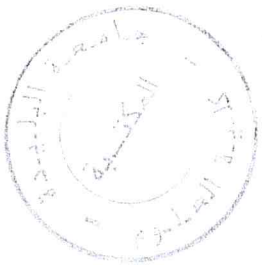
République Algérienne Démocratique et Populaire.
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique.

Université Saad Dahlab, Blida
USDB.

Faculté des sciences.
Département informatique.



Mémoire pour l'obtention
d'un diplôme d'ingénieur d'état en informatique.
Option : Intelligence Artificielle

Sujet :

Conception et réalisation d'un générateur dynamique de formulaires pour les documents XML schéma valides

Présenté par : Amokrane Abdelhalim
Guiddir Samir

Promoteur : Maredj Azze-Eddine
Encadreur : Hadj Yahia Ouahid.

Organisme d'accueil : CERIST

2005

Remerciements

Nous tenons à remercier tout particulièrement Monsieur ***Azze-Eddine Maredj*** responsable de la *Division Base de Données et Système Multimédia* du *CE.R.I.S.T* .Auteur du sujet traité, pour ces orientations , les conseils et les critiques constructives qu'il nous a octroyés tout au long de notre travail. Ainsi, pour l'aide précieuse qu'il nous a témoignée au cours de cette étude.

Nos reconnaissances vont également à Monsieur ***Ouahid Hadj Yahia*** pour son aide et ces remarques.

Que l'ensemble des membres du jury trouvent ici l'expression de notre profond respect.

A tous les enseignants ayant contribué à notre formation, notre profonde gratitude.

Nous tenons également à adresser nos sincères remerciements à nos amis de la promotion.

Que toute personne ayant contribué de près ou de loin à l'aboutissement de ce travail trouve ici notre sincère reconnaissance.

Dédicaces

À mes parents
qui m'ont tant donné
pour faire de moi
ce que je suis,
À mon frère Riad
À mes sœurs Maha et Affaf ;

A mes amis de toujours Lyes ,
Khaled , Safir ,
Moh et Tchaumi

Abdelhalim.

Dédicaces

À mes parents
qui m'ont tant donné
pour faire de moi
ce que je suis,
À la mémoire de mes grand parents,
À mes deux grand-mères paternelle
et maternelle,
À mon frère et mes deux Sœurs,
À mes oncles et tentes,
À mes nièces et neveux,
À tous mes amis(e)

Samir.

Sommaire :

Résumé :

Pour les années à venir le XML est pressenti comme le langage par excellence en matière de description et d'échange d'informations sur le Web.

Afin d'optimiser la production des documents XML et pour permettre la compatibilité des applications les traitant, il a été fortement recommandé de normaliser la structure des documents traitant d'une thématique particulière, d'où l'utilisation de DTD (Définition de type de document) ou de XML Schéma (une description formelle du contenu d'un document XML valide).

Toutefois et pour pouvoir disposer de l'information sous la forme XML, il est impératif de passer dans la majeur partie des cas par une saisie qui est une opération lourde, contraignante et coûteuse. En effet, le respect strict lors de la saisie, de la structure prédéfinie nécessite des connaissances dans le domaine, par voie de conséquence, l'appel à un personnel de saisie qualifié qu'il faut donc former. La encore le risque d'erreurs reste toujours possible ainsi qu'une saisie contraignante du fait de l'attention qu'il faut lui accorder même avec un éditeur XML doté d'un assistant de saisie. Il en résulte, pour les entreprises désirant s'approprier cette technologie, un coût de production élevé qui peut leurs représenter une contrainte insurmontable.

Afin de palier à ce problème, le présent travail est une contribution au domaine de l'édition de documents XML par la mise en ouvre d'un générateur dynamique de formulaires XML valides Schémas, désigné ci-après par XML_GENERATOR_EDIT.

C'est une solution qui n'exige pas de compétences particulières pour l'agent de saisie, à la limite quelques connaissances de base, étant donné que les contrôles et les vérifications seront assurés par le système.

Chapitre 1 :
Introduction générale

1. Préambule

Actuellement, Internet est devenu un des vecteurs principaux de diffusion de l'information et d'échange de données dans l'ensemble de la société. Il est universellement utilisé par les entreprises, les administrations et le grand public. Il représente aujourd'hui les deux tiers du volume total d'information circulant sur les réseaux.

Pourtant, si aujourd'hui tout un chacun, même le plus novice en la matière, utilise Internet dans son quotidien en tant qu'outil de travail ou de loisir, peu de gens s'en soucient quant à la manière dont sont conçues les pages qui constituent le *WEB*. Aussi, probablement, très peu de personnes reconnaissent les mérites de *Tim Berners Lee*, le fondateur du *World Wide Web* qui conçoit le premier serveur http et développa dans les années suivantes la notion d'*URL* et le code *HTML*.

Le langage HTML était parfaitement adapté à la naissance d'Internet, mais Internet est aujourd'hui au coeur d'un ensemble de nouveaux défis, e-commerce, e-gouvernement, recherche structurée d'informations, etc., qui le placent, en tant que réseau comme infrastructure de base et ses applications Web comme interfaces entre les différents acteurs sociaux. Malheuresement, *HTML* n'est plus en mesure de répondre à de telles demandes.

L'incapacité des navigateurs Internet à gérer les standards *HTML*, la difficulté de la validation des documents *HTML*, la faiblesse du système de liaison et l'absence de prise en charge internationale sont autant de facteurs dissuasifs au choix de *HTML*. *SGML* est un excellent outil; il est puissant, capable de documenter des systèmes complexes, mais il est malheureusement beaucoup trop compliqué pour répondre aux besoins actuels d'Internet.

A ce niveau, les professionnels qui suivent de près l'évolution du Web constatent l'émergence d'un sigle qui revient d'une manière quasi obsédante, c'est le XML pour *eXtensible Markup Language*.

La technologie XML s'insère progressivement au cœur des systèmes d'informations de nos entreprises. Elle facilite l'échange des documents, pérennise l'archivage des données, améliore l'accès et la recherche d'information et offre une grande souplesse lors du processus de diffusion de l'information indépendamment du média et du format de consultation.

En d'autres termes, un grand espoir est bâti sur cette technologie, capable à priori, de répondre à des besoins, de plus en plus exprimés par la société en général.

2. Objectif du travail

XML est pressenti comme le langage de description et d'échange d'information sur le Web pour les années à venir. Cependant, l'édition de documents XML s'avère une opération complexe ce qui la rend le plus souvent réservée à un personnel ayant des connaissances et des qualifications requises en la matière. Par conséquent son coût, en temps et en argent, peut devenir peu supportable pour les entreprises qui veulent adopter et adapter cette technologie, qui du reste commence à devenir incontournable.

Dans le cadre du présent projet, nous avons le privilège d'apporter notre modeste contribution pour une large utilisation de XML en proposant un outil destiné à faciliter l'édition des documents XML pour les non initiés.

Une solution à ce problème de coût et de profil requis reste la saisie des données à travers un formulaire dûment préparé en fonction de la structure du document cible.

Techniquement, il s'agit d'une solution sous forme d'un générateur dynamique de formulaires pour la saisie des documents XML reposant sur un schéma XML, et intégrant tous les contrôles nécessaires afin de générer par construction des documents valides.

En effet, un formulaire bien structuré n'exige pas une compétence donnée pour l'agent de saisie, à la limite c'est juste un personnel capable de faire de la saisie. Le risque d'erreur quant au non respect de l'ordre d'apparition des éléments, de la nature de leur contenu, et de l'oubli d'un attribut obligatoire ou de sa valeur est quasiment réduit à zéro. Les contrôles et les vérifications étant assurés par l'application.

C'est dans cette optique que nous avons mis au point un générateur de formulaire XML que l'on appellera tout au long de ce mémoire XML_GENERATOR_EDIT (XGE).

Ceci dit, la réalisation d'un tel système est loin d'être une tache facile, vu que chaque concept est régi par ses propres règles.

C'est pourquoi, une analyse minutieuse du problème s'avère nécessaire, afin de sortir avec une bonne conception permettant de solutionner le problème posé.

3. Structuration du document

Après le présent chapitre contenant une introduction générale destinée à la présentation du contexte général de notre étude, ce mémoire de fin d'étude est organisé en cinq parties, ce qui permet au lecteur de disposer d'une description claire de notre travail :

- **Partie 1 :** Etude bibliographique.
- **Partie 2 :** Conception du système.
- **Partie 3 :** Mise en œuvre du système.
- **Partie 4 :** Conclusion
- **Partie 5 :** Annexe.

3.1 Etude bibliographique

Cette partie présente les concepts et les techniques liés à la présente problématique. Elle est composée des parties suivantes :

- **Le Langage XML :** présentée en chapitre 2, elle est destinée à la présentation du langage XML. Ce chapitre parle des origines du XML, ses objectifs et ses règles. Ainsi que de la structure du document XML.
- **Les DTD :** elle est consacré à la présentation des DTD. C'est pratiquement un passage obligatoire pour une bonne compréhension du concept des schémas XML. Elle est présentée en annexe B.
- **les Schémas XML :** À ce niveau, une présentation des Schémas XML est effectuée. Elle est introduite par une comparaison entre les DTD et les Schémas, suite à quoi une étude des mécanismes régissant les schémas est détaillée. Le chapitre 3 lui est consacré.
- **Les formulaires :** Une étude des formulaires est présentée mettant en évidence leur utilisation et leur intérêt. Cette partie est détaillée dans le chapitre 4

3.2 Conception du système

La partie consacrée à la conception est présentée le long du chapitre 5. Il comporte l'objectif du travail et l'architecture du système réalisé.

L'objectif du travail s'explique par les fonctionnalités que notre système doit offrir. Après la présentation des concepts de base liés à notre système et qui sont utiles pour tracer son architecture, nous détaillerons les différents constituants de cette architecture, leurs rôles, leurs principes de fonctionnement ainsi que les différentes relations existantes entre eux.

3.3 Réalisation et mise en œuvre

La réalisation et la mise en œuvre du système sont présentées au chapitre 6. Il y est présenté l'aspect implémentation du système ainsi que le logiciel à travers des illustrations d'exemples dûment choisis.

3.4 Conclusion

A la fin, un bilan général ainsi que des perspectives éventuelles pour ce travail sont présentés sous forme d'une conclusion générale.

3.5 Annexe

Divisé en deux parties, la première est consacrée à une présentation de la plate-forme Microsoft. Net. Nous passerons en revue ses technologies de bases ainsi que ses avantages. Dans la seconde une présentation, assez détaillée, du concept des DTD est présentée.

Chapitre 2 :
Présentation du XML

1. Introduction :

XML est l'acronyme *d'eXtensible Markup Language*, Le Langage de balisage extensible. C'est le standard soutenu par *le World Wide Web Consortium* (W3C) pour le balisage de documents. Il définit une syntaxe générique utilisée pour formater des données avec des balises simple et compréhensibles par l'homme. Ce format est suffisamment souple pour être adapté à des contextes aussi variés que les sites Web, l'échange de données électroniques, les dessins vectoriels, la généalogie, les listes de biens immobiliers, la publication d'objets, les procédures d'appel à distance et les systèmes de messages vocaux.

Il a été conçu pour rendre l'utilisation de SGML (*Standard Generalized Markup Language*) sur le Web facile et directe, il simplifie le degré d'optionalité de SGML tout en permettant de développer sur le Web des types de documents crées par l'utilisateur. Avec le XML, il sera facile de définir des types de documents, de créer et de gérer des documents définis en SGML, de les transmettre et de les partager sur le Web. Il permet de supprimer deux contraintes qui limitent les développements du Web :

- dépendance envers un type de document unique et non flexible (HTML).
- complexité du SGML intégral, dont la syntaxe autorise un grand nombre d'options puissantes mais difficiles à programmer.

Ainsi, dans la mesure où il permet aux sociétés et à leurs ordinateurs de communiquer plus facilement, XML est le socle d'un ensemble de nouvelles façons de communiquer au travers d'Internet.

2. Origine et objectifs de XML :

Le développement de XML a commencé en 1996 et il a été approuvé par le W3C en février 1998. Il a été développé par un groupe de travail (GT) XML [XML *Working Group*] (initialement connu sous le nom de comité d'examen éditorial SGML [*SGML Editorial Review Board*]). **[ERWS 02]**

Le XML possède dix règles de base qui ressortent des objectifs fixés par le W3C :

- XML doit être directement utilisable sur Internet et sans difficultés.
- XML doit pouvoir supporter une large variété d'applications.
- XML doit être compatible avec SGML.
- Il doit être facile d'écrire des programmes qui puissent traiter les documents XML.
- Le nombre des caractéristiques optionnelles d'XML doit être minimum, idéalement il doit même être nul.
- Les documents XML doivent être humainement lisibles et raisonnablement clairs.
- La construction de documents XML doit pouvoir être préparée rapidement.
- Elle doit être formelle et concise.
- Les documents XML doivent être faciles à créer.

3. Différences entre le XML et le HTML :

Le HTML, le XML utilisent des balises comme conteneurs pour les éléments de données et semblent similaires en surface, mais ces balises sont en réalité sensiblement différentes, non seulement en ce qui concerne leurs définitions et leur sens, mais également quant à la manière dont elles sont crées et spécifiées. Alors que le HTML utilise des éléments plus ou moins universellement définis et acceptés, le XML permet de créer ces éléments, afin que les données puissent être structurées d'après des besoins spécifiques.

Les différences fondamentales entre le HTML et le XML sont simples : le HTML est un langage de balisage de présentation, lisible et reproductible par pratiquement n'importe quel navigateur Web moderne, tandis que le XML est un langage de balisage de contenu, sans élément intégré de présentation, uniquement composé des éléments de définition de contenu.

Le XML offre la promesse de répondre aux besoins sans cesse croissants du Web, en surmontant les limitations et les défauts du HTML. **[PTPB 03]**

L'un des importants domaines de recherche concernant le HTML est la recherche et la récupération des données. Comme tous les utilisateurs des moteurs de recherche le savent, la formulation de requêtes de recherche est une science à elle seule.
Selon son niveau de compétences, sa chance ou sa patience, l'utilisateur peut arriver à trouver soit trop peu soit trop d'informations. Ce qu'il faut c'est un moyen d'identifier et d'organiser les informations pertinentes. En HTML, on utilise des méta-éléments pour marquer les mots clés, les dates ….etc. ; à mesure que la masse des informations accessible via le Web s'accroît et que les types de données se diversifient, le travail d'indexation devient de plus en plus crucial. Face à ces enjeux, seul le XML a la capacité de pointer vers une portion d'un document et à identifier un bloc d'information de manière signifiante.

Voici maintenant certaines des différences clés entre le XML et le HTML :

- Le XML est un langage de balisage de contenu ; le HTML est un langage de balisage de présentation.
- Le XML permet de créer des éléments définis par l'utilisateur ; les éléments du HTML sont prédéfinis.
- Le XML requiert une validation ; en HTML, presque tout est acceptable.
- Le XML est orienté données ; le HTML est orienté affichage.
- Le XML permet l'échange de données entre les applications logicielles ; le HTML est destiné à la présentation visuelle des données.
- Le XML est défini et interprété de manière stricte ; le HTML est interprété de manière lâche.
- Les éléments XML doivent être fermés ; en HTML, les éléments vides n'ont pas besoin de l'être.

4. Différences entre le XML et le SGML :

Le SGML est l'un des premiers langages qui se soit attaqué au problème du transfert électronique des données sous la forme de texte balisé. Le SGML tire l'essentiel de sa puissance (et du même coup, de sa complexité) de sa fonction de métalangage, autrement dit du fait qu'il joue le rôle d'un langage dont la fonction est de décrire et de définir les langages de balisage. Le balisage contenu dans le SGML et ses dérivés est décrit dans le SGML lui-même.

De manière traditionnelle, les langages de balisage sont constitués de balises et d'éléments qui servent à décrire des données, soit pour les présenter, soit en tant que contenu. Le système ou la signification des balises ou des autres notations associées sont décrits par le métalangage qui les applique. **[PTPB 03]**

Le SGML a été utilisé de manière extensive pour la documentation technique et scientifique, les documents administratifs et diverses autres applications de ce genre, mais il est trop complexe pour être utilisé de manière standard sur le Web. Il existe toutefois des sous-ensembles et des dérivés du SGML qui se révèlent appropriés pour le balisage Web, comme le XML, le HTML et le XHTML. Tous ces langages se ressemblent fortement quant à la syntaxe et aux conventions de balisage.

Le XML partage quatre caractéristiques élémentaires essentielles avec le SGML, dont le HTML se trouve dépourvu :

- Le XML et le SGML assemblent tous deux un unique document à partir de nombreuses sources.
- Ils définissent une structure de document en utilisant une DTD ou un schéma.
- Ils marquent au moyen de balises les unités structurelles des documents.
- Ils peuvent être validés afin de s'assurer que les documents respectent les modalités de structuration définies dans la DTD ou le schéma.

Voici maintenant quelques-unes des différences clés entre le SGML et le XML

- Le SGML est très complexe ; le XML est simple.
- Les documents SGML requièrent toujours une DTD ; les documents XML peuvent être autonomes.
- Les applications logicielles SGML sont chères et complexes ; les logiciels XML sont largement répondus et disponibles, souvent en *open source.*

5. Règles et avantages de XML :

Le XML impose des règles de syntaxe très spécifiques par rapport aux autres langages de balisage:

- Il permet de définir ses propres balises et ses propres attributs. Il est donc plus flexible que HTML qui ne possède qu'un nombre limité de balise.
- Un document XML peut être validé par des règles strictes, contenues par des DTD ou des Schéma, décrivant sa structure et la hiérarchie de ses données.
- Les informations ainsi que le traitement de la mise en forme sont rigoureusement séparés de la structure du document XML.

- XML est un format standardisé ouvert ne nécessitant aucune licence, intégralement basé texte et qui peut être associé à n'importe quel jeu de caractère.
- XML est un document portable, il peut être lu sur n'importe quelle plate forme car c'est du texte et n'importe quel outil pouvant lire un fichier texte peut lire un document XML.
- De plus en plus d'application utilise le format XML ; C'est le cas de certains SGBD (Système de Gestion de Base de Données) mais aussi d'outils de bureautique comme Microsoft Office 2003 ou Sun Open Office. XML est également au cœur de la nouvelle plate-forme de développement de Microsoft.Net.
- Enfin, son interopérabilité et le fait que de grands acteurs de l'informatique dont IBM, Microsoft et Sun préconisent l'utilisation de ce puissant langage.

Voici les principaux avantages de XML:

- La lisibilité : aucune connaissance ne doit théoriquement être nécessaire pour comprendre un contenu d'un document XML.
- Une structure arborescente : permettant de modéliser la majorité des problèmes informatiques
- Universalité et portabilité : les différents jeux de caractères sont pris en compte.
- Déployable : il peut être facilement distribué par n'importe quels protocoles à même de transporter du texte, comme http.
- Intégrabilité : un document XML est utilisable par toute application pourvue d'un parseur (un logiciel permettant d'analyser un code XML).
- Extensibilité : un document XML doit pouvoir être utilisable dans tous les domaines d'applications.

Ainsi, XML est particulièrement adapté à l'échange de données et de documents. L'intérêt de disposer d'un format commun d'échange d'information dépend du contexte professionnel dans lequel les utilisateurs interviennent. C'est pourquoi, de nombreux formats de données issus de XML apparaissent (il en existe plus d'une centaine) :

- OFX : Open Financial eXchange pour les échanges d'informations dans le monde financier
- MathML : Mathematical Markup Language permet de représenter des formules mathématiques.
- CML : Chemical Markup Language permet de décrire des composés chimiques
- SMIL : Synchronized Multimedia Integration Language permet de créer des présentations multimédia en synchronisant diverses sources : audio, vidéo, texte,...

6. Concepts de base :

6.1 Document structuré

L'approche des documents structurés est basée sur deux principes:

a- Un document est un fichier texte (en format texte, exp. ASCII), auquel on superpose des conventions additionnelles qui permettent de représenter le document sous forme d'une structure hiérarchique (en arbre).

b- La structure hiérarchique du document doit correspondre le mieux et le plus explicitement possible à la nature et à la structure de l'information qui doit être véhiculée par le document.

6.2 Document XML

La norme XML permet de stocker dans un fichier des informations structurées. On parle alors de document XML. Ce dernier est alors composé de texte libre et de balises possédant éventuellement des attributs. **[KMPJ 01]**

Le XML crée des documents qui sont bien structurés et par extension tous les langages basés sur le XML sont aussi correctement structurés, ce qui signifie que les données XML sont plus faciles à utiliser.

6.3 Document XML bien formé (Well-formed document)

Un document XML est dit "bien formé" si celui-ci ne respecte que les règles de la grammaire XML (balises fermées, correctement imbriquées...) .Le balisage des éléments est librement choisi. Autrement dit, un document XML est bien formé s'il obéit à toutes les contraintes de forme données dans la spécification XML du W3C. Il doit comprendre la déclaration XML, un seul élément racine, conformité des noms des éléments et des attributs, les balises doivent être correctement imbriquées (le respect de l'imbrication stricte des éléments). D'autres conditions sont requises pour qu'un document soit bien formé. Ainsi, les valeurs d'attributs doivent être entourées de guillemets et possèdent des noms uniques. En outre, tous les éléments doivent être fermés, Ce document est alors déclaré correct par un parseur XML.

6.4 Document XML valide (valid document)

Le XML est destiné à être lu par des agents logiciels donc la tolérance aux erreurs ou aux ambiguïtés doit être bien moindre. Les données XML doivent être valides en plus d'être bien formées.

En général, les documents XML sont validés à l'aide d'un parseur XML. Ce dernier tente de lire et d'interpréter le document en suivant plusieurs étapes. Tout d'abord il le teste par rapport aux règles de bonne formation ; une fois le test satisfait, le document est ensuite comparé avec une DTD(Document Type Definition) ou un Schéma XML pour vérifier la validité.

La DTD ou le Schéma XML doivent être accessibles aux documents concernés par le processus de validation, que ce soit localement ou via un URI. Tous les éléments du document XML doivent être décrit par le DTD ou le Schéma XML référencé(e), sans que la validation échoue.

