur unique pour l'élément.
			name	indique le nom du groupe.
			ref	indique une référence à un groupe d'attributs.
anyAttribute	L'élément *anyAttribute* représente n'importe quel attribut dans un schéma XML.	attributeGroup complexType extension	***id***	précise un identificateur unique pour l'élément.
			namespace	spécifie un ou plusieurs espaces de noms.
			processContents	précise le type de processus de contenu.
complexType	L'élément *complexType* définit un type de données complexe pour des éléments XML.	element redefine schema	***abstract***	provoque l'abstraction (*true*) de l'élément XML, devant être remplacé par un autre élément.
			block	spécifie une valeur de blocage du type dans des éléments attendant le type de base.
			default	précise une valeur par défaut pour l'élément.
			final	empêche la dérivation de type par restriction, extension ou les deux.
			id	précise un identificateur unique pour l'élément.
			mixed	indique un contenu mixte (*true*) ou un contenu à base d'éléments seuls (*false*) par défaut.
			name	indique le nom de l'élément XML.
all	L'élément *all* permet de spécifier, dans un type de données complexe, un à plusieurs éléments devant apparaître une fois ou pas du tout et dans un ordre quelconque.	complexType group	***id***	précise un identificateur unique pour l'élément.
			maxOccurs	précise le nombre d'occurrences maximum de l'élément. Par défaut, ce nombre est égal à 1.
			minOccurs	précise le nombre d'occurrences minimum de l'élément. Par défaut, ce nombre est égal à 1.

Nom de l'élément	Description	Inclusion stricte	Attributs	
choice	L'élément *choice* propose une structure de choix entre plusieurs éléments possible.	choive complexType group sequence	***id***	précise un identificateur unique pour l'élément.
			maxOccurs	précise le nombre d'occurrences maximum de l'élément. Par défaut, ce nombre est égal à 1.
			minOccurs	précise le nombre d'occurrences minimum de l'élément. Par défaut, ce nombre est égal à 1.
sequence	L'élément *sequence* définit, dans un type de données complexe, un à plusieurs éléments devant obligatoirement apparaître dans un ordre prédéfini.	choive complexType group sequence	***id***	précise un identificateur unique pour l'élément.
			maxOccurs	précise le nombre d'occurrences maximum de l'élément. Par défaut, ce nombre est égal à 1.
			minOccurs	précise le nombre d'occurrences minimum de l'élément. Par défaut, ce nombre est égal à 1.
complexContent	L'élément *complexContent* permet de définir un contenu complexe pour un élément XML.	complexType	***id***	précise un identificateur unique pour l'élément.
			mixed	indique un contenu mixte (*true*) ou un contenu à base d'éléments seuls (*false*) par défaut.
simpleContent	L'élément *simpleContent* permet de créer un type de données complexe à partir d'un type de données simple.	complexType	***id***	Précise un identificateur unique pour l'élément.
extension	L'élément *extension* propose d'étendre la définition d'un élément ou d'un attribut XML à un autre type de données spécifié.	complexContent simpleContent	***base***	indique un type de données de base.
			id	précise un identificateur unique pour l'élément.
simpleType	L'élément *simpleType* définit un type de données simple pour des éléments XML.	complexType	***final***	empêche la dérivation de type par restriction, extension ou les deux.
			id	précise un identificateur unique pour l'élément.
			name	indique le nom de l'élément XML.
list	L'élément *list* permet de créer de nouveaux types de listes par dérivation	simpleType	***id***	précise un identificateur unique pour l'élément.

Nom de l'élément	Description	Inclusion stricte	Attributs	
	de types de données atomiques existants.		***itemType***	spécifie le nom d'un type de données existants.
union	L'élément *union* permet à un élément ou à un attribut XML d'être une ou plusieurs instances d'un type de données formé par la réunion de plusieurs types atomiques ou listes.	simpleType	***id***	précise un identificateur unique pour l'élément.
			memberTypes	spécifie une liste de noms de types de données séparés par un espace blanc.
restriction	L'élément *restriction* permet de restreindre les données permises dans un élément ou un attribut XML.	complexContent simpleContent simpleType	***base***	indique un type de données de base.
			id	précise un identificateur unique pour l'élément.
enumeration	L'élément *enumeration* permet de contraindre la valeur d'un élément ou d'un attribut à une seule possibilité.	restriction	***id***	précise un identificateur unique pour l'élément.
			value	spécifie une valeur de type simple (simpleType).
pattern	L'élément *pattern* permet de créer un modèle pour la valeur d'une élément ou d'un attribut XML à partir d'une expression régulière.	restriction	***id***	précise un identificateur unique pour l'élément.
			value	spécifie une valeur de type simple (simpleType).
totalDigits	L'élément *totalDigits* permet de définir le nombre total de chiffres dans un élément ou un attribut XML.	restriction	***fixed***	permet de fixer la valeur de l'élément.
			id	précise un identificateur unique pour l'élément.
			value	spécifie un nombre total de chiffres.
fractionDigits	L'élément *fractionDigits* permet de définir le nombre total de chiffres dans la partie fractionnaire d'un nombre à virgule flottante.	restriction	***fixed***	permet de fixer la valeur de l'élément.
			id	précise un identificateur unique pour l'élément.
			value	spécifie une longueur en caractères ou en octets.