Donc, si le document est bien formé et s'il obéit aux contraintes de structuration et de format définies dans la DTD ou le Schéma XML , il est dit « valide ». Si un document ne respecte pas les contraintes décrites par sa structure alors le document est bien formé mais non valide.

6.5 Les DTD

La DTD (*Document Type Definition*, définition de type de document) définit les éléments utilisés et les attributs qu'ils requièrent et permet au document d'être échangé et compris par d'autres logiciels. Chaque DTD présente un ensemble de règles qui définissent explicitement le nom, le contenu et le contexte de chaque élément du document XML. Elle est requise pour qu'un document XML soit valide, au lieu d'être simplement bien formé.
Tout document XML peut transporter avec lui sa propre description de format ou recourir à une DTD courante disponible via un URI *(Uniform Resource Identifier)*.
Les blocs constructeurs de base qui permettent de créer une DTD sont :

- **Eléments** : les blocs principaux du XML.
- **Attributs** : information permettant de décrire plus précisément un élément.
- **Entités** : variables utilisées pour décrire des références texte courantes.
- **PCDATA** : Signifie, *Parsed Character Data* (données caractères parsées). Les PCDATA correspondent au texte contenu entre les balises de début et de fin d'un élément XML
- **CDATA** : Signifie, *Character Data* (données caractères). Le texte contenu dans une balise CDATA n'est pas parsé et ses caractères de balisage ignorés.

Les deux composants clés d'une DTD sont les éléments et les attributs.
Dans une DTD, les éléments XML sont déclarés au moyen d'une déclaration d'éléments en utilisant la syntaxe suivante :

```
< !ELEMENT   nom_element   (contenu_element)>
```

Les DTD commencent habituellement par une définition de l'élément racine, que suivent des définitions progressives, jusqu'à atteindre le niveau du texte lui-même.
Si un élément possède des attributs, ceux-ci doivent également être déclarés. Ils prennent souvent la forme

```
< !ATTLIST  nom_element  nom CDATA  #IMPLIED>
```

Nous reviendrons avec plus de détails sur les notions qui interviennent dans ce mécanisme dans la partie Annexe B.

6.6 Les SCHEMAS XML

Les Schémas XML sont une solution alternative aux DTD, d'une autre façon des DTD améliorées, car ils font appel à la syntaxe du XML et supportent les types de données et les espaces de noms. Ils sont employés pour identifier un ensemble de composants à utiliser dans les documents XML et pour fournir les règles de leurs combinaisons correctes.
Comme les DTD les Schémas ont pour rôle de définir :

- Les éléments et les attributs qui apparaissent dans le document.
- Les éléments qui sont des éléments enfants.
- L'ordre de séquence dans lequel les éléments enfants peuvent apparaître.

- Le nombre d'éléments enfants.
- Si un élément est vide ou peut inclure du texte.
- Les valeurs par défaut des attributs.

Nous reviendrons sur toutes les notions relatives aux mécanismes et la description des Schémas XML de manière plus détaillée dans le chapitre suivant (base essentielle de notre étude).

7. Terminologie et syntaxe d'un document XML

En réalité un document XML est structuré en deux parties **[ERWS 02]** :

- Le prologue.
- l'arbre des éléments XML.

7.1 Le prologue

Le prologue est la partie introduction dans un document XML. Il concerne tous ce qui se trouve avant la balise de début de l'élément racine du document XML. Il n'est pas obligatoire, mais vivement recommandé (de part la recommandation XML 1.0). Il contient des informations utiles pour le traitement des données qui y sont contenues. Il est, de plus, subdivisé en plusieurs sous parties. Il inclut trois types de balises :

- la balise de Déclaration XML.
- La balise des Instructions de traitement.
- La balise de définition (ou de référence à) d'éventuelles contraintes régissant le document, déclaration de type de document ou Schéma XML

7.1.1 La déclaration XML

Elle permet d'indiquer la version de la norme XML utilisée pour créer le document, le jeu de caractères (en anglais encoding) utilisé et l'autonomie du document. **[SILA 00]**
Cette balise a la forme suivante :

```
<?xml version='numéro' encoding='encodage' standalone='yes|no'?>
```

Les constituants de la balise sont :

<	?xml	version='numéro'	encoding='encodage'	standalone='yes|no'	>
Ouverture de la balise	La déclaration du document	Le numéro de version d'XML	Le codage des caractères	spécifie le type de la DTD	Fermeture de la Balise

L'attribut *version* spécifie la version de XML, requise pour traiter le document, telle que '1. 0'. Cet attribut ne peut être omis. Il indique au navigateur que ce qui suit est un document XML selon sa version'1. 0'.

L'attribut *encoding* indique le type de codage ou bien le jeu de caractères d'encodage utilisé dans le document XML. Le jeu de caractères "ISO-8859-1" par exemple, a l'avantage d'accepter la plupart des lettres avec des accents. Mais il existe d'autres jeux de caractères comme UTF-8 ou UTF-16 plutôt destinés aux anglo-saxons car ils ne reprennent pas les accents. Le tableau suivant nous donne quelques types de codages :

Norme	Correspondance
UTF-8	Jeu de caractère universel sur 8 bits(Unicode compressé).
UTF-16	Jeu de caractère universel sur 16 bits.
ISO-8859-1	Latin1-langues d'Europe de l'ouest et d'Amérique latine.
ISO-8859-2	Latin2-langues d'Europe centrale et Slaves
ISO-8859-3	Latin3-langues Espéranto, Galicienne, Maltaise et Turc.
ISO-8859-4	Latin4-langues Estonienne, Lettonne et Lithuanienne.
ISO-8859-5	Langue Cyrilliques.
ISO-8859-6	Langue Arabe.
ISO-8859-7	Langue Grecque.
ISO-8859-8	Langue Hébraïque.
ISO-8859-9	Latin5-Langue Turc.
ISO-8859-10	Latin-6Langue Groenlandaises et Lapones.

Tableau 1- Les types de codage des caractères.

Si la déclaration XML ne comporte pas l'attribut *encoding* alors l'encodage par défaut sera le jeu de caractères Unicode compressé **[REMC 02]**

L'attribut *standalone* fait référence à l'autonomie du document. Si la DTD est interne, le document est autonome et la valeur de l'attribut standalone peut être définie à *yes*. Si la DTD référencée est externe la valeur de cet attribut doit être définie à *no*. Si cet attribut est omis, c'est la valeur no qui est prise par défaut.

Cette déclaration est facultative, mais il est préférable de l'utiliser, auquel cas les attributs *version, encoding* et *standalone* doivent être placés *dans cet ordre*. Si elle est utilisée, elle doit être placée en toute *première ligne* du document XML. **[ALMI 01]**

7.1.2 Les instructions de traitement

La déclaration XML se poursuit avec des informations facultatives sur des instructions de traitement à destination d'applications particulières.

Les instructions de traitement sont des instructions interprétées par l'application servant à traiter le document XML. Elles permettent aux documents de contenir des instructions pour des applications afin de fournir des informations supplémentaires sur le document aux analyseurs syntaxiques. Leur syntaxe est la suivante :

```
<?Cible [Données de l'instruction de traitement]?>
```

Les constituants de la balise sont :

<?	Cible	Données de l'instruction de traitement	?>
Ouverture de la balise	Un nom XML	chaîne de caractères que l'analyseur XML passera inchangé à l 'application identifiée.	Fermeture de la Balise

La plus petite de ces instructions est sûrement celle constituant le prologue d'un document XML :

```
< ? XML version="1. 0" ?>
```

7.1.3 Déclaration de type de document (DTD)

Lorsque cette déclaration est présente, elle permet de définir la structure du document. Elle peut être de deux types, externe ou interne. Exemple de déclaration de type de document :

```
<!DOCTYPE racine SYSTEM "racine.dtd">
```

Ce type de déclaration est celui d'une déclaration de type de document externe. Elle définit l'ensemble des éléments utilisables dans le document, y compris l'élément-racine (ici racine) ainsi que le nom de fichier racine.dtd dans lequel se trouve définie la structure du document.
Bien que facultative, il est souvent très intéressant de posséder une DTD, simplement pour vérifier la validité du document XML.

L'autre type de document permettant de définir la structure d'un fichier, le Schéma XML qui s'utilise autrement, comme nous le verrons plus tard.

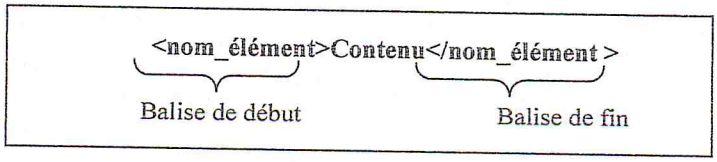
7.2 L'arbre des éléments XML :

Tout document XML est représenté sous la forme d'un arbre d'éléments. Comme tout arbre, il comporte une racine, des branches et des feuilles et qui représentent respectivement l'élément racine, les éléments et les attributs. Ces composants sont les constituants les plus élémentaires d'un document XML .

Cet arbre est constitué d'éléments imbriqués les uns dans les autres (ayant une relation père_ enfant) et d'éléments adjacents. Il représente le véritable contenu du document XML. **[ALMI 01]**

7.2.1 Les éléments :

L'unité de base d'un document XML et dite « élément ». Les éléments permettent d'identifier les différentes sections d'un document XML. Il se présente sous la forme suivante :



Il est pointé par deux marqueurs : un marqueur d'ouverture qui place le nom de l'élément entre un chevron ouvrant (<) et un chevron fermant (>) et un marqueur de fermeture qui ressemble au premier à l'exception du slash (/) qui apparaît devant le nom de l'élément. Le marqueur d'ouverture est appelé balise ouvrante et le marqueur de fermeture est appelé balise fermante. Comme XML est extensible, il est possible de créer des balises donc écrire soi-même le nom des balises utilisées. Ces balises XML encadrent le contenu de l'élément :
La syntaxe d'un élément est la suivante :

<Nom_élément Nom-attr$_1$=' val$_1$' ... Nom-attribut$_n$='val$_n$'>
Contenu de l'élément
</Nom_élément >

Toutes les balises doivent être équilibrées : tous les éléments qui pourraient contenir des données textuelles doivent avoir une balise de début et une balise de fin (l'omission est interdite sauf pour les éléments vides). Comme le XML est stricte, toutes les balises ouvertes doivent être fermées.

Dans un document XML, les éléments peuvent être imbriqués mais ne doivent en aucun cas se recouvrir. Les marqueurs doivent être emboîtés correctement. Le XML étant très préoccupé par la structure des données, des balises mal imbriquées sont des fautes graves de sens.

7.2.1.1 Les noms XML :

Les noms de balises ou bien les noms XML ou bien encore les noms d'éléments reflètent le type de leur contenu et non la façon dont le contenu devrait s'afficher. Pour composer un nom XML on doit respecter les règles suivantes :

- Les noms peuvent contenir des lettres, des chiffres ou d'autres caractères.
- Les noms ne peuvent débuter par un nombre ou un signe de ponctuation.
- Les noms ne peuvent commencer par les lettres xml (ou XML ou Xml. . .).
- Les noms ne peuvent contenir des espaces.
- La longueur des noms est libre mais on conseille de rester raisonnable.
- Certains signes qui pourraient selon les logiciels, prêter à confusion comme "-", ";", ". ", "<", ">" sont interdits.
- Le caractère ":" est autorisé dans un nom XML (dans le cas des espaces de noms ou bien les domaines nominaux qui sera décrit plus loin).
- Les caractères spéciaux pour les francophones comme é, à, ê, ï, ù sont à priori permis mais pourraient être mal interprétés par certains programmes.

Un document XML est sensible à la casse, un nom d'élément minuscule diffère du même écrit en majuscule ou du mixage des deux. Dans ce cas le respect de la logique de chaque élément est indispensable c'est à dire qu'un nom de balise de début doit être écrit sous la même forme qu'un nom de balise de fin. Plus précisément, La balise d'ouverture et la balise de fermeture doivent être identiques, donc il faut être très rigoureux dans l'écriture de leurs libellés. Toutefois une tendance se dégage pour n'écrire les balises qu'en minuscules, limitant ainsi les erreurs possibles.

7.2.1.2 Elément racine :

La première paire de balises d'un document XML sera considérée comme la balise de racine. Elle est unique et obligatoire. Elle encadre le contenu du document XML à produire : tous les autres éléments seront imbriqués entre les balises de l'élément racine.

7.2.1.3 Contenu d'un élément:

Le texte placé entre les deux balises d'ouverture et de fermeture présente le contenu d'un élément, il est considéré comme partie intégrante de l'élément et il est formaté en conformité avec les règles qui gouvernent cet élément (selon le type de l'élément). Un élément, s'il n'est pas vide, peut contenir des données textuelles, des commentaires, des espaces (retour de chariot, retour à la ligne ou tabulation), des références à des entités, des sections littérales et peut contenir aussi d'autres éléments.

Certains caractères ayant une signification particulière dans la grammaire du XML restent interdits tel que « < », « > », « & », « " » et « ' » dans un contenu. De cet effet, les caractères pouvant poser des problèmes à l'affichage, le XML utilise d'autres caractères de masquage ou d'échappement pour les remplacer soit sous forme d'entités nommées, soit sous forme d'entités codées.

Caractère	Entité nommée	Entité codée (entité caractères)
<	<	&#60 ;
>	>	&#62 ;
&	&	&#38 ;
"	"	&#34 ;
'	'	&#39 ;

Tableau 2- Les cinq entités caractères prédéfinies.

Lorsqu'un document XML inclut des exemples de code source XML ou HTML, les caractères '<' et '&' doivent être encodés et cette solution n'est pas souhaitable pour un document contenant de nombreuses sections de code. Pour palier à ce problème, et pour qu'il ne soit nécessaire de structurer les caractères spéciaux par des caractères de masquage la section littérale **CDATA** intervient. Toute partie comprise entre < ![CDATA[et]]> est traitée comme une données textuelle brute et elle ne peut pas être imbriquée La mise en œuvre d'une section CDATA s'écrit ainsi :

```
< ! [CDATA [Données textuelles]]>
```

7.2.1.4 Les éléments vides :

En dehors des éléments standard, XML supporte également des éléments vides c'est-à-dire des éléments qui peuvent être dépourvus de contenu. Un élément vide ne comporte pas de texte entre la balise d'ouverture et celle de fermeture. L'élément vide peut avoir des attributs et il est représenté par la syntaxe suivante :

```
<Nom-element Nom-attribut1='val1' ... Nom-attribut n='valn'></Nom-element>
                              Ou bien
<Nom-element Nom-attribut 1='val1' Nom-attribut 2='val2'...Nom-attribut n='valn'/>
```

7.2.2 Les attributs :

Un attribut est un mécanisme qui permet d'ajouter des informations descriptives à un élément. Ils sont souvent utilisés pour affiner ou modifier le comportement d'un élément. C'est une paire « nom="valeur" » associée à la balise de début de l'élément. Les noms sont séparés des valeurs par le signe « = » et parfois des blancs.

```
<Nom_element Nom_attribut1=' val1' Nom_attribut2 .....Nom_attribut2='valn'>
                    contenu de l'élément
                    </Nom_element >
```

Les noms d'attributs sont aussi des noms XML et un élément ne doit pas contenir deux attributs avec le même nom.

En XML, la valeur de l'attribut doit obligatoirement être entre des guillemets «" » ou bien des apostrophes « ' » quand la valeur de l'attribut contient elle même des guillemets mais jamais par un mélange des deux. Les attributs ne peuvent être hiérarchiques : ils ne peuvent contenir de sous éléments. La valeur d'un attribut est une donnée textuelles brute, mais elle ne peut inclure les caractères « & » , « < » et «" » et « ' », ils doivent être échappés. Les valeurs des attributs dépendent elles aussi de la casse.

7.2.2.1 Les attributs prédéfinis :

Il existe en XML deux attributs prédéfinis dont nous allons parler : il s'agit de **xml:lang** et **xml:space**. Ils permettent, respectivement, d'indiquer la langue utilisée dans une partie du document et de dire ce que l'on va faire des caractères de séparation.

Dans les deux cas, il faut les redéfinir dans la DTD pour pouvoir utiliser un système de validation (malgré le fait qu'ils soient prédéfinis en XML).

Si l'un de ces deux attributs est utilisé sur un tag (balise), cet attribut et sa valeur seront aussi cascadés sur tous les sous-tags. Il est donc possible de ne spécifier qu'une unique fois l'attribut **xml:lang** (par exemple) pour tout le document ; il suffit de le définir sur le tag racine.

- **xml:lang** :

  Permet de spécifier la langue utilisée par les données.

```
<Nom_element xml:lang = "Code">contenu</Nom_element>
```

  L'attribut **Code** est représenté sous la forme suivante : **code- sous_code**

  code : représente la langue du pays.
  Sous_code : il indique le code du pays. La valeur de cet attribut est codée sur les deux premiers caractères du nom du pays.

Ces deux sous_attributs sont représentés chacun par les deux premières lettres. La convention veut que le code s'écrit en minuscule et le sous_code en majuscule. Ils sont représentés par les deux premières lettres de la langue et du code du pays.

- **xml:space** : Cet attribut indique si un espace blanc à l'intérieur d'un élément est significatif et ne doit pas être altéré par le processeur XML. Il accepte deux valeurs. La valeur par défaut est "default" et la seconde est "preserve".
- **default** : Le processeur XML est libre de faire ce qu'il désire avec les espaces blancs à l'intérieur de l'élément.
- **preserve** : Les espaces seront préservés tels qu'ils sont dans le document source. Les caractères de séparation seront préservés ; les outils de traitement de données XML pourront ainsi les utiliser. Par caractères de séparation, on entend les espaces, les tabulations et les caractères de passage à la ligne.

7.2.3 Les Commentaires

Le XML nous permet d'introduire des commentaires. Ce ne sont pas des éléments, ils sont utilisés pour éclaircir le code source et le rendre lisible pour une personne. Ils commencent par < !-- et se termine par -- >. Leur forme est la suivante :

```
<!-- commentaire -->
```

Le signe «--» ne doit pas apparaître dans le commentaire ainsi que le triple trait d'union. Ils peuvent être placés avant ou après l'élément racine et dans la donnée textuelle. Par contre, ils ne doivent pas apparaître à l'intérieur des balises ou dans un autre commentaire (commentaires imbriqués). Le contenu des commentaires n'est pas obligatoirement transmis. Les parseurs peuvent ne pas le faire.

8. Les espaces de noms

Le concept d'espace de noms se rapporte à l'idée que chaque élément possède un jeu donné d'attributs et que chaque attribut possède un nom et une valeur. Aucun problème ne se pose lorsque les noms des éléments ne sont pas ambigus ; dès qu'on combine deux documents qui utilisent le même nom d'élément dans différents contextes, la confusion est semée et on arrive pas à distinguer de quel élément s'agit il. C'est précisément pour lever ce genre d'ambiguïté que la spécification XML définit les **espaces de noms**.

Les espaces de noms répondent à deux objectifs pour XML : **[ERWS 02]**

a- Distinguer des éléments et attributs issus de vocabulaires différents avec des significations différentes et pouvant être amenés à partager le même nom.
b- Grouper tous les éléments et attributs relatifs à une même application XML pour que les logiciels puissent les reconnaître facilement.

Les espaces de nom permettent d'identifier de manière unique des éléments spécifiques dans un document XML. Pour cela, on ajoute un préfixe unique à un élément, l'associant ainsi à un ensemble de données.
La mise en place d'un espace de noms est très simple. La syntaxe est la suivante :

```
<nomElement xmlns:prefixe="URI">
```

9. Les feuilles de style :

Le XML est un langage très attrayant pour écrire et servir des pages Web. Pour pouvoir afficher des documents XML, il serait trop prétentieux d'attacher des feuilles de style à un document XML pour lui donner les instructions nécessaires au rendu de chaque élément. Le contenu doit être formaté et présenté aux utilisateurs. Donc, pour ceci, des informations de formatage sont appliquées au document XML et le balisage sémantique est transformé en un langage de présentation.

Les feuilles de style permettent de mettre en forme une page Web d'une manière équivalente à celle d'un magazine ou d'un journal de la presse écrite créée par un logiciel de publication assistée par ordinateur (PAO). A l'aide des feuilles de style, la gestion des différents éléments de présentation comme les titres, les paragraphes, les images ou bien les tableaux d'un document XML devient plus pratique, et améliore la cohérence et l'ordre au sein de l'ensemble des pages d'un site Internet. Donc, une feuille de style permet à l'auteur de mieux contrôler la mise en page d'un document XML. Or, comme dans tout système où les fichiers peuvent être visualisés au hasard par des utilisateurs arbitraires, l'auteur ne peut pas connaître les ressources (polices par exemple) disponibles sur le système de l'utilisateur; des précautions sont donc de rigueur. La feuille de style est associée au document par une instruction de traitement xml-stylesheet :

```
<?xml stylesheet href=" URL-feuille" type=" type-feuille"?>
```

La feuille de style est associée au document par une instruction de traitement xml-stylesheet dans le prologue : après la déclaration XML et avant la balise de début de l'élément racine du document. Cette instruction de traitement utilise des pseudo_attributs pour décrire la feuille de style.
Il existe deux types de pseudo-attributs :

Pseudo-attributs obligatoires

href : indique l'URL où la feuille de style peut être trouvée.
type : indique le type MIME de la feuille de style : text/css pour une feuille de style CSS, text/xml ou application/xml pour une feuille de style XSLT.

Pseudo-attributs optionnels

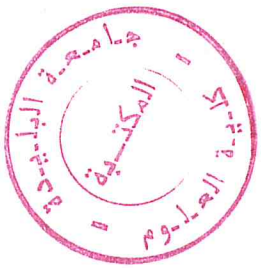
Media : contient une information sur le media utilisé par la feuille de style. Il peut être seul ou dans une liste ou les éléments sont séparés par des virgules. Par exemple : screen, tty, projection,...
Charse : indique dans quel encodage est écrite la feuille de style.
Alternate : indique si la feuille de style est la principale pour un media donné ou une alternative dans des cas particuliers. Elle peut prendre la valeur « no » ou « yes ». Sa valeur par défaut vaut « no », elle indique qu'il s'agit de la feuille de style principale, si elle vaut « yes » le navigateur peut donner à l'utilisateur le choix d'autres feuilles de style dans ce cas il utilise le pseudo attribut title.

Title : il est utilisé quand alternate vaut « yes », il indique à l'utilisateur en quoi la feuille de style diffère (par exemple le choix de la police grande, moyenne,…).

Quand le choix d'une feuille de style n'est pas demandé par un navigateur, le premier choix qui c.rrespond le mieux au type media de l'environnement est pris en considération.

Les langages de feuilles de style actuels majeurs sont :

- Cascading StyleSheet (CSS).
- XSL Formatting Objects (XSL-FO).



10. les outils XML

Les outils XML liés au XML se répartissent en une grande variété de catégories. Les groupements majeurs sont :

- **Les éditeurs :** Il existe des éditeurs pour les DTD, pour les schémas et pour les documents XML complets.
- **Les convertisseurs :** Les convertisseurs sont conçus pour traduire des documents d'un langage de balisage à un autre.
- **Les parseurs :** Le principale rôle des parseurs est de parser et d'interpréter les documents XML.
- **Les outils de stockage et de gestion :** Les outils pour le stockage et la gestion des documents vont des bases de données aux moteurs de recherche. Il s'agit d'un domaine de développement rapide, et de nouveaux produits font leurs apparitions fréquemment.
- **Les outils de restitution :** La restitution des documents est également un domaine de développement passionnant. Cette catégorie inclut les outils de publication, les navigateurs Web et les agents logiciels.

11. En résumé :

XML est devenu le langage universel d'échange d'informations associant aux données une structure sémantique (sous forme d'éléments et d'attributs) laquelle permet de comprendre le sens (la signification) de la donnée. La structure en question est définie selon un modèle (sous forme de schémas ou de DTD) qui donne les règles d'assemblage des données.
XML apparaît de plus en plus comme un outil stratégique, garantissant l'interopérabilité des formats de données et fournissant une structuration standardisée. Il est parfaitement adapté à la prochaine génération d'applications Internet, au commerce électronique et au marketing.

Chapitre 3 :
Etude des mécanismes des Schéma XML

1 . Introduction

Les définitions de types de documents (DTD) sont des outils de validation courant pour les document XML, bien qu'elles possèdent des avantages, dont le fait d'être implémenter de langue date, elles ont aussi plusieurs désavantages, dont les suivants :

- Premièrement, la syntaxe des DTD est issue du langage EBNF (Extented Backus Naur Form), beaucoup plus compliqué et lourd que ne l'est le langage XML . Cela signifie qu'il est nécessaire d'utiliser un outil spécial pour "parser" un tel fichier, différent de celui utilisé pour l'édition du fichier XML.

- Deuxièmement, les DTD ne supportent pas les "espaces de nom". En pratique, cela implique qu'il n'est pas possible, dans un fichier XML défini par une DTD, d'importer des définitions de balises définies par ailleurs.

- Troisièmement, le "typage" des données est extrêmement limité.

L'adoption d'un langage de définition des documents basé sur XML, était donc inévitable afin de conserver une homogénéité dans la sphère XML déjà fort bien pourvue.
Pour cela le W3C a conçu le langage de schéma XML qui apporte une grande souplesse et une puissance inégalée dans la définition des documents XML, en passant par les étapes suivantes :

- octobre 1998 : création du XML Schema WG
- février 1999 : publication de la note "XML Schema Requirements"
- mars 1999 : publication des premiers drafts
- mai 2001 : recommandation XML Schema 1.0 en 3 documents
  - Primer
  - Structures
  - Datatypes
- juillet 2002 : nouveau charter (maintenance)

Conçu pour palier aux déficiences pré-citées des DTD, XML Schema propose, en plus des fonctionnalités fournies par les DTD, des nouveautés :

- Un grand nombre de types de données intégrées comme les booléens, les entiers, les intervalles de temps, etc. De plus, il est possible de créer de nouveaux types par ajout de contraintes sur un type existant.
- Des types de données utilisateurs qui vous permettent de créer votre propre type de données nommé.
- La notion d'héritage. Les éléments peuvent hériter du contenu et des attributs d'un autre élément. C'est sans aucun doute l'innovation la plus intéressante de XML Schema.
- Le support des espaces de nom.
- Les indicateurs d'occurrences des éléments peuvent être tout nombre non négatif.
- Une grande facilité de conception modulaire de schémas.

Le langage de schémas constitue donc une pièce essentielle dans la sphère XML, assurant une évolutivité, un dynamisme et une souplesse que ne pouvait assumer le langage de DTD.

2. Structure de base

Comme tout document XML, un Schema XML commence par un prologue et a un élément racine. [REMC 02]

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema>
    <!-- déclarations d'éléments, d'attributs et de types ici -->
</xsd:schema>
```

L'élément racine est l'élément *xsd:schema*. Il faut retenir que tout éléments et attributs d'un schéma doivent commencer par le préfixe *xsd* qui identifie l'espace de noms au quel ils font référence.

3. Déclarations d'éléments et d'attributs

3.1. Déclarations d'éléments

Un élément, dans un schéma, se déclare avec la balise <xsd:element>. Par exemple,

```
<xsd:element name="  nom élément " type="type élément"/>
```

Chaque élément déclaré est associé à un type de données via l'attribut type. Les éléments pouvant contenir des sous-éléments ou porter des attributs sont dits de types *complexes*, tandis que les éléments ne contenant pas de sous-éléments sont dits de types *simples*.

3.2. Déclarations d'attributs

3.2.1. Déclaration simple

A la différence des éléments, un attribut ne peut être que de type simple. Cela signifie que les attributs, ne peuvent contenir d'autres éléments ou attributs. [ALMI 01]. Pour la déclaration d'attributs on utilise l'élément xsd :*attribute*

Exemple :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
  <xsd:element name=" nom élément" type="type élément"/>
     <xsd:attribute name=" nom attribut" type=" type attribut">
  </xsd:schema>
```

Un attribut peut avoir un indicateur d'occurrences.

3.2.2. Contraintes d'occurrences

L'élément *attribute* de XML Schema peut avoir trois attributs optionnels: *use, default* et *fixed*. Selon les valeurs prises par ces attributs, l'attribut défini considéré peut ne pas apparaître ou avoir une valeur par défaut.

Le tableau suivant fournit une description exhaustive des cas selon la valeur des attributs *use, fixed* et *value.*

Valeur de l'attribut use	Valeur de l'attribut value	Effet
Required	-	L'attribut doit apparaître, en prenant n'importe quelle valeur
Required	21	l'attribut doit apparaître, en prenant comme valeur 21
Optional	-	l'attribut peut apparaître, il peut avoir n'importe quelle valeur
Fixed	21	l'attribut peut apparaître, s'il est présent sa valeur doit être 21, s'il n'apparaît pas sa valeur est 21
Default	21	l'attribut peut apparaître, s'il n'est pas présent sa valeur est 21, autrement sa valeur est celle donnée
Prohibited	-	l'attribut ne doit pas apparaître !

Tableau 3 - Contraintes d'occurrences pour les attributs

3.3. Groupes d'attributs

L'utilisation d'un groupe d'attributs permet d'améliorer la lisibilité et facilite la maintenance d'un schéma parce qu'un groupe d'attributs peut être défini à un seul endroit et référencé dans beaucoup de déclarations et de définitions. **[REMC 02]**

Signalons également qu'un groupe d'attributs peut contenir un autre groupe d'attributs.

Exemple :

```
<xsd:attributeGroup  name="nom du groupe d'attribut">
    <xsd:attribute       name="nom attribut1"  Type="type attribut1">
                  ⋮
<xsd:attribute  ame="nom attribut n"  Type="type attribut n">

</xsd: attributeGroup >
```

4. les types de données des schémas XML

XML Schema permet de spécifier des types de données bien plus finement que le langage DTD. Il distingue notamment types simples et types complexes. **[REMC 02]**

4.1. Types simples :

Les types de données simples ne peuvent comporter ni attributs, ni éléments enfants. Il en existe de nombreux, prédéfinis, mais il est également possible d'en "dériver" de nouveaux. Enfin, il est possible de déclarer des "listes" de types.

4.1.1. Types de données prédéfins

Les types de données prédéfinies sont divisés en deux ensembles distincts, les primitifs et les dérivés, les seconds découlant des premiers. [SILA 00]

4.1.1.1. Types de données primitifs :

Sont la base de tous les autres types de données.

Type	Description
string	représente une chaîne de caractères.
boolean	représente une valeur booléenne *true* ou *false*.
decimal	représente un nombre décimal
float	représente un nombre à virgule flottante.
double	représente un nombre réel double.
duration	représente une durée.
dateTime	représente une valeur date/heure.
time	représente une valeur horaire (format : hh:mm:ss.sss).
date	représente une date (format : CCYY-MM-DD).
gYearMonth	représente un mois et une année grégorienne (format : CCYY-MM).
gYear	représente une année (format : CCYY).
gMonthDay	représente le jour d'un mois (format : MM-DD).
gDay	représente le jour d'un mois (format : DD).
gMonth	représente le mois (format : MM).
hexBinary	représente un contenu binaire hexadécimal.
base64Binary	représente un contenu binaire de base 64.
anyURI	Représente une adresse URI (ex.: http://www.site.com).
QName	Représente un nom qualifié.
NOTATION	Représente un nom qualifié.

Tableau 4 - Les types de données primitifs

4.1.1.2. Les types de données dérivés :

Sont construits à partir des types de données primitifs.

Type	Description
normalizedString	Représente des chaînes normalisées comportant des espaces blancs. Ce type de données est dérivé de **string**.
Token	Représente des chaînes sous forme de jetons. Ce type de données est dérivé de **normalizedString**.
Language	Représente des identificateurs de langage naturel (définis par la norme RFC 1766). Ce type de données est dérivé de **token**..
NMTOKEN	Représente le type d'attribut **NMTOKEN**. Un **NMTOKEN** est un ensemble de caractères de nom (lettres et chiffres, entre autres) organisés selon une combinaison quelconque. Contrairement à **Name** et à **NCName**, **NMTOKEN** peut utiliser n'importe quel caractère de départ. Ce type de données est dérivé de **token**.
NMTOKENS	Représente le type d'attribut **NMTOKENS**. Contient un ensemble de valeurs de type **NMTOKEN**.

Name	Représente des noms en XML. Un **Name** est un jeton qui commence par une lettre, un trait de soulignement ou un signe deux-points et se poursuit par des caractères de nom (lettres et chiffres, entre autres). Ce type de données est dérivé de **token**.
NCName	Représente des noms ne comportant pas de signes deux-points. Ce type de données est identique à **Name**, si ce n'est qu'il ne peut pas commencer par un signe deux-points. Ce type de données est dérivé de **Name**.
Id	Représente le type d'attribut **ID** défini dans la recommandation « Langage de balisage extensible (XML) 1.0 » du W3C. Cet **ID** doit être un NCName (*No-Colon-Name*, nom sans deux-points) et être unique au sein d'un document XML. Ce type de données est dérivé de **NCName**.
IDREF	Représente une référence à un élément dont l'attribut **ID** correspond à l'**ID** spécifié. Un **IDREF** doit être un NCName et la valeur d'un élément ou d'un attribut de type ID présent dans le document XML. Ce type de données est dérivé de **NCName**.
IDREFS	Représente le type d'attribut **IDREFS**. Contient un ensemble de valeurs de type **IDREF**.
ENTITY	Représente le type d'attribut **ENTITY** défini dans la recommandation « Langage de balisage extensible (XML) 1.0 » du W3C. Il s'agit d'une référence à une entité non analysée dont le nom correspond à celui qui a été spécifié. Un **ENTITY** doit être un NCName et être déclaré dans le schéma en tant que nom d'une entité non analysée. Ce type de données est dérivé de **NCName**.
ENTITIES	Représente le type d'attribut **ENTITIES**. Contient un ensemble de valeurs de type **ENTITY**.
Integer	Représente une séquence de chiffres décimaux, éventuellement précédés d'un signe (+ ou -). Ce type de données est dérivé de **decimal**.
nonPosi .iveInteger	Représente un entier inférieur ou égal à zéro. Un **nonPositiveInteger** est constitué d'un signe moins (-) et d'une séquence de chiffres décimaux. Ce type de données est dérivé de **integer**.
negativeInteger	Représente un entier inférieur à zéro. Il est constitué d'un signe moins (-) et d'une séquence de chiffres décimaux. Ce type de données est dérivé de **nonPositiveInteger**.
Long	Représente un entier compris entre -9223372036854775808 et 9223372036854775807, inclus. Ce type de données est dérivé de **integer**.
Int	Représente un entier compris entre -2147483648 et 2147483647, inclus. Ce type de données est dérivé de **long**.
Short	Représente un entier compris entre -32768 et 32767, inclus. Ce type de données est dérivé de **int**.
Byte	Représente un entier compris entre -128 et 127, inclus. Ce type de données est dérivé de **short**.
NonNegativeInteger	Représente un entier supérieur ou égal à zéro. Ce type de données est dérivé de **integer**.
unsignedLong	Représente un entier compris entre zéro et 18446744073709551615, inclus. Ce type de données est dérivé de **nonNegativeInteger**.

unsignedInt	Représente un entier compris entre zéro et 4294967295, inclus. Ce type de données est dérivé de **unsignedLong**.
unsignedShort	Représente un entier compris entre zéro et 65535, inclus. Ce type de données est dérivé de **unsignedInt**.
unsignedByte	Représente un entier compris entre zéro et 255, inclus. Ce type de données est dérivé de **unsignedShort**.
positiveInteger	Représente un entier supérieur à zéro. Ce type de données est dérivé de **nonNegativeInteger**.

Tableau 5- Les types de données dérivés

C'est une nouveauté importante, puisque les DTD se contentaient de types de données plutôt limités (PCDATA, CDATA…).

Les types de données simples les plus courants sont représentés : chaînes de caractères, entiers, dates, réels etc. Nous remarquerons aussi que les types de données des DTD ont été repris par XML Schema pour des raisons de compatibilité ascendante.

4.1.2. Listes

Les types listes sont des suites de types simples (ou *atomiques*) séparés par des blancs. XML Schema possède trois types de listes intégrés: NMTOKENS, ENTITIES et IDREFS. Il est également possible de créer une liste personnalisée, par "dérivation" de types existants. Par exemple, NMTOKENS est un type liste constitué d'une série d'éléments NMTOKEN délimités par un espace blanc faisant office de séparateur.
La syntaxe associée est :

```
<xsd:simpleType name="nm type liste">
<xsd:list itemType="nom type atomique"/>
</xsd:simpleType>
```

4.1.3. Unions

Les listes et les types simples intégrés ne permettent pas de donner un choix sur le type de contenu d'un élément. On peut désirer, par exemple qu'un type autorise soit un nombre, soit une chaîne de caractères particuliers. Il est possible de le faire à l'aide d'une déclaration d'union.

La syntaxe associée a l'union est :

```
<xsd:simpleType name="nom typa union">
    <xsd:union memberTypes="nom type1   nom type 2  ….. nom type n "/>
</xsd:simpleType>
```

4.1.4. Bibliothèque de types simples

A l'instar des types simples fournis par tout langage de programmation comme Java ou C, XML Schema comporte une bibliothèque intégrée de types simples. La figure suivante montre une représentation sous forme de graphe d'héritage. **[ERWS 02]**

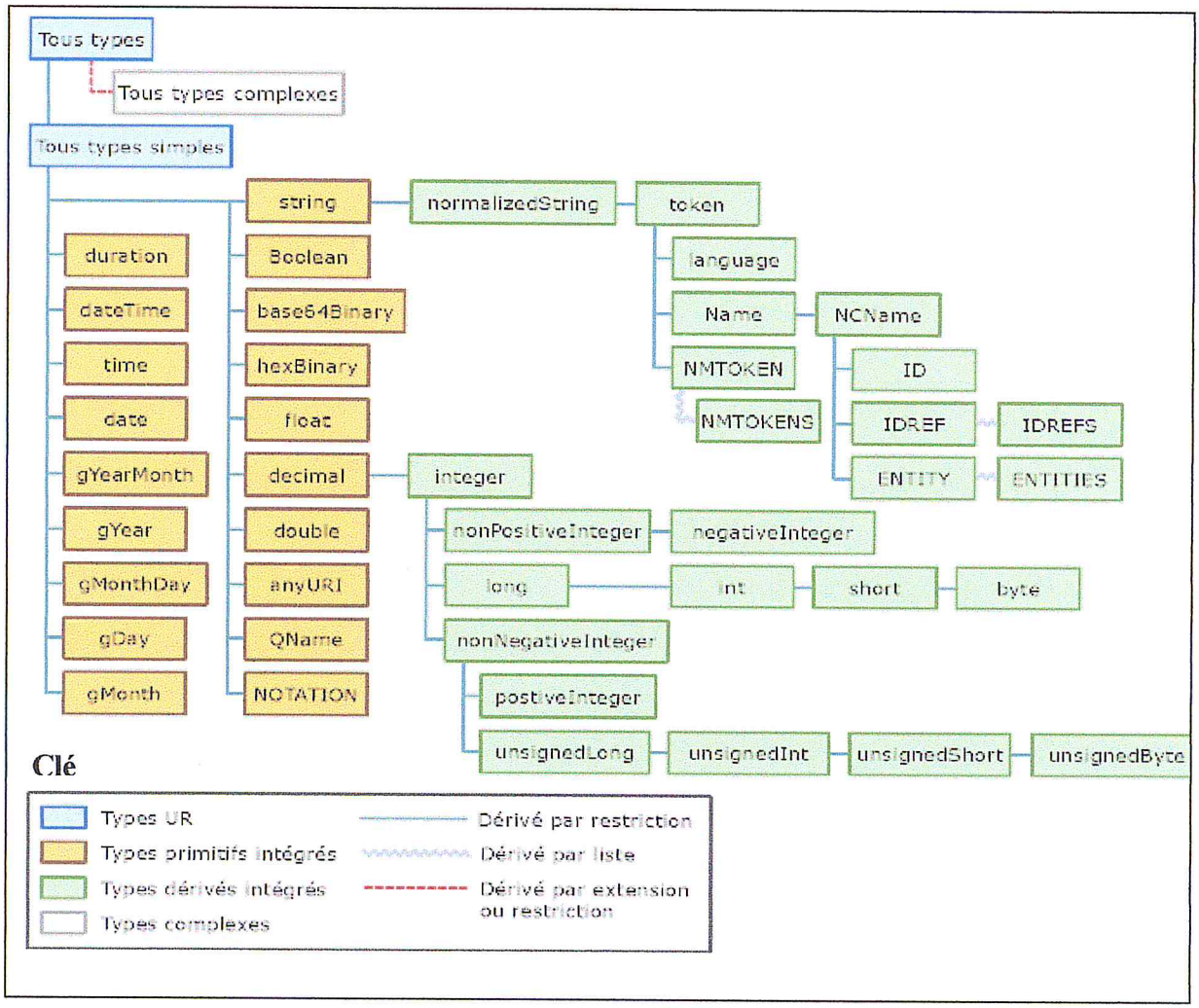


Figure 1- Hiérarchie des types simples.

Le support étendu du typage par XML Schema lui donne déjà un avantage décisif par rapports aux DTD. Cependant, les types de données simples ne permettent pas aux éléments de contenir des sous-éléments. Pour cela, XML Schema définit les types de données complexes, pouvant être créés par un auteur de schéma.

4.2. Les types complexes

Nous verrons dans cette section comment sont implémentés les contraintes d'occurrences ainsi que les connecteurs de séquences et de choix.

4.2.1. Modèle de contenu

Un type de données complexe peut être caractérisé par son modèle de contenu ; c'est-à-dire comment ses sous-éléments sont susceptibles d'apparaître. XML Schema propose un ensemble de connecteurs permettant de représenter n'importe quel modèle de contenu.

4.2.2. Le connecteur de séquence

Le connecteur de séquence permet de définir un type complexe comportant des suites d'éléments. Il impose la présence, dans le même ordre que la déclaration, de tous les éléments fils de *xsd:sequence*. Pour pouvoir rendre des éléments optionnels, il faut appliquer des contraintes d'occurrences sur les éléments.

La syntaxe associée est :

```
<xsd:complexType name="nom du type complexe">
    <xsd:sequence>
        <xsd:element name="nom élément fils 1" type="type élément fils 1"/>
                    .
                    .
                    .
        <xsd:element name="nom élément fils n" type="type élément fils n"/>
    </xsd:sequence>
</xsd:complexType>
```

4.2.3. Le connecteur de choix

Il permet de choisir entre plusieurs éléments qui seront déclarés à l'intérieur de l'élément *xsd:choice*. Un seul élément pourra être présent dans le document instance.
La syntaxe associée est :

```
<xsd:complexType name="nom type complexe">
        <xsd:choice>
            <xsd:element name="nom élément fils 1" type="type élément fils 1"/>
                        .
                        .
                        .
            <xsd:element name="nom élément fils n" type="type élément fils n"/>
        </xsd:choice>

</xsd:complexType>
```

4.2.4. L'élément all

Cet élément est une nouveauté par rapport aux DTD. Il indique que les éléments enfants doivent apparaître une fois (ou pas du tout), et dans n'importe quel ordre. Cet élément *xsd:all* doit être un enfant direct de l'élément xsd:complexType. **[REMC 02]**

Exemple

```
<xsd:complexType name="nom type complexe">
    <xsd:all>
        <xsd:element name="nom élément fils 1" type="type élément fils 1"/>
                                    ⋮
        <xsd:element name="nom élément fils n" type="type élément fils n"/>
    </xsd:all>
</xsd:complexType>
```

Les fils de l'élément xsd:all doivent impérativement apparaître au plus une fois, ce qui signifie pour leurs indicateurs d'occurrences que *minOccurs*=0 et *maxOccurs*=1. Nous allons voir maintenant ces indicateurs plus en détails.