Nom de l'élément	Description	Inclusion stricte	Attributs	
minInclusive	L'élément *minInclusive* permet de définir une valeur minimum inclusive pour l'élément ou l'attribut XML.	restriction	***fixed***	permet de fixer la valeur de l'élément.
			id	précise un identificateur unique pour l'élément.
			value	spécifie une valeur de type simple (simpleType).
maxInclusive	L'élément *maxInclusive* permet de définir une valeur maximum inclusive pour l'élément ou l'attribut XML.	restriction	***fixed***	permet de fixer la valeur de l'élément.
			id	précise un identificateur unique pour l'élément.
			value	spécifie une valeur de type simple (simpleType).
minExclusive	L'élément *minExclusive* permet de définir une valeur maximum exclusive pour l'élément ou l'attribut XML.	restriction	***fixed***	permet de fixer la valeur de l'élément.
			id	précise un identificateur unique pour l'élément.
			value	spécifie une valeur de type simple (simpleType).
maxExclusive	L'élément *maxExclusive* permet de définir une valeur maximum pour l'élément ou l'attribut XML.	restriction	***fixed***	permet de fixer la valeur de l'élément.
			id	précise un identificateur unique pour l'élément.
			value	spécifie une valeur de type simple (simpleType).
minLength	L'élément *minLength* permet de définir une longueur minimum pour l'élément ou l'attribut XML.	restriction	***fixed***	permet de fixer la valeur de l'élément.
			id	précise un identificateur unique pour l'élément.
			value	spécifie une longueur maximum.
maxLength	L'élément *maxlength* permet de définir une longueur maximum pour l'élément ou l'attribut XML.	restriction	***fixed***	permet de fixer la valeur de l'élément.
			id	précise un identificateur unique pour l'élément.
			value	spécifie une longueur maximum.
length	L'élément *length* permet de définir une longueur pour l'élément ou l'attribut XML.	restriction	***fixed***	permet de fixer la valeur de l'élément.
			id	précise un identificateur unique pour l'élément.
			value	spécifie une longueur en caractères ou en octets.
whiteSpace	L'élément *whiteSpace* permet de définir le comportement à adopter vis-à-vis des espaces blancs dans une valeur de chaîne	restriction	***fixed***	permet de fixer la valeur de l'élément.
			id	précise un identificateur unique pour l'élément.

Nom de l'élément	Description	Inclusion stricte	Attributs	
	de caractères.		***value***	spécifie un comportement à appliquer aux espaces blancs dans une chaîne de caractères.
key	L'élément *key* permet de définir un élément clé dans une structure XML.	element	***id***	précise un identificateur unique pour l'élément.
			name	spécifie un nom pour l'élément.
keyref	L'élément *keyref* permet de créer une référence à une clé existante dans le schéma XML.	element	***id***	précise un identificateur unique pour l'élément.
			name	spécifie un nom pour l'élément.
			refer	se réfère à un élément *key* existant.
unique	L'élément *unique* permet de définir une contrainte d'unicité sur un noeud d'une arborescence XML.	element	***id***	précise un identificateur unique pour l'élément.
			name	spécifie un nom pour l'élément.
selector	L'élément *selector* permet de définir une expression XPath chargée de sélectionner un élément XML afin de lui appliquer une clé.	key keyref unique	***id***	précise un identificateur unique pour l'élément.
			xpath	spécifie un sous-ensemble d'expressions XPath.
field	L'élément *field* permet de sélectionner par l'intermédiaire d'une expression XPath , des éléments ou des attributs destinés à être utilisés comme clé.	key keyref unique	***id***	précise un identificateur unique pour l'élément.
			xpath	spécifie un sous-ensemble d'expressions XPath.

Nom de l'élément	Description	Inclusion stricte	Attributs	
annotation	L'élément *annotation* propose une structure permttant de fournir des informations applicatives ou destinées à l'utilisateur dans un schéma.	list maxExclusive maxInclusive maxLength minExclusive minInclusive minLength pattern redefine restriction schema selector sequence simpleContent simpleType totalDigits union unique whiteSpace	***id***	précise un identificateur unique pour l'élément
appinfo	L'élément *appinfo* permet de spécifier des informations destinées à être utilisée par une application.	annotation	***source***	spécifie une adresse URI pointant vers une information.
documentation	L'élément *documentation* spécifie une information à propos du schéma destinée à l'utilisateur.	annotation	***source***	spécifie une adresse URI pointant vers une information.
			xml:lang	précise le langage dans lequel est écrit la documentation.