4.2.5. Contraintes d'occurrences sur les éléments

Les DTD ont des contraintes d'occurrences qui s'inspirent de celles des grammaires de type EBNF (Extended Backus Naur Form). Elles ont donc des contraintes d'occurrences limitées à 0,1 ou l'infini. XML Schema est beaucoup plus souple puisque tout nombre entier non négatif peut être utilisé pour fixer la valeur du nombre maximum et/ou minimum d'occurrences. Lorsqu'ils sont définis localement, les éléments d'un schéma peuvent comporter les attributs minOccurs et maxOccurs.

La syntaxe associée est :

```
<xsd:element name="nom élément" type="type élément" minOccurs="val1"/>
                        ou
<xsd:element name="nom élément" type="type élément" maxOccurs="val2"/>
                        ou
<xsd:element name="nom élément" type="type élément" minOccurs="val1"
      maxOccurs="val2"/>
```

La valeur spéciale 'unbounded' peut être donnée à l'attribut maxOccurs pour spécifier qu'un élément peut apparaître un nombre illimité de fois dans le document instance. Le tableau suivant donne une équivalence entre les contraintes d'occurrences des DTD et de XML Schema.

Dans une DTD	Valeur de minOccurs	Valeur de maxOccurs
*	0	unbounded
+	1 (pas nécessaire, valeur par défaut)	unbounded
?	0	1 (pas nécessaire, valeur par défaut)
Rien	1 (pas nécessaire, valeur par défaut)	1 (pas nécessaire, valeur par défaut)

Tableau 6 - Correspondance des contraintes d'occurrences (DTD et XML schema).

En plus des attributs *minOccurs* et *maxOccurs*, un élément déclaré localement peut porter les attributs *fixed* et *default*.
Le tableau suivant fournit une description exhaustive des cas selon la valeur des attributs précités. (*minOccurs* et *maxOccurs* , *fixed* et *default.*)

Eléments (minOccurs, maxOccurs) fixed, default	Attributs use, fixed, default	Remarques
(1, 1) -, -	required, -, -	l'élément ou l'attribut doit apparaître une seule fois, il peut prendre n'importe quelle valeur
(1, 1) 21, -	required, 21, -	l'élément ou l'attribut doit apparaître une seule fois, sa valeur doit être 21
(2, unbounded) 21, -	Non applicable	l'élément doit apparaître deux fois ou plus, sa valeur doit être 21; en général, les valeurs de minOccurs et maxOccurs sont des entiers positifs, mais maxOccurs peut également prendre la valeur "unbounded"
(0, 1) -, -	optional, -, -	l'élément ou l'attribut peut apparaître 0 ou une fois et peut prendre n'importe quelle valeur
(0, 1) 21, -	optional, 21, -	l'élément ou l'attribut peut apparaître 0 ou une fois; quand il apparaît, sa valeur doit être égale à 21; si l'attribut est omis, sa valeur est fixée à 21. Si l'élément est omis, le processeur ne lui impose pas une valeur particulière.
(0, 1) -, 21	optional, -, 21	l'élément ou l'attribut peut apparaître 0 ou une fois; quand il apparaît, sa valeur est libre; si l'attribut est omis, sa valeur par défaut est 21. Si l'élément est omis, sa valeur n'est pas imposée ; si l'élément est vide, sa valeur est alors 21.
(0, 2) -, 21	Non applicable	l'élément ou l'attribut peut apparaître 0, une ou deux fois; il n'y un pas de possibilité équivalente pour les attributs. la valeur du contenu de l'élément n'est pas fixée mais si l'élément est vide la valeur considérée par le processeur sera, par défaut, 21;
(0, 0) -, -	prohibited, -, -	l'élément ou l'attribut ne doit pas apparaître

Tableau 7 - Description exhaustive des cas selon la valeur des attributs *minOccurs* et *maxOccurs* , *fixed* et *default.*

4.2.6. Contenu d'un élément

4.2.6.1. Contenu vide

Supposons que nous voulions qu'un élément ait une unité sous la forme d'un attribut au lieu que cela soit sous la forme d'un contenu :

Un tel élément n'a pas de contenu du tout ; son modèle de contenu est vide. Pour définir un type dont le contenu est vide, le principe est de définir un type ne permettant d'avoir que des sous-éléments comme contenu, sous-éléments qui ne seront jamais déclarés.

```
<xsd:element name="nom de l'élément">
    <xsd:complexType>
        <xsd:complexContent>
            <xsd:restriction base="xsd:anyType">
                <xsd:attribute name="nom de l'attribut" type="type de l'attribut"/>
            </xsd:restriction>
        </xsd:complexContent>
    </xsd:complexType>
</xsd:element>
```

4.2.6.2. Contenu simple

On utilise l'élément *xsd :simpleContent* pour indiquer qu'un élément pourrais contenir seulement un contenu simple.

```
<xsd:element name="nom de l'élément">
    <xsd:complexType>
        <xsd:simpleContent>
            <xsd:extension base="xsd:string">
                <xsd:attribute name="nom de l'attribut" type="type de l'attribut"/>
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>
</xsd:element>
```

Le type de base pour l'extension de notre exemple est *xsd :string* ; mais les types simples ne sont pas limités prédéfinis, l'élément *xsd :simpleContent* peut définir de nouveaux types de données simples, qui peuvent être référencés par des déclarations d'éléments et d'attributs dans le schéma.

4.2.6.2. Contenu complexe

On utilise l'élément *xsd :complexContent* pour indiquer qu'un élément pourrais contenir seulement un contenu complexe. La raison la plus fréquente d'utiliser cet élément est de dériver un type complexe d'un type existant.

```
<xsd:element name="nom de l'élément">
    <xsd:complexType>
        <xsd:complexContent>
            <xsd:restriction base="xsd:anyType">
                <xsd:attribute name="nom de l'attribut" type="type de l'attribut"/>
            </xsd:restriction>
        </xsd:complexContent>
    </xsd:complexType>
</xsd:element>
```

4.2.6.4. Contenu mixte

XML Schema fournit la possibilité de construire des schémas dans lesquels les caractères de données peuvent apparaître au même niveau que des sous-éléments, n'étant pas ainsi confinés au niveau des feuilles de l'arborescence ; nous parlons alors de contenu mixte.
La déclaration d'un élément complexe ayant un contenu mixte est la même que la déclaration d'un élément complexe à laquelle nous devons ajouter l'attribut *mixed* positionné à *true*

```
<xsd:complexType mixed="true">
      <!..déclaration des élément fils ..>
</xsd complexType>
```

4.2.6.5. Contenu libre (tout Type)

Le mot *anyType* représente une abstraction appelée *ur-Type* qui est le type de base à partir duquel tous les types simples et complexes sont dérivés. Un type *anyType* ne contraint un contenu d'aucune manière. Il est possible d'utiliser *anyType* comme n'importe quel autre type, par exemple :

```
<xsd:element name="nom de l'élément" type="xsd:anyType"/>
```

Le contenu de l'élément déclaré de cette manière n'est pas contraint, la valeur de l'élément peut donc être n'importe quelle autre séquence de caractères, ou un mélange de caractères et de sous-éléments. En fait, anyType est le type par défaut quand aucun type n'est spécifié.

Un contenu d'élément libre est nécessaire dans le cas où il faudra embarquer du balisage étranger dans un élément, alors la déclaration par défaut ou une forme légèrement restreinte peut être pratique.

4.7.3. Groupe d'éléments :

Il est possible de définir des groupes d'éléments afin de contrôler la manière dont chaque élément d'un groupe se comporte dans les données XML. On utilise les groupes d'éléments pour imposer les contraintes d'apparition suivantes à un ensemble d'éléments :

Utilisez	Contrainte d'apparition
Groupe **sequence**	Tous les éléments apparaissent dans l'ordre exact.
Groupe **choice**	Un seul élément du groupe apparaît.
Groupe **all**	Tous les éléments apparaissent ou bien aucun.

Tableau 8 – Contraintes d'apparition pour les groupes d'élément

La syntaxe associée est :

```
<xsd:group name="nom du groupe">
    <xsd:sequence>
        <!… déclaration des éléments fils…>
    </xsd:sequence>
</xsd:group>
```

Les groupes peuvent être nommés ou sans nom, comme ils peuvent être référencés une seule fois sans avoir à les déclarer une deuxième fois à l'aide de la syntaxe suivante :

```
<xsd:group ref="nom du groupe">
```

5. Espaces de noms

Alors que les DTD ne tiennent pas compte des espaces de noms, XML schema les utilisent fréquemment. Il existe deux espaces de noms dédiés aux schémas : **[PTPB 03]**

- http://www.w3.org/2001/XMLSchema. Espace de nom utilisé pour les éléments XML schema du W3C. Cet espace peut être utilisé comme espace de nom par défaut ou avec un préfixe xsd.
- http://www.w3.org/2001/XMLSchema-instance. Espace de noms utilisé pour les extensions XML schema du W3C employées dans le document instance. Cet espace de nom doit être utilisé avec le préfixe xsi.

En plus des deux espaces de noms dédiés aux schémas, on a la possibilité de déclarer des espaces de noms appelés cible pour vu qu'ils correspondent aux éléments et attributs déclarés.

5.1. Espace de nom cible

L'espace de nom cible nous permet de différencier les définitions et déclarations de différents vocabulaires. La valeur de l'attribut *targetNamespace* doit être l'espace de nom associé au schéma. **[REMC 02]**

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
xmlns= http://site.voila.fr/xmlschema/ressources/contacts
targetNamespace="http://site.voila.fr/xmlschema/ressources/contacts">
        <!-déclarations et définitions -->
 </xsd:schema>
```

Dans l'exemple précédent, l'espace de nom associé au schéma des contacts est l'identificateur de ressource uniforme (URI) 'http://site.voila.fr/xmlschema/ressources/contacts '.

Par conséquent, tout élément de ce schéma précédé du qualificatif associé à l'espace de nom cible est interprété par le processeur XML comme un élément de cet espace de nom.

Le nom de l'élément ne peut alors en aucun cas entrer en conflit avec le nom d'un élément issu d'un autre espace de nom s'il est adéquatement qualifié. Nous allons justement nous intéresser maintenant à la qualification d'éléments.

5.2. Qualification explicite et implicite

Nous avons associé, grâce à la valeur de l'attribut xmlns, l'espace de nom de XML Schema avec le préfixe xsd. C'est ce qu'on appelle une qualification explicite. Pour être valide, tout élément appartenant au vocabulaire de XML Schema doit être précédé du préfixe xsd: (xsd:element par exemple). Signalons que nous aurions très bien pu choisir n'importe quel autre préfixe. Néanmoins, il est d'usage d'associer le préfixe xsd: à l'espace de nom de XML Schema.

Il existe également un procédé de qualification dit implicite qui utilise l'espace de nom par défaut. Tout document XML peut avoir un espace de nom par défaut ; c'est à dire que les éléments non préfixés du document appartiennent à l'espace de nom par défaut.

5.3. Déclarations locales et globales

On dit qu'un élément (ou un attribut) est déclaré globalement lorsqu'il est fils de l'élément schema. Par conséquent la déclaration globale d'un élément lui permet d'être au plus haut niveau dans une instance d'un document c'est à dire l'élément racine. **[SILA 00]**
Les éléments ou attributs déclarés à l'intérieur d'un type complexe sont dits locaux.
Nous pouvons aussi référencer un élément défini *globalement* à l'intérieur de la déclaration d'un type complexe, c'est à dire qu'il sera référencer à la suite d'un élément déclaré *localement* avec la précaution d'utiliser le mot clé *ref.*
La principale contrainte pour une déclaration globale dans un schéma XML est que le nom de l'élément (ou de l'attribut) doit être unique. Il y a également une différence entre les éléments (attributs) déclarés localement et globalement au niveau de la qualification dans les documents instances.

5.4. Qualification dans les documents instances

Dans le document source, les noms d'éléments et attributs sont qualifiés par le préfixe déclaré (tout préfixe, sauf *xml* et *xmlns*, doit être déclaré comme étant un attribut de déclaration d'espace de noms). Notons que contrairement au schéma lui même, le document instance a des attributs qualifiés car aucune balise ne peut contenir deux attributs non qualifiés qui ont des nom identiques ou qui ont les mêmes noms , ils sont qualifiés avec des préfixes associés au même espace de noms.
La déclaration de l'espace de noms est appliquée à l'élément où elle est spécifiée et à son contenu sauf si un autre espace de noms y est déclaré.
Exemple :

```
<pref :père xmlns :pref=URI elementFormDefault=qualified>
        <perf : fils 1/>
                .
                .
        <perf : fils n/>
</perf :père>
```

5.4.1. Eléments et attributs globaux

Tous les éléments et les attributs déclarés globalement dans un schéma XML doivent être qualifiés, de façon explicite ou implicite.
La qualification, dans les documents instances, des éléments et attributs locaux peut être contrôler par l'auteur du schéma XML.

5.4.2. Eléments et attributs locaux

Un auteur de schéma XML peut décider si les éléments et les attributs déclarés localement doivent être qualifiés par un espace de nom en utilisant une qualification explicite ou implicite. Il dispose pour cela de deux méthodes, l'une spécifiant globalement la qualification des attributs et éléments locaux, et l'autre spécifiant de façon ponctuelle la qualification d'un attribut ou d'un élément particulier.

La première méthode consiste à spécifier globalement si la qualification pour les éléments et attributs locaux est requise dans les documents instances. Il existe deux attributs, *elementFormDefault* et *attributeFormDefault*, portés par l'élément *schema*. Les deux attributs ont pour valeur par défaut '*unqualified*', ce qui signifie que la qualification n'est pas requise.

La seconde méthode consiste à spécifier pour certains éléments (ou attributs) locaux si la qualification est requise, à l'aide de l'attribut *form*. Cet attribut peut être porté par l'élément *element* et l'élément *attribute* de XML Schema. IL faut noter que la valeur de cet attribut (qualified ou unqualified) surcharge la valeur de elementFormDefault ou de attributeFormDefault selon qu'il est porté par l'élément element ou par l'élément attribute respectivement.

5.5. Schémas sans espace de nom cible

Les déclarations d'éléments dans un schéma sans espace de nom cible valident uniquement les éléments non qualifiés du document instance. C'est à dire qu'elles valident les éléments pour lesquels il n'y a aucune qualification, ni explicite, ni implicite. Donc, pour valider un document XML 1.0 traditionnel n'utilisant pas d'espaces de nom, il faut utiliser un schéma sans espace de nom cible.**[SILA 00]**

6. Dérivation de types

Les types simples et complexes permettent déjà de faire plus de choses que les déclarations dans le langage DTD. Il est possible de raffiner leur déclaration de telle manière qu'ils soient une "restriction" ou une extension d'un type déjà existant, en vue de préciser un peu plus leur forme.
De même que certains langages de programmation orientés objet permettant au concepteur d'un objet d'imposer des limites sur la façon dont un objet peut être étendu, le langage de schéma permet aux rédacteurs de schémas de positionner des restrictions sur l'extension et la restriction de type.
C'est ainsi que XML Schema propose deux techniques de dérivations de types : la première appelée dérivation par restriction du type ancêtre, et la seconde appelée dérivation par extension du type ancêtre. Notons que ces deux mécanismes de dérivation s'appliquent aussi bien aux types de données simples que complexes, ce qui les rend très puissants.

6.1. Restriction

6.1.1. Restriction de types simples (facettes)

Le mécanisme de dérivation par restriction permet de créer de nouveaux types simples (ou atomiques) à partir de la bibliothèque de types simples intégrée à XML Schema. On introduit à cette fin la notion de 'facettes', qui sont en réalité des contraintes applicables sur un type simple particulier.
Une 'facette' permet de restreindre l'ensemble des valeurs légales d'un type de base.[ERWS 02]

Exemple : Supposons que nous souhaitions créer un type simple *monEntier* dont les valeurs seraient comprises entre 0et 99 inclus. Il nous suffit de dériver le type simple intégré *nonNegativeInteger* et de lui appliquer la 'facette' *maxEclusive*.

```
<xsd:simpleType name="monEntier">
    <xsd:restriction base="xsd:nonNegativeInteger">
        <xsd:maxExclusive value="100"/>
    </xsd:restriction>
</xsd:simpleType>
```

Il existe un nombre important de 'facettes' qui permettent de :

- Fixer la longueur d'un type simple, restreindre sa longueur maximale et minimale
  Ex : la facette *length.*
- Enumérer toutes les valeurs possibles d'un type
  Ex : la facette *enumeration*
- Gérer des expressions régulières
  Ex : la facette *patern*
- Fixer la valeur minimale ou maximale d'un type
  Ex : la facette *maxExclusive*
- Contrôler la façon dont le processeur de schéma gérera les espaces blancs
  Ex : la facette *whiteSpace*

6.1.2. Restriction de types complexes

Il est également possible de l'appliquer aux types complexes. Un type complexe dérivé par restriction est semblable à son type de base, excepté le fait que ses déclarations dans son modèle de contenu sont plus limitées que celles du type de base. En réalité, les valeurs représentées par le nouveau type sont un sous-ensemble des valeurs du type de base, ce qui également le cas pour les types simples.

Exemple :
Supposons que nous désirions restreindre un type complexe, pour cela nous pouvons faire appel à l'attribut *maxOccurs* d'où la syntaxe suivante :

```
<xsd:complexType name="nom du nouveau type comlexe">
    <xsd:complexContent>
        <xsd:restriction base="type complexe de base">
            <xsd:sequence>
                <xsd:element name="nom1"maxOccurs="val1" type="type1"/>
            </xsd:sequence>
        </xsd:restriction>
    </xsd:complexContent>
</xsd:complexType>
```

Signalons que les types dérivés par restriction doivent reprendre tous les composants déclarés dans le type de base.

6.2. Extension

6.2.1. Extension de types simples

Si nous avons un élément de type simple et nous voulions lui rajouter un attribut, le type de cet élément deviendrait complexe.
Pour définir un type complexe basé sur un type simple, on applique la dérivation par extension de types simples. Elle permet de créer des types complexes dont le contenu est un type simple et qui peuvent porter des attributs. **[ERWS 02]**

La syntaxe associée est :

```
<xsd:complexType name="nom du nouveau type complexe">
    <xsd:simpleContent>
        <xsd:extension base="le type simple prédéfini">
            <xsd:attribute name="nom attribut 1" type="type attribut 1"/>
                .
                .
            <xsd:attribute name="nom attribut n" type="type attribut n"/>
        </xsd:extension>
    </xsd:simpleContent>
</xsd:complexType>
```

L'élément *simpleContent* du langage XML Schema indique que notre nouveau type contient uniquement un type simple et pas de sous-éléments. L'élément *extension* porte l'attribut *base* qui indique le type simple de base ; *l'attribut nom attribut 1.....nom attribut n* étant un élément fils de l'élément *extension*.

6.2.2. Extension de types complexes

Le mécanisme de dérivation par extension de XML Schema est plus proche conceptuellement du mécanisme d'héritage des langages orientés objet. Alors que la dérivation par restriction de type complexe réduit la valeur du type à sous-ensemble du type de base, la dérivation par extension rend le modèle de contenu du type de base plus riche. Elle permet par exemple de rajouter des éléments ou des attributs au modèle complexe de base.
Cette extension se fait comme suit :

```
<xsd:complexType name="nom du nouveau type complexe">
   <xsd:complexCentent>
        <xsd:extension base=" nom du nouveau type complexe ">
         <xsd:sequence>
              <!…les déclarations d'éléments rajoutés….>
           </xsd:sequence>
         <!…nous pouvons eventuellement  rajouter des attributs….>
        </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
```

En prenant soin de préciser que le modèle de contenu de ces types est complexe avec la balise *complexContent*. Lorsqu'un type complexe est dérivé par extension, son modèle de contenu effectif est celui du type de base concaténé au modèle de contenu spécifié dans la dérivation. En effet, les deux modèles de contenu sont considérés par XML Schema comme deux fils d'un groupe séquentiel. **[REMC 02]**

7. Conception avancée de schémas

7.1. Types anonymes

Jusqu'à présent, nous avons utilisé dans notre schéma des types déclarés globalement qui portaient tous un nom. Nous utilisions l'attribut type pour référencer le type d'un élément, comme personne par exemple. Ce style de schémas est très simple à appréhender mais n'est pas forcément optimal, surtout quand il s'agit d'un grand nombre de types référencés une seule fois et possédant très peu de contraintes. A l'instar du langage Java qui permet de se servir de classes internes anonymes, XML Schema nous autorise à utiliser des types de données anonymes déclarés localement.

Il est à noter également que les types anonymes peuvent aussi bien être utilisés pour des types simples que complexes.

7.2. Réutilisation de schémas existants

7.2.1. Inclusion

Lorsqu'un schéma dépasse une taille « critique », en général une page, il devient plus difficile à maintenir. Il est alors opportun de scinder ce schéma en plusieurs documents.

Pour inclure un schéma *fichir1.xsd* à l'espace de nom de *fichier2.xsd,* nous utilisons l'instruction :

<xsd:includeschemaLocation="URI de *fichier1.xsd*"/>

dans le *fichier2*.
L'espace de nom cible des deux schémas doit être le même, et l'attribut *schemaLocation* est obligatoirement porté par l'élément *include*.

La seule condition pour utiliser l'élément xsd:include est que l'espace de nom cible du schéma inclus doit être le même que celui du schéma englobant.

7.2.2. Redéfinition

Le mécanisme d'inclusion nous permet d'utiliser des composants de schémas externes tels quels, c'est à dire sans aucune modification. Le mécanisme de redéfinition nous permet, lui, de redéfinir des types simples ou complexes, ainsi que des groupes et groupes d'attributs issus de schémas externes.
Pour redéfinir un type, nous le dérivons par extension en donnant le même nom au nouveau type comp exe, et cela doit être absolument à l'intérieur de l'élément *redefine* déclaré comme suit :

<xsd:redefine schemaLocation="chemin de l'élément à redéfner">

Tout comme l'inclusion, la redéfinition nécessite que les composants externes appartiennent au même espace de nom cible que le schéma qui les redéfinit. Les composants externes n'ayant pas d'espace de nom peuvent également être redéfinis, dans ce cas ils deviennent partie intégrante de l'espace de nom cible du schéma les redéfinissant.

7.2.3. Importation de types

Pour importer un type, il faut utiliser l'élément *import.* Cet élément a un attribut nommé *namespace* dont la valeur doit être l'espace de nom cible auquel appartient les composants que l'on désire importer. Il faut également associer un préfixe à cet espace de nom, qui sera utilisé pour référencer le type en question.

Le mécanisme d'importation est sans doute, avec la dérivation, l'un des mécanismes les plus puissants de XML Schema. Il permet de concevoir des schémas de façon modulaire.

XML Schema permet en fait d'importer plusieurs composants de schémas de différents espaces de nom ; ces composants pouvant être référencés aussi bien dans des déclarations que des définitions. Signalons également que seuls les éléments, attributs et types déclarés globalement peuvent être importés, et que les types complexes importés peuvent être utilisés comme types de base pour la dérivation de nouveaux types.

Lorsque des composants de schémas sont importés de plusieurs espaces de nom, chaque espace de nom doit être identifié avec un élément *import* séparé. Les éléments *import* doivent

apparaître comme les premiers fils de l'élément *schema*. De plus, chaque espace de nom doit être associer à un préfixe en utilisant une déclaration standard d'espace de nom. Ce préfixe sera utilisé pour qualifier les références à n'importe quel composant appartenant à cet espace de nom. Pour finir, l'attribut *schemaLocation* peut être porté optionnellement par l'élément *import* pour indiquer l'emplacement du schéma à importer au processeur XML.

8. Autres caractéristiques intéressantes de XML Schema

8.1. Annotations

XML Schema fournit trois éléments pour documenter un schéma. Le premier élément, *annotation*, est un élément englobant qui peut contenir soit des informations à l'intention d'un lecteur humain signalées par le sous-élément *documentation*, soit des informations destinées à une application contenue dans le sous-élément *appInfo*.

Nous pouvons par exemple fournir des informations aux lecteurs de notre schéma de la façon suivante :

```
<xsd:annotation>
    <xsd:documentation>
        informations qu'on veut apportées
    </xsd:documentation>
</xsd:annotation>
```

8.2. Documentation

XML Schema permet, outre l'utilisation des commentaires comme tout format XML, l'adjonction de documentation aux éléments. La documentation à l'intention des lecteurs humains peut être définie dans des éléments *xsd:documentation*, tandis que les informations à l'intention de programmes doivent être incluses dans des éléments *xsd:appinfo*. Ces deux éléments doivent être placés dans un élément *xsd:annotation*. Ils disposent d'attributs optionnels: *xml:lang* et *source*, qui est une référence à une URI pouvant être utilisée pour identifier l'objectif du commentaire ou de l'information.

8.3. Valeur "null"

Il est toujours préférable de pouvoir indiquer explicitement qu'un élément est non renseigné plutôt que d'omettre cet élément. La valeur *null* des bases de données relationnelles est utilisée dans ce but. XML Schema intègre un mécanisme similaire permettant d'indiquer qu'un élément peut être nul.

L'attribut *null* fait partie de l'espace de nom des instances XML Schema et il est par convention explicitement qualifié à l'aide du préfixe *xsd*. Signalons qu'un élément portant l'attribut *xsd:null* égal à *true* ne peut contenir aucun élément mais qu'il peut néanmoins porter des attributs. **[ERWS 02]**

8.4. Contraintes d'unicité et contraintes référentielles

Les DTD fournissent déjà un mécanisme assurant l'unicité, au moyen de l'attribut ID et de ses attributs associés *IDREF* et *IDREFS*. Bien que ce mécanisme ait été repris pour des raisons de compatibilité ascendante, XML Schema introduit de nouveaux mécanismes plus souples et plus puissants.

L'élément *xsd :key* est fortement associé à l'élément *xsd :unique,* ils emploient l'élément *xsd :selector* pour définir un ensemble des éléments auxquels ils s'appliquent, puis un ou plusieurs éléments *xsd :field* sont utilisés pour définir quelles valeurs composent cette clé particulière. La principale différence entre les deux éléments cités précédemment est que l'objectif de l'élément *xsd :key* est de définir un ensemble d'éléments qui peuvent être référencés en utilisant l'élément *xsd :keyref.*

De plus, grâce à *XPath*, XML Schema permet de spécifier la protée de la contrainte d'unicité tandis que la portée d'un attribut ID est fixée s'applique à tout le document. Enfin, il est possible de créer des clés et des références à partir de combinaisons d'éléments ou d'attributs, ce qui est impossible avec les DTD. **[REMC 02]**

Pour conclure ce chapitre nous allons présenter tous les composants chargés de représenter l'arborescence et les informations d'un document XML.**[ERWS 02]**

9. Les composants de XML Schema

Ils sont divisés en plusieurs catégories :

9.1. Un élément racine :

xsd :schéma.

Un schéma XML commence par l'ouverture d'un élément *schema* se comportant comme un élément racine destiné à accueillir la définition des composants d'un document XML. L'élément *schema* possède plusieurs attributs destinés à définir le cadre du schéma XML.

Attributs	Description
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.

L'élément *schema* ne peut être inclus dans aucun élément.
Il possède les éléments : *include, import, annotation, redefine, attribute, attributeGroup, element, group, notation, simpleType, complexType* enfants.

9.2. Trois éléments de déclaration :

xsd :attribute

L'élément *attribute* correspond à un type d'attribut déclaré qui peut apparaître à l'intérieur de la portée d'un élément défini par *elementType*.
L'élément *attribute* possède deux attributs définissant le contenu complexe.

Attributs	Description
default	définit une valeur par défaut pour l'élément *attribute*.
type	indique le nom de l'élément *attributeType* défini dans le schéma en cours.
required	détermine si l'élément *attribute* est requis dans l'élément.

L'élément *attribute* posséde forcément un élément parent parmi : *attributeGroup, schema, complexType, restriction (simpleContent), extension (simpleContent), restriction (complexContent), extension (complexContent),* et *annotation, simpleType* comme enfants.

xsd :element

L'élément *element* permet de représenter un élément XML dans une définition de schéma.
L'élément *element* possède plusieurs attributs destinés à définir précisément l'élément XML.

Attributs	Description
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*.
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.

L'élément *element* ne peut être inclus que dans les éléments : *schema, choice, all, sequence* et avoir comme fils les éléments : *simpleType, complexType, key, keyref, unique.*

xsd :notation

L'élément *notation* définit une notation. Chaque notation possède un nom, un ID publique et un ID système identifié par l'attribut pertinent de cet élément.
L'élément *notation* a quatre attributs destinés à l'identifier.

Attributs	Description
Id	précise un identificateur pour cet élément.
Name	Représente le Nom de l'élément notation
Public	Référence URI correspondant à l'identificateur public.
System	Référence URI correspondant à l'identificateur system.

L'élément *notation* a comme parent l'élément *schema* et il ne peut contenir que l'élément *annotation.*

9.3. huit éléments pour définir des types :

xsd :complexContent

L'élément *complexContent* permet de définir un contenu complexe pour un élément XML. L'élément *complexContent* possède deux attributs définissant le contenu complexe.

Attributs	Description
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.

L'élément *complexContent* ne peut être inclus que dans l'élément *complexType* et avoir comme fils les éléments : *annotation ,restriction , extension .*

xsd :complexType

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

Attributs	Description
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.

L'élément *complexType* ne peut être inclus que dans les éléments : *element, redefine et schema et avoir comme fils les éléments : annotation, simpleContent, complexContent, group, all, choice, sequence, attribute, attributeGroup, anyAttribute.*

xsd :simpleContent

L'élément *simpleContent* permet de créer un type de données complexe à partir d'un type de données simple.
L'élément *simpleContent a* un seul attribut destiné à l'identifier.

Attributs	Description
Id	précise un identificateur unique pour l'élément.

L'élément *simpleContent* ne peut être inclus que dans l'élément *complexType* et avoir comme fils les éléments : *annotation ,restriction , extension .*

xsd :simpleType

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

Attributs	Description
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.

L'élément *simpleType* ne peut être inclus que dans les éléments : *attribute, element, list, redefine, restriction, schema, union* et avoir comme fils les éléments *: annotation, list, restriction, union.*

xsd :extention

L'élément *extension* propose d'étendre la définition d'un élément ou d'un attribut XML à un autre type de données spécifié. L'élément *extension* possède deux attributs définissant la structure extensive.

Attributs	Description
base	indique un type de données de base.
id	précise un identificateur unique pour l'élément.

L'élément *extension* ne peut être inclus que dans l'élément *complexContent* ou *simpleContent* et avoir co nme fils les éléments : *annotation, attribute, attributeGroup, anyAttribute, choice, all, sequence, group.*

xsd :restriction

L'élément *restriction* permet de restreindre les données permises dans un élément ou un attribut XML. L'élément *restriction* possède deux attributs définissant la structure restrictive.

Attributs	Description
base	indique un type de données de base.
id	précise un identificateur unique pour l'élément.

L'élément *restriction* ne peut être inclus que dans l'élément *simpleType* ou *simpleContent* ou *complexContent* et avoir comme fils les éléments : *annotation, fractionDigits, enumeration, length, maxExclusive, maxInclusive, maxLength, minExclusive, minInclusive, minLength, pattern, s mpleType, totalDigits, whiteSpace, , whiteSpace, attribute, attributeGroup, anyAttribute group, all, choice, sequence.*

xsd :list

L'élément *list* permet de créer de nouveaux types de listes par dérivation de types de données atomiques existants. L'élément *list* possède plusieurs attributs destinés à l'identifier et à préciser un type de données.

Attributs	Description
id	précise un identificateur unique pour l'élément.
itemType	spécifie le nom d'un type de données existants.

L'élément *list* ne peut être inclus que dans l'élément *simpleType* et avoir comme enfants les éléments : *annotation, simpleType*

xsd :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. L'élément *union* possède plusieurs attributs destinés à l'identifier et à préciser des types de données.

Attributs	Description
id	précise un identificateur unique pour l'élément.
memberTypes	spécifie une liste de noms de types de données séparés par un espace blanc.

L'élément *union* ne peut être inclus que dans l'élément *simpleType* et avoir comme enfants les éléments : *annotation, simpleType*

9.4. Sept éléments pour définir les modèles de contenu :

xsd :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. L'élément *all* possède plusieurs attributs destinés à définir ce connecteur.

Attributs	Description
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.

L'élément *all* ne peut être inclus que dans les éléments *group, restriction, extension, , complexType* et avoir comme enfants les éléments : *annotation, element*

xsd :any

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

Attributs	Description
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.

L'élément *any* ne peut être inclus que dans les éléments *choice* et *sequence* et avoir comme enfants l'élément *annotation*

xsd: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.

Attributs	Description
id	précise un identificateur unique pour l'élément.
namespace	spécifie un ou plusieurs espaces de noms.
processContents	précise le type de processus de contenu.

L'élément *anyAttribute* ne peut être inclus que dans les éléments : *complexType, restriction , extension, , attributeGroup* et avoir comme enfants l'élément *annotation.*

xsd: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.

Attributs	Description
id	Précise un identificateur unique pour l'élément.
name	indique le nom du groupe.
ref	indique une référence à un groupe d'attributs.

L'élément *attributeGroup* ne peut être inclus que dans les éléments suivants : *attributeGroup, complexType, schema, restriction, extension* et avoir comme fils les éléments : *annotation, attribute, attributeGroup, anyAttribute*

xsd :choice

L'élément *choice* propose une structure de choix entre plusieurs éléments possibles. L'élément *choice* possède plusieurs attributs destinés à définir le connecteur de choix.

Attributs	Description
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, il est égal à 1.

L'élément *choice* ne peut être inclus que dans les éléments : *group, choice, sequence, complexType, restriction , extension* et avoir comme fils les éléments : *annotation, any, choice, element, group, sequence.*

xsd :group

L'élément *group* organise le contenu à l'intérieur d'un groupe pour spécifier une séquence. L'élément *group* possède plusieurs attributs destinés à définir le connecteur de choix.

Attributs	Description
minOccurs	détermine si l'élément est requis au mois une fois (*1*) ou ne l'est pas (*0*).
maxOccurs	détermine si l'élément doit apparaître au maximum une fois (*1*) ou un
Order	permet une seule instance (*one*) de chaque élément contenu dans un groupe, ou plusieurs éléments apparaissant dans une séquence spécifiée (*seq*) ou un ordre quelconque (*many*).

L'élément *group* possède les éléments : *schema, choice, sequence, complexType, restriction), extension* comme parents *ElementType* et les éléments : *annotation, all, choice, sequence* comme fils.

xsd :sequence

L'élément *sequence* définit, dans un type de données complexe, un à plusieurs éléments devant obligatoirement apparaître dans un ordre prédéfini. L'élément *sequence* possède plusieurs attributs destinés à définir le connecteur de séquence.

Attributs	Description
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.

L'élément *sequence* ne peut être inclus que dans les éléments : *group, choice, sequence, complexType, restriction , extension* et avoir comme fils les éléments : *annotation, any, choice, element, group, sequence*

9.5. Cinq éléments pour spécifier des contraintes d'identité :

xsd :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. L'élément *field* possède plusieurs attributs destinés à identifier le champ et à préciser une expression XPath.

Attributs	Description
id	précise un identificateur unique pour l'élément.
xpath	spécifie un sous-ensemble d'expressions XPath.

L'élément *field* ne peut être inclus que dans les éléments : *key, keyref, unique* et avoir comme fils l'élément *annotation.*

xsd :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.

Attributs	Description
id	précise un identificateur unique pour l'élément.
name	spécifie un nom pour l'élément.

L'élément *key* ne peut être inclus que dans l'élément *element* et avoir comme fils les éléments : *annotation, field, selector*

xsd :keyref

L'élément *keyref* permet de créer une référence à une clé existante dans le schéma XML. L'élément *keyref* possède plusieurs attributs destinés à définir la clé référencée.

Attributs	Description
id	précise un identificateur unique pour l'élément.
name	spécifie un nom pour l'élément.
refer	se réfère à un élément *key* existant.

L'élément *keyref* ne peut être inclus que dans l'élément *element* et avoir comme fils les éléments : *annotation, field, selector*

xsd :selector

L'élément *selector* permet de définir une expression XPath chargée de sélectionner un élément XML afin de lui appliquer une clé. L'élément *selector* possède plusieurs attributs destinés à identifier le champ et à préciser une expression XPath.

Attributs	Description
id	précise un identificateur unique pour l'élément.
xpath	spécifie un sous-ensemble d'expressions XPath.

L'élément *selector* ne peut être inclus que dans les éléments *key, keyref, unique* et avoir l'élément *annotation* comme fils.

xsd :unique

L'élément *unique* permet de définir une contrainte d'unicité sur un noeud d'une arborescence XML. L'élément *unique* possède plusieurs attributs destinés à l'identifier et à préciser son nom.

Attributs	Description
id	précise un identificateur unique pour l'élément.
name	spécifie un nom pour l'élément.

L'élément *unique* ne peut être inclus que dans l'élément *element* et avoir comme fils les éléments : *annotation, field, selector.*

9.6. Trois éléments pour composer des schémas à partir de sources externes :

xsd :import

L'élément *import* permet d'importer un schéma XML avec un espace de noms différent dans un autre schéma. L'élément *import* possède plusieurs attributs destinés à l'identifier et à préciser les adresses du schéma à importer et des l'espaces de noms.

Attributs	Description
id	précise un identificateur unique pour l'élément.
names	spécifie l'espace de noms du schéma XML.
schemaLocation	spécifie une adresse URI pointant vers un schéma XML.

L'élément *import* ne peut être inclus que dans l'élément *schema* et avoir l'élément *annotation* comme fils.

xsd :include

L'élément *include* permet d'inclure un schéma XML d'un même espace de noms dans un autre schéma. L'élément *include* possède plusieurs attributs destinés à l'identifier et à préciser l'adresse du schéma à inclure.

Attributs	Description
id	précise un identificateur unique pour l'élément.
schemaLocation	spécifie une adresse URI pointant vers un schéma XML.

L'élément *include* ne peut être inclus que dans l'élément *schema* et avoir l'élément *annotation* comme fils.

xsd :redefine

L'élément *redefine* permet d'importer et de redéfinir les déclarations d'un schéma XML pour un même espace de noms.
L'élément *redefine* possède plusieurs attributs destinés à l'identifier et à préciser le schéma à importer.

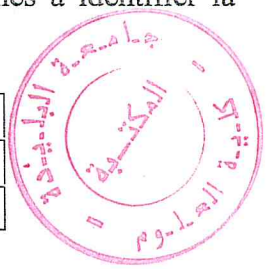
Attributs	Description
id	précise un identificateur unique pour l'élément.
schemaLocation	spécifie une adresse URI pointant vers un schéma XML à rédéfinir.

L'élément *redefine* ne peut être inclus que dans l'élément *schema* et avoir les éléments *annotation, attributeGroup, complexType, group, simpleType* comme fils.

9.7. 12 facettes pour contraindre des types simples : (Ne peuvent être incluses que dans l'élément restriction)

xsd :enumeration

L'élément *enumeration* permet de contraindre la valeur d'un élément ou d'un attribut à une seule possibilité. L'élément *enumeration* possède plusieurs attributs destinés à identifier la facette et à préciser une valeur possible.



Attributs	Description
id	précise un identificateur unique pour l'élément.
value	spécifie une valeur de type simple (simpleType).

xsd :fractionDigits

L'élément *fractionDigits* permet de définir le nombre total de chiffres dans la partie fractionnaire d'un nombre à virgule flottante. L'élément *fractionDigits* possède plusieurs attributs destinés à l'identifier, à préciser une valeur et le comportement de cette dernière.

Attributs	Description
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.

xsd :length

L'élément *length* permet de définir une longueur pour l'élément ou l'attribut XML. L'élément *length* possède plusieurs attributs destinés à l'identifier, à préciser une valeur et le comportement de cette dernière.

Attributs	Description
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.

xsd :maxExclusive

L'élément *maxExclusive* permet de définir une valeur maximum pour l'élément ou l'attribut XML. L'élément *maxExclusive* possède plusieurs attributs destinés à l'identifier, à préciser une valeur et le comportement de cette dernière.

Attributs	Description
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).

xsd :maxInclusive

L'élément *maxInclusive* permet de définir une valeur maximum inclusive pour l'élément ou l'attribut XML. L'élément *maxInclusive* possède plusieurs attributs destinés à l'identifier, à préciser une valeur et le comportement de cette dernière.

Attributs	Description
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).

xsd :maxLenght

L'élément *maxlength* permet de définir une longueur maximum pour l'élément ou l'attribut XML. L'élément *maxlength* possède plusieurs attributs destinés à l'identifier, à préciser une valeur et le comportement de cette dernière.

Attributs	Description
fixed	permet de fixer la valeur de l'élément.
id	précise un identificateur unique pour l'élément.
value	spécifie une longueur maximum.

xsd :minExclusive

L'élément *minExclusive* permet de définir une valeur maximum exclusive pour l'élément ou l'attribut XML. L'élément *minExclusive* possède plusieurs attributs destinés à l'identifier, à préciser une valeur et le comportement de cette dernière.

Attributs	Description
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).

xsd :minInclusive

L'élément *minInclusive* permet de définir une valeur minimum inclusive pour l'élément ou l'attribut XML. L'élément *minInclusive* possède plusieurs attributs destinés à l'identifier, à préciser une valeur et le comportement de cette dernière.

Attributs	Description
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).

xsd :minLenght

L'élément *minLength* permet de définir une longueur minimum pour l'élément ou l'attribut XML. L'élément *minLength* possède plusieurs attributs destinés à l'identifier, à préciser une valeur et le comportement de cette dernière.

Attributs	Description
fixed	permet de fixer la valeur de l'élément.
id	précise un identificateur unique pour l'élément.
value	spécifie une longueur maximum.

xsd :pattern

L'élément *pattern* permet de créer un modèle pour la valeur d'un élément ou d'un attribut XML à partir d'une expression régulière. L'élément *pattern* possède plusieurs attributs destinés à identifier la facette et à préciser une expression régulière.

Attributs	Description
id	précise un identificateur unique pour l'élément.
value	spécifie une valeur de type simple (simpleType).

xsd :totalDigits

L'élément *totalDigits* permet de définir le nombre total de chiffres dans un élément ou un attribut XML. L'élément *totalDigits* possède plusieurs attributs destinés à l'identifier, à préciser une valeur et le comportement de cette dernière.

Attributs	Description
fixed	permet de fixer la valeur de l'élément.
id	précise un identificateur unique pour l'élément.
value	spécifie un nombre total de chiffres.

xsd :whitespace

L'élément *whiteSpace* permet de définir le comportement à adopter vis-à-vis des espaces blancs dans une valeur de chaîne de caractères. L'élément *whiteSpace* possède plusieurs attributs destinés à l'identifier, à préciser une valeur et le comportement de cette dernière.

Attributs	Description
fixed	permet de fixer la valeur de l'élément.
Id	précise un identificateur unique pour l'élément.
value	Spécifie un comportement à appliquer aux espaces blancs dans une chaîne de caractères.

9.8. Trois éléments pour documenter les schémas :

xsd :annotation

L'élément *annotation* propose une structure permettant de fournir des informations applicatives ou destinées à l'utilisateur dans un schéma. L'élément *annotation* possède un seul attribut destiné à l'identifier.

Attributs	Description
id	précise un identificateur unique pour l'élément.

L'élément *annotation* peut être inclus dans tous les éléments et avoir comme fils les éléments *appinfo, documentation.*

xsd :appinfo

L'élément *appinfo* permet de spécifier des informations destinées à être utilisées par une application. L'élément *appinfo* possède un seul attribut destiné à cibler une information applicative.

Attributs	Description
source	spécifie une adresse URI pointant vers une information.

L'élément *appinfo* ne peut être inclus que dans l'élément *annotation* et peut contenir tout contenu XML correctement construit.

xsd :documentation

L'élément *documentation* spécifie une information à propos du schéma destinée à l'utilisateur. L'élément *documentation* possède deux attributs destinés à cibler une information et à définir le langage utilisé par cette dernière.

Attributs	Description
source	spécifie une adresse URI pointant vers une information.
Xml:lang	précise le langage dans lequel est écrit la documentation.

L'élément *documentation* ne peut être inclus que dans l'élément *annotation* et peut contenir tout contenu XML correctement construit.

10. Résumé :

Ce chapitre nous a permis de découvrir l'essentiel de ce qu'il faut savoir sur les Schémas XML. Nous avons vu que le schéma XML comme les DTD est un mécanisme décrivant chaque objet pouvant apparaître dans le document XML. Il est doté d'une syntaxe, d'une grammaire et d'un vocabulaire particulier.

À la différence des DTD, les schémas sont exprimés en XML ce qui permet à une grammaire d'être manipulée comme n'importe quel document XML, ils permettent de typer les données se trouvant dans le document. D'autre part, un schéma permet de préciser le nombre d'occurrences d'un élément au sein d'un document XML plus précisément qu'avec une DTD. Finalement, il offre plus de liberté quant à l'expression de l'imbrication des éléments au sein d'un document XML.

Chapitre 4 :
Les formulaires

1. Introduction

Une manière répandue d'obtenir des informations de la part des utilisateurs consiste à saisir des données à partir d'un formulaire de saisie. Les utilisateurs peuvent introduire des informations ou sélectionner des choix à l'aide des zones de texte, des boutons d'option, des listes déroulantes et des cases à cocher.

2. Notion de formulaire

Les formulaires sont une cueillette systématique de l'information qui proviendra de la création des instances ou bien de la saisie au moment ou le blanc du formulaire est complété. Tout comme les états facilitant l'impression de tables et de requêtes, les formulaires constituent un moyen pratique d'afficher, de saisir et de modifier les informations à partir d'une simple saisie.

Les formulaires permettent d'ajouter et de modifier des données, et fournissent aux utilisateurs les bases de l'interface utilisateurs. Ce sont des objets qui possèdent leurs propres propriétés, événements et méthodes :

Les propriétés d'un formulaire déterminent son aspect (tel que son emplacement, sa taille et sa couleur) ainsi que certains de ses comportements (par exemple, s'il peut être redimensionné).

Un formulaire peut également répondre à des événements initiés par l'utilisateur ou déclenchés par le système tel que le clic de souris.

Outre les propriétés et les événements, l'emploi des méthodes existe pour manipuler les formulaires. Par exemple, la méthode *Move* permet de modifier l'emplacement et la taille d'un formulaire.

Il est possible de créer des formulaires qui imitent vos formulaires papier habituels et reconstituer ainsi un environnement familier pour la saisie de données. Un formulaire peut être redimensionnable, masqué, avec une barre de titre,…

3. L'utilité des formulaires

Les formulaires sont utilisés pour offrir aux utilisateurs une interface conviviale (ergonomique) permettant de visualiser et de saisir des données. Cependant, les formulaires représentent bien plus qu'une interface. Les formulaires proposent un grand ensemble d'objets qui peuvent répondre aux évènements déclenchés par l'utilisateur ou le système (cliquer sur un bouton, remplir un champ, taper sur une touche qui déclenche un effet sonore, …). De cette façon, les utilisateurs peuvent effectuer leurs tâches de gestion de l'information aussi facilement et intuitivement que possible.

4. Contenu d'un formulaire

Un formulaire contient un ensemble d'éléments dit « forme de saisie ». Une forme de saisie est un Objet graphique, tel qu'une zone de texte, un rectangle ou un bouton de commande placé dans un formulaire pour faciliter sa lecture, afficher des données ou effectuer une opération. Les contrôles comprennent notamment les cases à cocher, les zones d'édition, les étiquettes, les traits, les images, les formes, etc. Les formes de saisies constituent l'aspect le plus important dans un formulaire. Il s'agit des composants d'interface utilisateur qui facilitent la saisie et l'affichage des données. (figure : un ensemble de contrôles).

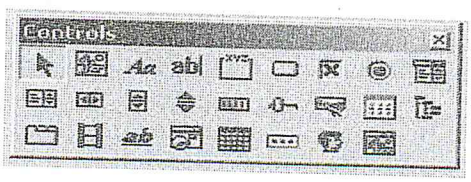


Figure 2 - palette des différentes formes de saisie

5. Description des contrôles

Un formulaire contient des formes de saisies qui permettent d'afficher et de modifier des données .On trouve six catégories de contrôles qui permettent de bénéficier d'une certaine souplesse dans la création des formulaires, on les a regroupé en deux catégories :
Catégorie 1 : Contrôle statique (caption).
Catégorie 2 : *Boutons* : permettent de communiquer avec l'application au moyen d'un clic de souris. Ils sont de trois types :

- Boutons radio.
- Case à cocher.
- Bouton de commande.

Catégorie 3 : Barre de défilement
Catégorie 4 : Zone de liste.
Catégorie 5 : Contrôle d'édition (champs de saisie).
Catégorie 6 : Zone de liste modifiable

6. Tableau récapitulatif

Le tableau ci dessous récapitule les contrôles cités précédemment :

Type	Aspect	Utilité
Contrôle statique	Controle statique	Il contient des informations statiques, telles que titres ou instructions, ou simplement des éléments d'illustrations.
Bouton radio	Bouton radio	Les boutons d'option ou radio se présentent généralement par groupes. Si l'un des boutons est coché, les autres ne le sont pas.
Bouton case à cocher	Case à cocher	Les cases à cocher, quant à elles, peuvent être sélectionnées individuellement, il est possible d'en cocher plusieurs à la fois.
Bouton de commandes	Bouton de commande	Permet de réaliser des actions et d'effectuer des évènements (OK et Annuler permettent généralement de refermer un formulaire).
Barre de défilement		Elle est fixée généralement sur la bordure du formulaire. Elle permet le parcours du formulaire.
Zone de liste		Elle affiche une liste ou vous pouvez sélectionner des éléments prédéfinis. Elle est généralement dépourvue de barre de défilement. elle nous permet de saisir.

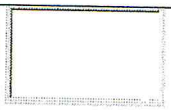
Contrôle d'édition (simple)		Réduit à sa plus simple expression, il permet de saisir et de modifier une ligne de texte.
Contrôle d'édition (complexe)		Les plus complexes contrôles d'édition permettent d'éditer des paragraphes.

Tableau 11- Récapitulatif des contrôles.

7. Contraintes de formes d'un formulaire

Les auteurs de formulaires cherchent à la fois à minimiser l'écriture de scripts et à maximiser la réutilisation des composants d'un formulaire donné. Donc, afin d'atteindre cet objectif un formulaire doit respecter les points suivants :

- *Réutilisation* : les formulaires peuvent être réutilisés indépendamment des informations collectées.
- *Indépendance vis à vis des terminaux* : les contrôles de l'interface utilisateur étant abstraits (leurs caractéristiques génériques étant seuls visibles), ils peuvent être facilement représentés sur différents terminaux avec des capacités différentes.
- *Accessibilité* : la présentation et le contenu étant séparés, l'information est davantage accessible aux utilisateurs de logiciels d'assistance. De plus, les contrôles de l'interface utilisateur en capsulent les métas données pertinents, améliorant ainsi l'accessibilité de l'application quelque soit les modalités de son utilisation.
- *La logique* : permet de définir des rapports entre les champs, par exemple, on pourra exiger des champs additionnels d'être complétés si un champ particulier est complété.
- *Les données* : permettent de définir un modèle de données pour une forme. Cette couche facilite la validation de l'information de forme.
- *Présentation :* le markup réel est utilisé avec des commandes de base de forme. Chaque commande de forme sera attachée à un champ dans le modèle de données.

En respectant ces contraintes, il sera facile de concevoir des formulaires sans se préoccuper de savoir comment ils seront utilisés ou présentés.

Chapitre 5 :
Conception

1. Introduction

La production des documents XML valides, nécessite l'appel à un personnel qualifié, pour qui des connaissances dans le domaine sont requises. Par conséquent cette production revient à des coûts élevés (temps de saisie élevé, traitement du personnel qualifié, etc.) qui peuvent représenter des contraintes financières insurmontables pour des entreprises voulant mettre leur données sous forme XML.

Pour lever ces contraintes et rendre l'édition et la production de documents XML accessible à toute personne, nous proposons un système (XGE : XML_GENERATOR_EDIT) qui d'une part, permet une saisie structurée et contrôlée des données à travers des formulaires automatiquement générés et personnalisables, et d'autre part, à la fin du processus, les documents XML seront valides par construction.

Dans la suite de ce chapitre nous allons présenter et détailler l'architecture de XML_GENERATOR_EDIT ainsi que tous les mécanismes mis en œuvre pour atteindre les objectifs visés.

2. Architecture du système

L'architecture de XML _GENERATOR_EDIT doit répondre non seulement au double objectif d'une génération d'un formulaire de saisie structuré et contrôlable et d'une production d'un document XML valide par construction, mais également elle doit présenter tous les aspects d'un éditeur avancé en offrant tous les moyens d'aide à une édition facile et aisée. Pour se faire, nous proposons dans la figure-3 l'architecture de XML_GENERATOR_EDIT. Elle s'articule autour de six parties essentielles :

- l'Analyse du SCHEMA XML.
- la Génération du formulaire.
- la Saisie des instances.
- la Génération du document XML.
- la Mise à jour.
- Adaptation du formulaire de saisie.

Dans ce qui suit, nous allons détailler ces six parties ainsi que leurs interactions :

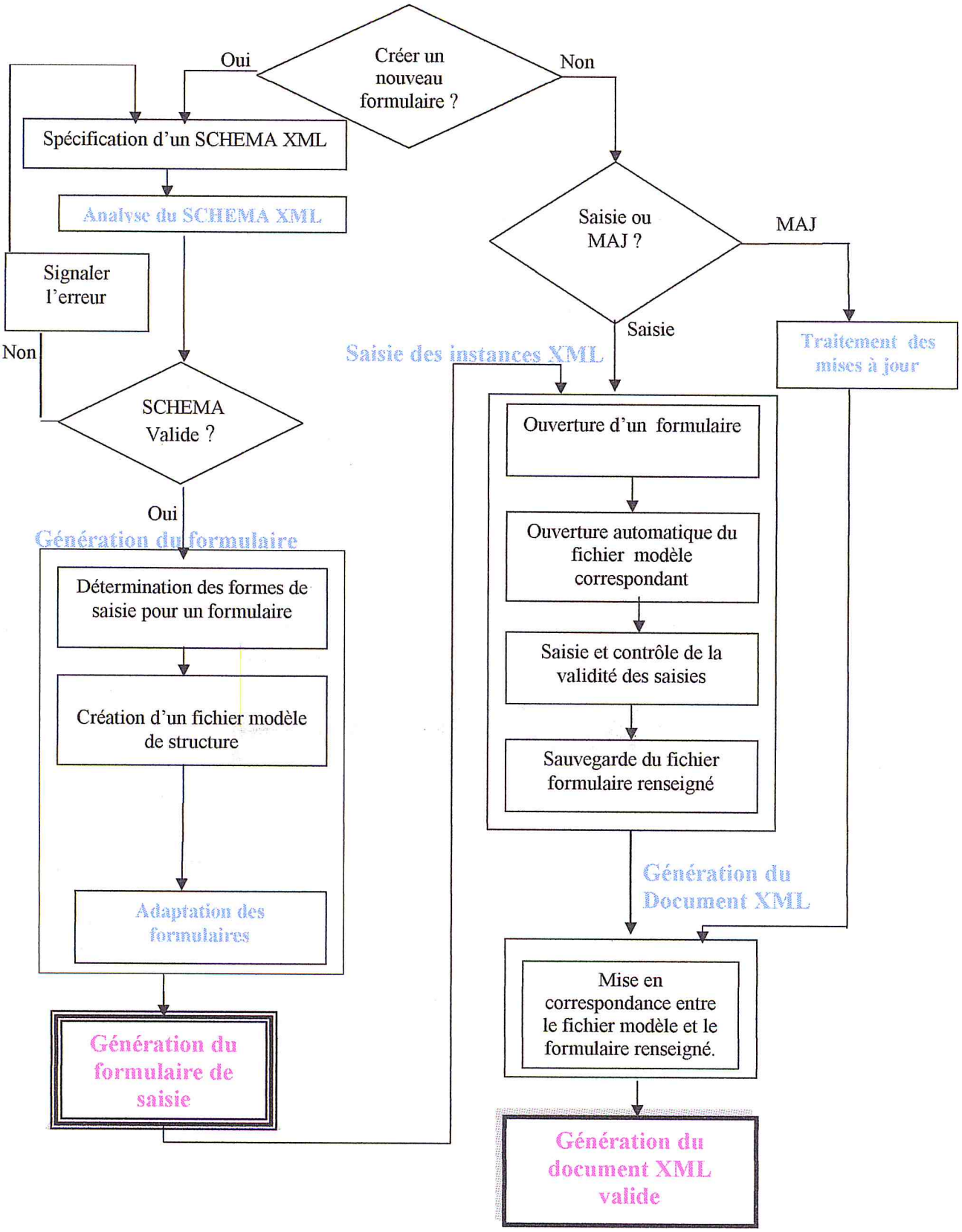


Figure 3 - Architecture générale du système

3. Analyse du Schéma XML

Cette partie représente le lancement du processus de génération des formulaires. En effet, la première action à réaliser par l'utilisateur est la spécification d'un schema XML de référence pour la production de ses documents XML valides, si celui-ci n'est pas valide après analyse, le système l'arrête à ce niveau jusqu'à validation du schema XML. Il est à rappeler que notre travail rentre dans le cadre d'un projet global de mise en place d'un éditeur validateur XML DTD et Schéma, à ce propos l'analyse d'un schema XML a été réalisée et par conséquent notre travail consiste à l'intégration de cette partie dans notre système, après avoir effectuer quelques améliorations selon la nouvelle recommandation XML schéma du W3C.

4. Génération du formulaire

Le choix du formulaire comme support de saisie, nous a été dicté non seulement par le fait qu'il représente une forme habituelle et conviviale pour toute opération de saisie, mais également du fait qu'il constitue un très bon support d'illustration visuelle d'une structure organisée et c'est justement le cas des documents XML.

4.1. Constituants d'un formulaire

Pour mieux répondre à l'objectif d'une saisie facile et conviviale, notre formulaire doit être composé, d'une part d'un ensemble de formes de saisie qu'il faut choisir en fonction du type de l'information à saisir, de son importance et de sa nature. De plus, ces formes, doivent être caractérisables par des attributs de couleur, des caractères et de trames, de dimension des traits et de nombre de lignes et de colonnes, etc. Et d'autre part, d'un ensemble de boutons de commandes à choisir en fonction du domaine de l'édition.

Après étude, nous avons retenu l'ensemble des composants suivants que nous avons scindé en deux catégories :

- ***Texte Statique*** : Sert à afficher du texte à l'utilisateur. Ce dernier ne peut pas le modifier. Il sert essentiellement à créer des étiquettes ou des intitulés comme par exemple les noms des éléments et des attributs.
- ***Champs de saisie*** : Permet à l'utilisateur d'entrer ou de modifier du texte. Il est associé au contenu d'un élément ou d'un attribut lorsqu'il est une chaîne de caractères.
- ***Champ de saisie multi ligne*** : Comme un simple *champ de saisie*, il permet à l'utilisateur d'introduire des paragraphes d'une taille considérable (un texte édité sur plusieurs lignes). Il est utilisé quand un élément est déclaré de type contenu mixte.
- ***Bouton radio*** : est employé lorsque l'utilisateur doit faire un choix parmi plusieurs options s'excluant mutuellement. Une seule option peut être activée à la fois. Il a été choisi quand le type d'un élément ou de l'attribut est un choix. Il affiche le nom de l'élément à choisir.
- ***Case à cocher*** : Elle ressemble au *bouton radio* et est utilisée pour effectuer un choix, mais les cases à cocher offrent la possibilité de sélectionner plus d'un choix parmi plusieurs. Cet outil est employé lorsque le type de l'attribut indique un choix ou un élément est de type choix.
- ***Section*** : Elle permet de fournir un mode de regroupement identifiable pour les autres formes de saisies. Les zones de ce groupe sont principalement utilisées pour diviser un formulaire par fonctions.

- ***Zone de liste déroulante*** : c'est un champ de saisie qui contient une liste d'éléments plus un bouton permettant de parcourir la liste .
- ***Sélecteur de date :*** c'est plus un bouton permettant d'afficher un calendrier pour la sélection d'une date
- ***Etoile*** : même trop utilisée dans les formulaires elle est placé à coté d'un champs de saisie pour indiquer que le champ est obligatoire.
- « ? »: Bouton aide, il permet d'afficher un commentaire sur les types données qui devront être saisies par l'utilisateur.
- **Le bouton « + » :** Il permet la duplication d'une section ou d'une forme de saisie à la demande de l'utilisateur.
- **Le bouton « - »:** Il permet la suppression d'une section ou d'une forme de saisie à la demande de l'utilisateur.

Avec cet ensemble nous sommes en mesure de traiter tous les constituants d'un SCHEMA XML (éléments de type simple ou de type complexe et les attributs…etc.) .
De plus, et comme nous sommes dans le domaine des documents structurés qui exigent, entre autre, le respect strict de la position d'un élément, ces formes doivent donc être positionnées dans l'ordre de l'apparition des éléments du SCHEMA XML associé.

4.2. Détermination des formes de saisie pour un formulaire

Comme indiquer, c'est l'exploitation des informations contenues dans le SCHEMA XML qui détermine les formes de saisie pour un formulaire. En effet, une fois le SCHEMA XML signalé Valide, XML_GENERATOR_EDIT récupère les informations se trouvant dans les composants du SCHEMA XML pour, entre autre, créer les formes correspondantes.
Le tableau suivant donne les formes de saisie en fonction des trois composants de déclaration d'un schema xml :

Composants	Formes de saisie
Element	Section, Section de choix, Section de choix facultatif, zone de liste déroulante, sélecteur de date, champ de saisie, champ de saisie multi ligne, bouton radio, case a cocher et texte statique
Attribute	champ de saisie, champ de saisie multi ligne, zone de liste déroulante, sélecteur de date, texte statique, et case à cocher.
Notation	case à cocher ou bouton radio

Tableau 9- Association des formes de saisies au composant XML schéma.

Pour ce qui est de la façon d'effectuer les choix des formes, celle-ci est détaillée dans ce qui suit :

4.2.1. Cas d'un élément :

Les formes de saisie varient selon le type ou le contenu de l'élément qu'elles représentent.

4.2.1.1 Variation des formes de saisies selon le type de l'élément :

L'élément peut avoir deux types différent (simple et complexe)

- Traitement d'un élément de type simple (primitif intégré, dérivé intégré, dérivé) :
Pour tous les éléments de type simple notre système va créer un texte statique afin d'afficher le nom de l'élément et autre forme de saisie selon son type de donnée :

- **traitement d'un élément de type primitif intégré :**

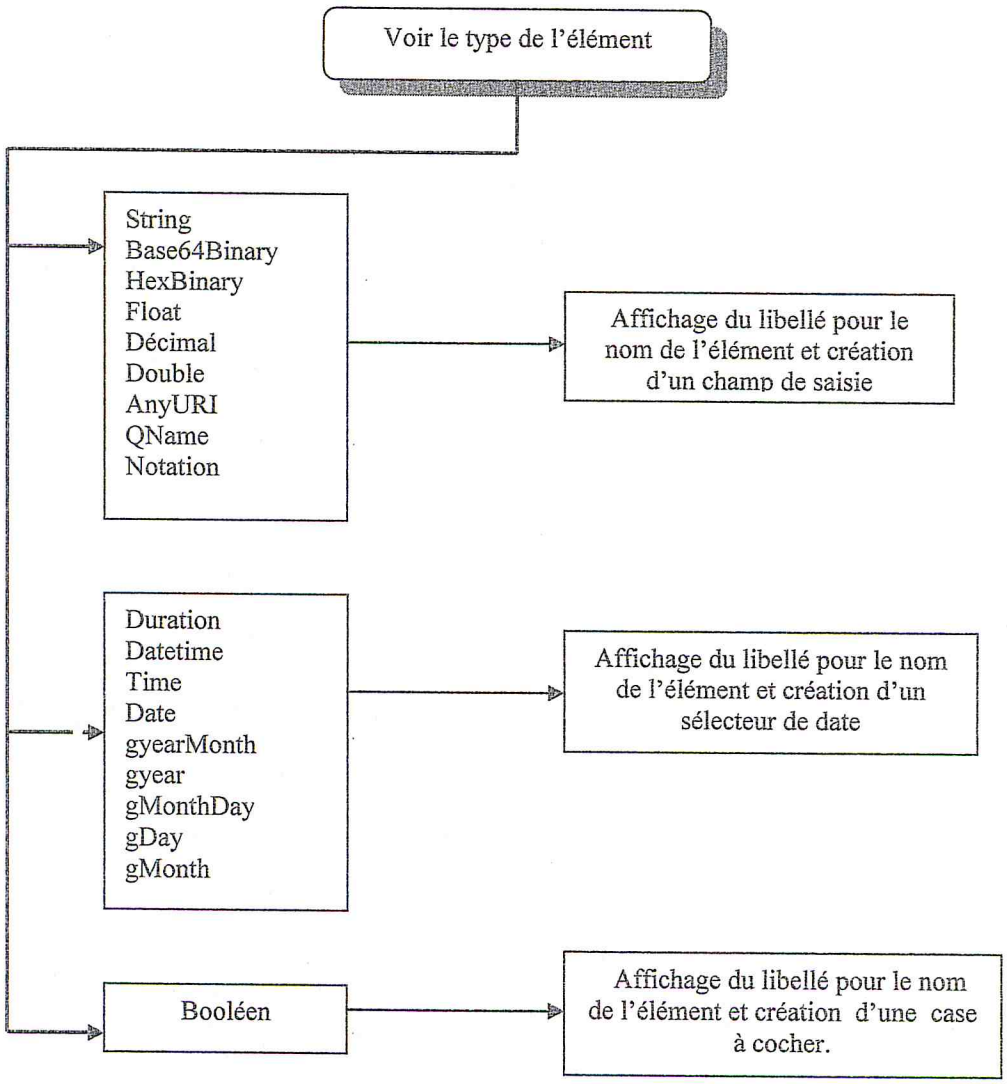


Figure 4 - Traitement d'un élément de type primitif intégré

- traitement d'un élément de type dérivé intégré :

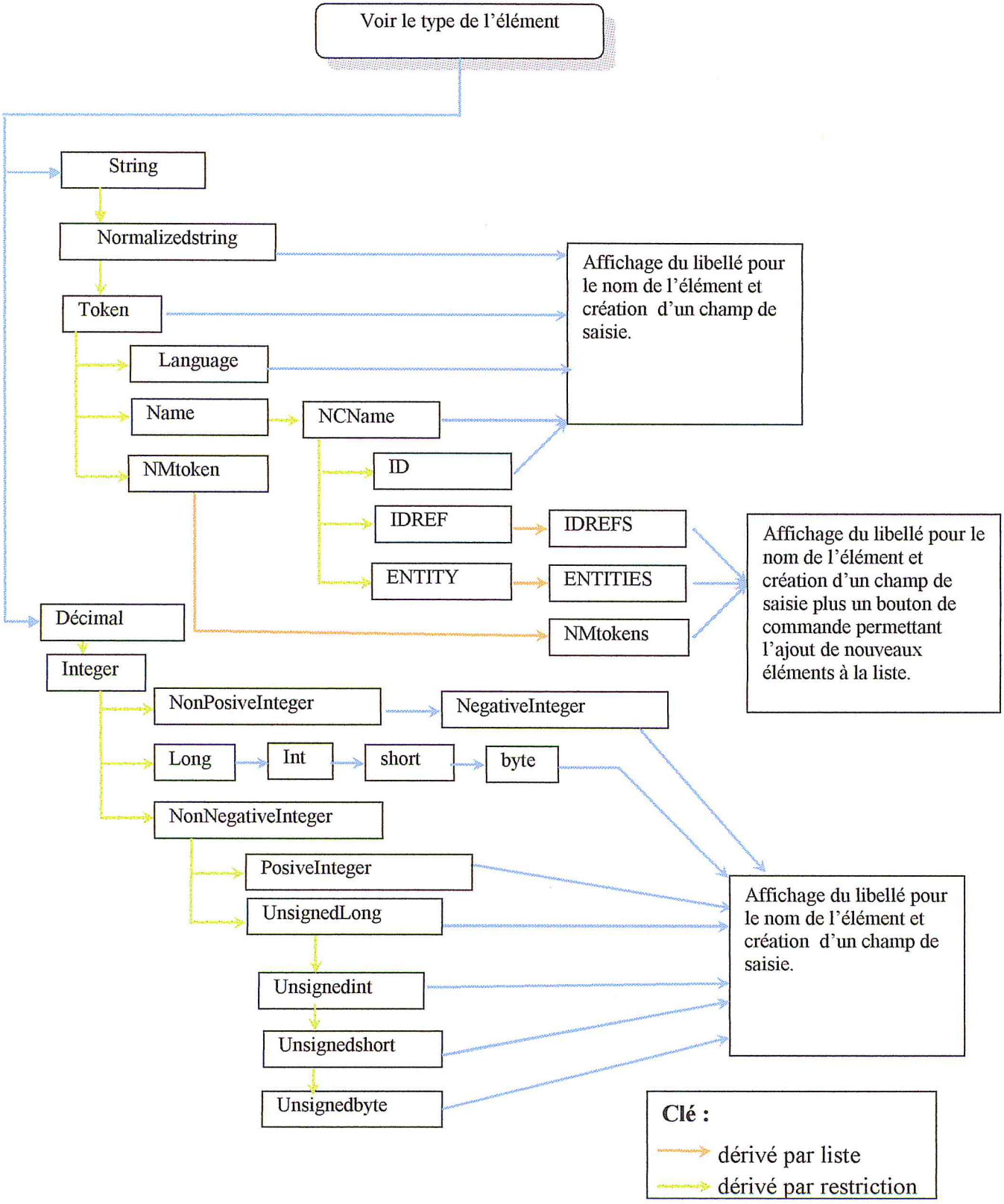


Figure 5 - Traitement d'un élément de type dérivé intégré

- Traitement d'un élément de type simple (liste, union et énumération)

Après l'étude des éléments de type primitif intégré et dérivé intégré. Nous allons aborder le traitement des éléments de type simple définis dans les schémas XML. Pour ce type, le système va créer un texte statique pour leurs noms et des formes de saisie correspondantes déduites à partir de la définition du type simple.

Les traitements pour des composants intervenant dans la définition de type simple sont donnés dans le tableau suivant.

Composants	Traitements associés
union	Il permet à un type d'être déclaré à partir d'espaces de types différents, pour les éléments de type simple définis par union, notre système crée un champ de saisie et un texte statique pour le nom de l'élément.
liste	le type simple défini par l'élément *list* est une liste d'instances de types atomiques séparés par des blancs, pour ces éléments le système crée, pour le nom de l'élément, un champ de saisie et un texte statique et un bouton de commande permettant l'ajout de nouveaux éléments à la liste.
enumeration	le type *énuméré* définit une liste de choix de valeur possibles pour un élément, le système génère une zone de liste déroulante comportant toutes les valeurs précédemment définies par énumération.

Tableau10- Traitement des éléments de types simple.

- Traitement d'un élément de type complexe

Pour ce type, le système crée une section dont l'intitulé est le nom de l'élément, quant à son contenu, il est déduit de la définition du type complexe.

La liste des traitements pour chaque composant intervenant dans la définition de type complexe est donnée ci-dessous :

- *all* : à la rencontre du connecteur *all* le système va crée une section, puis il extrait ses éléments fils un par un tout en leurs créant des formes selon leurs types et leurs modèles de contenu.

- *sequence* : à la rencontre du connecteur *sequence* tous ses fils sont traités tour à tour selon leurs types et leurs modèles de contenu.

- *choice* : à sa rencontre le système crée, pour chacun de ses fils, une section de choix et un bouton radio. Le contenu de la section est déduit à partir de son type et de son modèle de contenu.

- *group* : pour chaque groupe d'éléments défini ou référencé dans la définition du type complexe le système extrait ses éléments un par un, chacun est alors traité selon son type et son modèle de contenu.

- *attributegroup* : pour chaque groupe d'attributs défini ou référencé dans la définition du type complexe le système extrait les attributs fils du groupe qui seront à leurs tour traités un par un selon leurs types de données.

- *attribut* : pour chaque attribut déclaré ou référencé dans la définition du type complexe le système le traite selon son type de données. concernant le traitement des attributs on le verra en détail une fois le traitement des élément sera achevé.

4.2.1.2. Variation des formes de saisie selon leurs contenus

- Contenu vide

Un élément vide est un élément qui ne peut rien contenir. La plupart de ces éléments transportent toutes leurs informations via les attributs ou simplement par leurs positions par rapport à d'autres éléments.
Pour les éléments contenant des attributs notre système crée une section dont l'intitulé est le nom de l'élément, et contenant les formes de saisies correspondantes aux attributs.

- Contenu simple

Pour ses éléments, le système crée une section dont l'intitulé est le nom de l'élément, son contenu est déduit à partir de la définition du contenu simple

- Contenu complexe

Pour ces éléments une section dont l'intitulé est le nom de l'élément est créée, son contenu est déduit à partir de la définition du contenu complexe

- Contenu mixte

Un élément de contenu mixte peut contenir des données caractères analysées et des occurrences illimitées d'éléments issus d'une liste donnée, en y ajoutant la possibilité de contrôler le nombre et la séquence dans lesquels les éléments apparaissent dans les données caractères.
A ce niveau, il leurs est créé, pour les éléments fils, des formes de saisie selon leurs modèles de contenu. Ces dernières seront entrecoupées par des champs de saisie multi lignes pour les données textuelles.

- Contenu quelconque

Le système crée une section intitulée par le nom de l'élément plus un champ de saisie multi ligne permettant l'entrée des données pour les utilisateurs.

4.2.2. Cas d'un attribut

Les attributs sont traités exactement comme des éléments de type simple, à cet effet, le système crée un texte statique pour le nom de l'attribut en plus d'une forme de saisie selon le type de l'attribut.
Si de plus, la valeur de l'attribut *use* est *required,* alors une étoile est placée à coté de la forme de saisie correspondante pour marquer le caractère obligatoire de l'attribut.
Sinon, si la valeur de *use* est *prohibited,* le système ignorer l'attribut et aucun traitement ne lui est associé.
Si la valeur de l'attribut *fixed* est spécifiée, alors un champ de saisie contenant la valeur définie est crée et aucune modification n'est permise.
En fin, si la valeur de l'attribut *default* est spécifiée alors un champ de saisie contenant la valeur définie est crée, et toute modification est permise.

4.3. Création d'un fichier modèle de structure

La création d'un formulaire de saisie implique une série de traitements à réaliser (spécifier un Schéma XML, l'analyser et la génération du formulaire). Et comme il est plus que probable qu'un formulaire est amené à être réutilisé, nous avons donc pensé à offrir cette option dans le système. Pour cela, la solution consiste à mettre en place un modèle de structure unique, appelé fichier modèle, rattaché au formulaire associé, contenant la structure hiérarchique du document défini dans le Schéma. Ce dernier contient pour chaque composant défini dans le schéma XML :

- Les propriétés extraites : qui sont des propriétés obtenues à partir du schéma XML
- Les propriétés ajoutées : ce sont des propriétés ajoutées par le système afin de garder les relations existantes entre les éléments du schéma.

Le tableau suivant décrit les différentes propriétés par rapport aux composants d'un schéma.

propriétés / composants		Informations récupérées au niveau des nœud de l'arborescence (fichier modèle)	
		Propriétés extraites	**Propriétés ajoutées**
element	Type simple	name, type, maxOccurs, minOccurs, fixed, default, nillable .	Id*, idParent*, ligne*, portée*
	Type complexe	name, type, maxOccurs, minOccurs, nillable .	Id, idParent, ligne, portée
Attribute		name, type ,use, default, fixed	Id, idParent, ligne
Group		maxOccurs, minOccurs	Id, idParent, ligne
Sequence		maxOccurs, minOccurs	Id, idParent, ligne
Choice		maxOccurs, minOccurs	Id, idParent, ligne
complexType		Mixed	Id,idParent,ligne
All		maxOccurs, minOccurs	Id, idParent, ligne
simpleType		Name	Id, idParent, ligne
Restriction		Base	Id, idParent, ligne
Extension		Base	Id, idParent, ligne
AttributeGroup		Name	Id, idParent, ligne
Field		Xpath	Id, idParent
Selector		Xpath	Id, idParent
Key		Name	Id, idParent, ligne
Keyref		Name	Id, idParent, ligne
Unique		Name	Id, idParent, ligne

Tableau 11 – Les informations du modèle de structure.

Pour mieux expliquer les propriétés qui interviennent dans le fichier modèle nous introduiront dans ce qui suit des exemple génériques

- **Id et idParent** : représente le rang des éléments père et fils :

```
<xsd:element name="elt">  ──►  id=i idParent=j
  <xsd:complexType>  ──►  id=k idParent=i
    <xsd:sequence>  ──►  id=g idParent=k
      ⋮
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

- **Portée** : c'est une chaîne de caractères contenant le chemin a suivre afin d'atteindre l'élément cible :

```
<xsd:element name="elt1">  ◄── Portée="elt1"
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="elt2">  ◄── Portée="elt1/elt2"
        <xsd:complexType>  ──────────► Portée="elt1/elt2/elt3"
          <xsd:sequence>
            <xsd:element name="elt3"></xs:element>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

- **Ligne** : elle représente le numéro de la ligne de la balise ouvrante des composants du schéma.

```
<xsd:element name="elt">  ──►  ligne=i ;
  <xsd:complexType>  ──►  ligne=i+1
    <xsd:sequence>  ──►  ligne=i+2
      ⋮
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

4.4. Génération du formulaire de saisie

Une fois que la détermination des formes de saisies est parachevée, et le fichier modèle de structure créé, le système générera un formulaire de saisie destiné à recevoir les données des instances.

5. Saisie et contrôle des données

Une fois le formulaire généré l'utilisateur peut, directement ou par appel à ce dernier, commencer la production de ses documents XML à travers le renseignement du formulaire. Les données saisies sont alors sauvegardées dans une structure de liste constituée des champs ci-dessous, pour les besoins de contrôle des données et la mise à jour, éventuelle, du fichier formulaire renseigné.

Contenu : il contiendra la valeur introduite ou l'état des cases à cocher ou des boutons radio (activé ou désactivé).

Occurrence : ce champ indique l'occurrence de chaque élément se trouvant dans le formulaire.

Pour rester dans les objectifs assignés, en occurrence la production de documents XML valides par construction, le système effectue plusieurs types de contrôles le long du processus de production. La production d'un document est effective si et seulement si toutes les conditions de contrôles sont vérifiées.

5.1 Les contrôles de validité des instances

Divisés en plusieurs catégories, qu'on va détaillés dans ce qui suit :

5.1.1. Les contrôles sur la déclaration et l'occurrence des éléments et des attributs

5.1.1.1. Champ de saisie

- *Pour un élément :*

- *nillable* : indique si une valeur nulle explicite peut être assignée à l'élément. Cela s'applique au contenu de l'élément et non à ses attributs. Sa valeur par défaut est *false*, dans ce cas le champ de saisie correspondant à l'élément doit être remplie sinon un message d'erreur sera affiché signalant que le champs correspondant devra être rempli.

- *minOccurs* : c'est le nombre minimal de fois que l'élément peut apparaître dans l'élément conteneur. A ce niveau si la valeur de *minoccurs* est égale à 0 alors l'élément est facultatif et aucun contrôle ne lui sera associé, sinon (valeur > 0), le système signale une erreur dans les cas où aucune occurrence n'est introduite ou que le nombre d'occurrences de l'élément est inférieur à *minoccurs*.

- *maxOccurs* : nombre maximal de fois que l'élément peut se présenter dans l'élément conteneur. La valeur peut être un entier supérieur ou égal à 0. Par construction du formulaire

de saisie aucun élément ne peut dépasser la valeur de *maxoccurs* alors aucun contrôle n'est associé.

- *Pour un attribut :*

- *use* : indicateur de la façon dont l'attribut est utilisé :
  - *optional* : la valeur de l'attribut est optionnelle, par conséquent aucun contrôle n'est effectué à ce niveau.
  - *prohibited* : l'attribut ne peut pas être utilisé aucun contrôle ne lui est associé puisque il est ignoré lors de l'étape de la construction du formulaire.
  - *required* : la valeur est requise, c'est-à-dire obligatoire. Si l'attribut n'a pas de valeur fixe ou une valeur par défaut l'utilisateur doit saisir une valeur sinon un message d'erreur est signalé.

5.1.1.2 Zone de liste déroulante

A ce niveau on traite les attributs ou les éléments ayant un type simple défini par *enumeration.*

- Pour l'élément

- *nillable* : si la valeur *nillable* est a *false* une valeur dans la zone de liste déroulante doit être sélectionné sinon un message d'erreur signalant que l'élément ne peut pas être nulle.
- *minOccurs* : a ce niveau si la valeur de *minoccurs* égale a 0 alors l'élément est facultatif aucun contrôle ne lui est associé,sinon si minoccurs est strictement supérieur a 0 alors : si l'utilisateur ne sélectionne aucune valeur dans la liste déroulante le système s'arrête en
lui signalant l'erreur.

- Pour un attribut

- *use* : indique la façon dont un attribut est utilisé.
- *optional* : la valeur de l'attribut est optionnelle, par conséquent aucun contrôle n'est effectué à ce niveau.
- *required*: une valeur est requise, c'est-à-dire obligatoire. Si l'attribut n'a pas de valeur fixe ou une valeur par défaut l'utilisateur doit en saisir une, sinon un message d'erreur lui est signalé.

5.1.1.3. Sélecteur de date

A ce niveau le traitement sur le *Selecteur de date* est pratiquement le même que pour la zone de liste deroulante.

5.1.2. Contrôles lié aux types des éléments et des attributs

5.1.2.1 éléments de type simple ou attributs

A ce niveau nous effectuons un ensemble de contrôles qui portent sur les définitions des types simples des elements et des attributs :

- Champ de saisie

Pour le type primitif et dérivé intégré, le système prend en charge tous les types intégrés d'un schéma XML. Si la valeur saisie n'appartient pas au type déclaré un message d'erreur est alors affiché. Pour faciliter l'édition ou le renseignement du formulaire on a rajouté quelques contrôles qui sont spécifiques à chacun des types suivants :

TYPE	Contrôle associé
Décimal (et ses dérivées)	lors de la saisie seule le pavé numérique est activée toute autre touche du clavier est désactivée
ID	la valeur doit être unique dans tout le formulaire sinon un message d'erreur est affiché
IDREFS et *IDREF*	les valeurs de ces derniers doivent être celles des ID utilisés, sinon un message d'erreur est affiché.
NMTOKEN et NMTOKENS	ne doivent comporter que des caractères alphanumériques et quelques autres symboles, une restriction de touche de clavier aux caractères correspondant est opérée.

Tableau 12 – Les contrôles associés aux types intégrés spécifiques.

- **Case à cocher**

A ce niveau seul les éléments ou les attributs de type booléen seront traités aucun contrôle de type ne sera associé puisque il y aura aucune valeur à introduire.

- **Zone de liste déroulante :**

Seulement les éléments ou attributs ayant un type simple défini par énumération sont traités, aucun contrôle n'y est associé puisque les valeurs ont été validées à partir de l'étape de la validation du schéma.

En plus des contrôles associés aux types de base nous avons ajoutés des contrôles liés à la définition du type simple :

- ***union*** : définit une collection de définitions *simpleType* multiples. Si la valeur saisie n'appartient pas à l'union de type simple un message d'erreur est alors affiché.
- ***list*** : définit une collection d'une seule définition *simpleType* pour chaque valeur introduite dans la liste un contrôle de type est associe selon le type de base.
  - ***les facettes de contrainte*** : peuvent être utilisées pour contraindre les valeurs de types simples. Nous avons :
    - *énumération* : ensemble de valeurs spécifié. Limite un type de données aux seules valeurs spécifiées, par conséquent aucun contrôle de type n'est associe.
    - *fractionDigits* : valeur avec un nombre maximal spécifique de chiffres dans la partie fractionnaire. En plus du contrôle associé au types de base, un autre contrôle qui permet de détecter l'erreur, si le nombre de chiffre dans la partie fractionnées dépasse la valeur *totaldigits* est réalisé.
    - *length* : nombre d'unités de longueur. Les unités de longueur dépendent du type de données. Si le nombre d'unité de longueur de la valeur introduite dépasse la valeur spécifié par *length* un message d'erreur signalant l'erreur.
    - *maxExclusive* : valeur de limite supérieure. Si la valeur introduite est supérieur ou égale à la valeur spécifié par *maxexclusive* un message d'erreur est affiché.

- *maxInclusive* : valeur maximale. Si la valeur saisie est supérieure à la valeur spécifiée par *maxInclusive* un message d'erreur est affiché.

- *minExclusive* : valeur de limite inférieure. Si la valeur introduite est inférieure ou égale à la valeur spécifié par *minExclusive* un message d'erreur est affiché.

- *minInclusive* : valeur minimale. Si la valeur saisie est inférieur à la valeur spécifiée par *minInclusive* un message d'erreur est affiché .

- *maxLength* : nombre maximal d'unités de longueur. Si le nombre d'unité de longueur de la valeur saisie est supérieur à la valeur spécifiée par *maxLength* un message d'erreur est affiché.

- *minLength* : nombre minimal d'unités de longueur. Si le nombre d'unité de longueur de la valeur saisie est inférieur a la valeur spécifiée par minLength un message d'erreur signalant l'erreur est affiché.

- *pattern* : La valeur de pattern doit être une expression régulière. Si la valeur saisie n'appartient pas au langage défini par l'expression régulière un message d'erreur est affiché.

- *totalDigits* : valeur avec un nombre maximal spécifique de chiffres décimaux. Si la valeur saisie par l'utilisateur a un nombre de chiffres décimaux supérieur à la valeur spécifié par *totalDigits* un message d'erreur est affiché.

- Elément de type complexe

A ce niveau en plus des contrôles qui portent sur la déclaration et l'occurrence des éléments, le système prend en charge les contrôles liés aux composants intervenants dans la définition du type complexe.

- Section : À ce niveau par construction du formulaire l'ordre d'apparition des éléments fils du connecteur séquence est respecté, de là aucun contrôle n'est associé.
- Section de choix : A ce niveau les contrôles seront associé aux éléments fils du connecteur de choix. *<xsd :choice >* permettant à un et un seul des éléments contenus dans la section sélectionnée d'être présent dans l'élément conteneur. Si la valeur de *minOccurs* est supérieure à zéro et aucun bouton radio n'est activé alors un message d'erreur est affiché.

5.1.3. Les contrôles infligés par les éléments de définition de contraintes d'identité

A ce niveau chaqu'un des deux éléments de définition de contraintes d'identité aura un traitement spécifique :

- Le composant *<Key>* :

L'élément *<Key>* permet de définir une clé dans une structure XML. Notre système récupére les valeurs spécifiées par les éléments *Field* suivant la portée indiquée par l'élément *<Selector>*, les valeurs doivent être uniques et non nulles dans tout le chemin sinon un message d'erreur est affiché.

- Le composant <unique> :
L'élément <unique> permet de définir une clé unique dans une structure XML. Le système récupère la liste des valeurs spécifiée par les éléments *Field* suivant la porté indiqués par les éléments <*Selector*>, les valeurs doivent être unique dans tout le chemin sinon un message d'erreur est affiché

6. Génération du document XML Valide :

Un document XML se compose, d'une part, de texte et d'autre part d'informations de structure. Les informations de structure servent le plus souvent à délimiter le texte, pour identifier essentiellement la sémantique et ce par le biais de balises. Afin d'aboutir à la génération d'un document XML respectant cette structure d'affichage, XML_GENERATOR_EDIT utilise les groupes de données pour récupérer les instances des éléments et des attributs , et les fichiers modèle pour respecter la hiérarchie du document à généré.

Rappelons qu'un document XML est constitué essentiellement de deux parties :

- Prologue.
- Arbre XML.

La figure suivante schématise la génération du document XML :

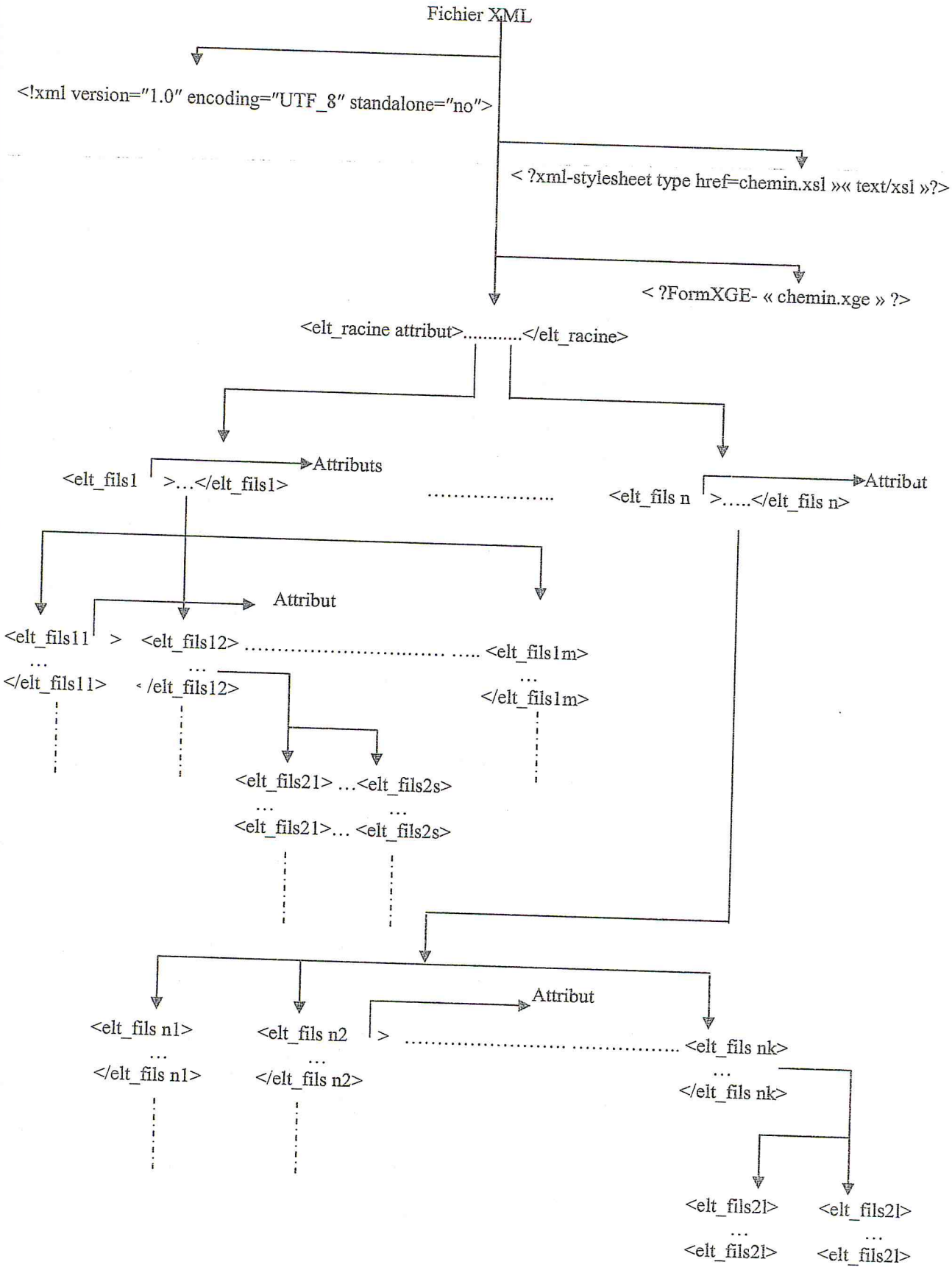


Figure 6 - Génération d'un document XML

Nous allons dans ce qui suit détailler la procédure de génération d'un document XLM.

6.1. Génération du prologue

Il s'agit de la partie introduction d'un document XML. Cette partie inclut les balises de déclaration XML et les balises des Instructions de traitement.

6.1.1. Génération de la balise Déclaration XML

Etant donné que cette balise fournie des informations sur la version de XML, le codage utilisé et l'autonomie du document. Le système doit impérativement la générer en premier, afin de respecter les contraintes du Langage XML. L'affectation des valeurs pour ces attribut est faite comme ci-dessous :

- L'attribut *version* : le système lui affecte la valeur "1.0" par défaut car c'est la version traitée par notre système.
- L'attribut *encoding* : il prend la valeur "ISO-8859-1" car ce jeu de caractères a, pour les francophones, l'avantage d'accepter la plupart des lettres avec des accents. Si ce codage ne répond pas au besoin de l'utilisateur, le système lui donne la main pour inclure le codage qui lui convient par le biais d'une boite de dialogue qui contient la liste de tous les codages.

La balise générée a la syntaxe suivante :

```
<?xml version="1.0"encoding="ISO-8859-1"standalone="no"?>
```

6.1.2. Génération de la balise feuille de style

L'instruction de traitement permet de fournir des informations supplémentaires sur le document XML. Les feuilles de styles sont des instructions de traitement apportant une information sur la mise en forme et l'affichage d'un document XML. Afin de permettre à l'utilisateur d'associer une feuille de style au document s'il le désire, le système met à sa disposition une boite de dialogue pour introduire le chemin de la feuille (Chemin_fichier.xsl). Cette balise à la forme suivante :

```
<?xml-stylesheet  href ="Chemin_fichier.xsl" type = " text/xsl "?>
```

6.1.3 Génération de la balise FormXGE

A ce niveau, nous avons introduit une balise de type instructions de traitement, créée par le système permettant de sauvegarder le chemin du fichier formulaire renseigné et servant pour les besoins de mise à jour et de réutilisation du formulaire. Elle se présente sous la forme suivante :

```
<?FormXGE ="Chemin_xgel"?>
```

6.2. Génération de l'arbre XML

Un document XML est une imbrication de balises d'éléments qui doivent apparaître dans l'ordre de leurs déclarations dans le SCHEMA XML. Afin d'assurer la production des documents XML valides, le système procède à la génération des balises de l'élément racine

suivie de la génération des balises des éléments fils. Le mécanisme de génération se base sur la récupération des données contenues dans le fichier de formulaire renseigné en les mettant en correspondance avec le fichier modèle.

6.2.1. Génération de la balise élément racine :

L'élément racine est obligatoire et unique dans un document XML. Il encadre l'arbre du document. Le système récupère le nom de cet élément du fichier modèle de structure et le chemin du Schéma XML associé et génère par la suite ses balises comme suit :

```
<Nom_racine xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
            xsi : noNamespaceSchemaLocation="Chemin_schema.xsd" >
                    :
                    :
            </Nom_racine>
```

6.2.2. Génération des balises des éléments :

La génération de ces balises se fait selon la disposition des attributs. Si un élément possède un attribut, il est généré dans la balise de début de l'élément. Dans le cas contraire rien n'est généré et la balise ne contient aucun attribut. A cet effet la génération des balises se fait selon leurs contenus et de leurs types.

- Élément à contenu vide

Le système crée la balise de l'élément ainsi que ses attributs s'il en a comme suit :

```
<Nom_element Nom_attribut1="valeur_attribut1" Nom_attribut2="valeur_attribut2"..... >
                    </Nom_element>
```

- Élément a contenu nul

Le système crée la balise de l'élément ainsi que ses attributs s'il en a en ajoutant l'attribut d'instance *xsi :nill* comme suit :

```
<Nom_element xsi :nill Nom_attribut1="valeur_attribut1"..... >
                    </Nom_element>
```

- Élément a contenu mixte

Le système crée la balise de l'élément ainsi que ses attributs s'il en a en ajoutant les balises des éléments fils selon leurs contenus qui seront entrecoupées par le contenu des champs de saisie multi ligne correspondant au contenu non analysable.

```
<Nom_element Nom_attribut1="valeur_attribut1" Nom_attribut2="valeur_attribut2"..... >
                        Contenu1
    <Nom_elementfils1> ..... </Nom_elementfils1>
                        Contenu2
    <Nom_elementfils1> ..... </Nom_elementfils1>
                        Contenu3
    <Nom_elementfils1> ..... </Nom_elementfils1>
                        Contenu4
</Nom_element>
```

- Élément a contenu simple

Le système crée la balise de l'élément ainsi que ses attributs s'il en a en ajoutant comme contenu la valeur récupérée à partir de la forme de saisie correspondante à l'élément.

```
<Nom_element Nom_attribut1="valeur_attribut1" .....> valeur_element </Nom_element>
```

La variété des types de données des Schéma XML exige quelques traitements en plus pour la génération des valeurs des éléments de type simple et des attributs.

NMTOKENS IDREFS

Les valeurs d'élément ou d'attribut sont séparées par des blancs :

```
Nom_attribut=" valeur _attribut1 valeur _attribut2..... valeur _attribut n"

<Nom_element > valeur_element1 valeur_element2 valeur_element3 </Nom_element>
```

ENTITIES

Le système génère les appels entités qui correspondent aux entités sélectionnées. Elles sont séparées par des blancs.

```
Nom_attribut="&nom_entités1; "&nom_entités2; ..... "&nom_entités n;"

<Nom_element >&nom_entités1; &nom_entités2; ..... &nom_entités n </Nom_element>
```

BOOLEAN

Le système génère la valeur de l'élément ou de l'attribut selon l'état de la case à cocher Si elle est cochée :

```
Nom_attribut="True"

<Nom_element >True </Nom_element>
```

Sinon :

```
Nom_attribut="False"

<Nom_element >False </Nom_element>
```

Pour tout autre type d'éléments ou d'attributs les valeurs seront générées sans être modifiés :

```
Nom_attribut="valeur_attribut"

<Nom_element >valeur_element</Nom_element>
```

7. Mise à jour

Pour une utilisation optimale du système, nous offrons à l'utilisateur la possibilité d'effectuer des mises à jours sur ce qui a été saisi et crée. Nous avons défini deux types de mise à jour :

- une mise à jour sur les instances introduites.
- une mise à jour de la structure du document XML, elle concerne :
  - La modification des éléments et des attributs.
  - L'ajout d'un élément ou d'un attribut.
  - La suppression d'un élément ou d'un attribut.

La figure ci-dessous montre les différents traitements réalisés :

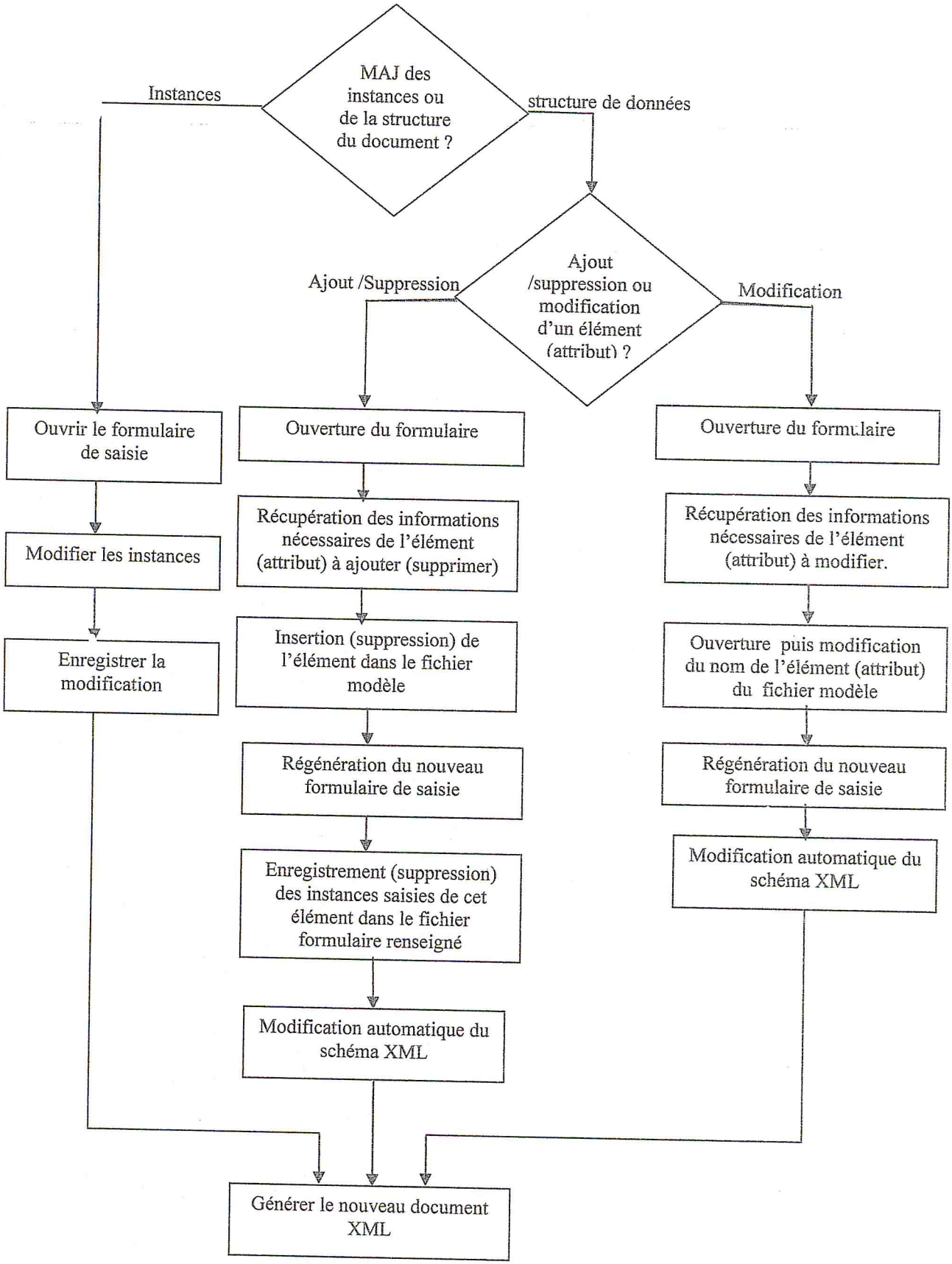


Figure 7 - Traitements des Mises à jour

7.1. Mise à jour des instances

A ce niveau, il s'agit du cas où l'utilisateur aimerait effectuer des mises à jour sur son document XML. Alors comme cela était le cas pour la première saisie, le système doit lui afficher son document XML sous forme d'un formulaire renseigné. A cet effet, le système l'invite à sélectionner le formulaire correspondant et d'accomplir les mêmes opérations que lors d'une saisie normale.

7.2. Mise à jour de la structure du document XML

Le système permet cette mise à jour dans le cas ou l'utilisateur aimerait effectuer des mises à jour sur la structure de son document après la saisie d'un nombre considérable d'instances. Pour ce faire, le système propose les opérations suivantes :

- Ajout d'un élément simple ou d'un attribut.
- Suppression d'un élément ou attribut.
- Modification d'un élément ou d'un attribut.

7.2.1. Ajout d'un élément ou d'un attribut

Pour pouvoir ajouter un élément ou un attribut à un formulaire qui existe déjà, le système affiche une boite de dialogue pour récupérer les données nécessaires. Cette boite recueille les informations suivantes :

- Le nom de l'élément (attribut) à ajouter.
- Le type de l'élément (attribut) à ajouter
- les contraintes sur le type de l'élément (attribut) à ajouter.
- L'occurrence de l'élément (*maxOccurs* ,*minOccurs*).

L'élément ou l'attribut est alors rajouté au fichier modèle ainsi que toutes les données récupérées Le nouveau formulaire contenant cet élément est régénéré, pour laisser à l'utilisateur la possibilité d'introduire ses instances et par conséquent, générer, par la suite, le document XML valide correspondant.

7.2.2. Suppression d'un élément ou d'un attribut

La suppression d'un élément ou d'un attribut nécessite la spécification du formulaire afin de récupérer les informations suivantes:

- Pour un élément : le nom de l'élément.
- Pour un attribut : le nom de l'attribut.

Une boite de dialogue est présentée à l'utilisateur pour récupérer ces informations. Le système supprime automatiquement cet élément ou l'attribut du fichier arborescence et formulaire renseigné. Le nouveau formulaire et le document XML associé sont alors régénérés.

7.2.3. Modification d'un élément ou d'un attribut

La modification elle-même suit une série de traitement, XML_GENERATOR_EDIT récupère le nom du formulaire et les informations nécessaires via la boite de dialogue qui est affichée à la demande de l'utilisateur. Les données à récupérer sont :

- Pour un élément : son nom, le nouveau nom.
- Pour un attribut : le nom de l'attribut, le nouveau nom.

Le système récupère ces informations et remplace le nom de l'attribut ou le nom de l'élément par la nouvelle valeur dans le fichier arborescence. Le fichier formulaire renseigné n'est pas

modifié. Le nouveau formulaire est affiché pour permettre à l'utilisateur de générer le nouveau document XML.

Il est à signaler qu'après une mise à jour de la structure du document, le document XML généré est bien formé par construction mais n'est plus valide par rapport au schéma mis à jour. Pour cela, et afin d'assurer la validité des documents XML ainsi générés, le système crée automatiquement un nouveauté schéma, dérivé du schéma initial.

8. Adaptation des formulaires

Afin d'assurer une ergonomie à même d'aider et d'assister l'utilisateur du système, XML_GENERATOR_EDIT propose à l'utilisateur la personnalisation de l'interface de saisie avant de la générer. Cette personnalisation touche la présentation des champs de saisie, du fond du formulaire, la police des caractères et la couleur. Le système propose à l'utilisateur les options suivantes :

- La taille des champs de saisie
- La couleur du fond du formulaire.
- La couleur des items du formulaire.
- La couleur du fond des champs de saisie
- La couleur du texte des champs de saisie.
- L'image associée au formulaire.
- La police des caractères par champ ou par intitulé.

Si l'utilisateur n'a aucun désire a personnaliser le formulaire, le système attribut des couleurs par défaut pour le texte introduit dans certains champs de saisie afin d'attirer l'attention de l'utilisateur. Par exemple, le champ de saisie de type *ID* est souligné en rouge, et le champ de saisie de type IDREF ou IDREFS est souligné en bleu pour signaler leur importance.

Chapitre 6 : Réalisation et mise en œuvre

1. Introduction

Après la présentation des différents choix conceptuels effectués, nous allons entamer la mise en œuvre du système. Dans cette présente partie du rapport, nous exposerons la réalisation d'XML_GENERATOR_EDIT.

2. L'environnement de développement

Nous avons utilisé comme environnement de développement pour la réalisation du système le *Visual Studio.Net 2003*, il offre tous les avantages de la programmation orientée objet tel que la modularité et l'héritage tout en nous donnons la possibilité d'accéder aux services de la plateforme .NET (.NET FrameWork).

Nous avons utilisées plusieurs bibliothèques de classes offertes par la plateforme .NET tel que :

- ***System.Xml*** :

  L'espace de noms System.Xml fournit une prise en charge standard du traitement XML. Les standards pris en charge sont :

- XML 1.0 - http://www.w3.org/TR/1998/REC-xml-19980210 - y compris prise en charge DTD.
- Espaces de noms XML - http://www.w3.org/TR/REC-xml-names/ - tant au niveau des flux que de DOM.
- Schémas XSD - http://www.w3.org/2001/XMLSchema
- Expressions XPath - http://www.w3.org/TR/xpath
- Transformations XSLT - http://www.w3.org/TR/xslt
- Noyau DOM Niveau 1 - http://www.w3.org/TR/REC-DOM-Level-1/
- Noyau DOM Niveau 2 - http://www.w3.org/TR/REC-DOM-Level-2/

- ***System.Xml.Schema*** :

  L'espace de noms System.Xml.Schema contient les classes XML qui assurent la prise en charge standard des schémas XSD (XML Schema Definition). Les standards pris en charge sont :

- Prise en charge de XML Schemas for Structures http://www.w3.org/TR/xmlschema-1/ - pour schémas de mappage et de validation.
- Schémas XML for Data Types - http://www.w3.org/TR/xmlschema-2/ - prend en charge les types de données pour les définitions XML Schema (XSD).

- ***System.Data***

Se compose principalement des classes qui constituent l'architecture ADO.NET. L'architecture ADO.NET permet de construire des composants qui gèrent efficacement les données provenant de plusieurs sources de données. Dans un scénario déconnecté (tel qu'Internet), ADO.NET fournit les outils permettant de demander, mettre à jour et rapprocher les données de systèmes à plusieurs couches. L'architecture ADO.NET est également implémentée dans les applications clientes, telles que Windows Forms ou les pages HTML créées par ASP.NET.

- ***System.IO***
  L'espace de noms System.IO contient des types qui permettent la lecture et l'écriture dans des fichiers et des flux de données, ainsi que des types qui permettent la prise en charge de fichiers et de répertoires de base.

- ***System.Windows.Forms*** **:**
  L'espace de noms **System.Windows.Forms** contient des classes permettant la création d'applications Windows qui tirent parti des fonctionnalités d'interface utilisateur évoluées disponibles dans le système d'exploitation Microsoft Windows.

  Les classes dans cet espace de noms se répartissent dans les catégories suivantes :

- Control, User Control, and Form.
- Controls.
- Components.
- Common Dialog Boxes.

Le source du système comporte environ 40000 lignes de code. Il manipule les classes suivantes :

Les Classes basées sur *UserControl* :
attributdetypesimple, attributdetypeenuméré, attributdetypenmtokens, attributdetypeidrefs, attriutdetypedate, elemntdetypenmtokens, elementdetypeidrefs, elementdetypesimple, elementdetypedate

Les Classes basées sur GroupBox :
Element, Choice, Group, Sequence, filsdechoix, filsdeall, All

Il est à signaler que ces classes sont toutes créées par XML_GENERATOR_EDIT selon ses besoins.

3. Description de l'interface de XML_GENERATOR_EDIT

L'interface du système comporte quatre barres, la barre de titre, la barre de menu, barre d'outils et barre d'état :

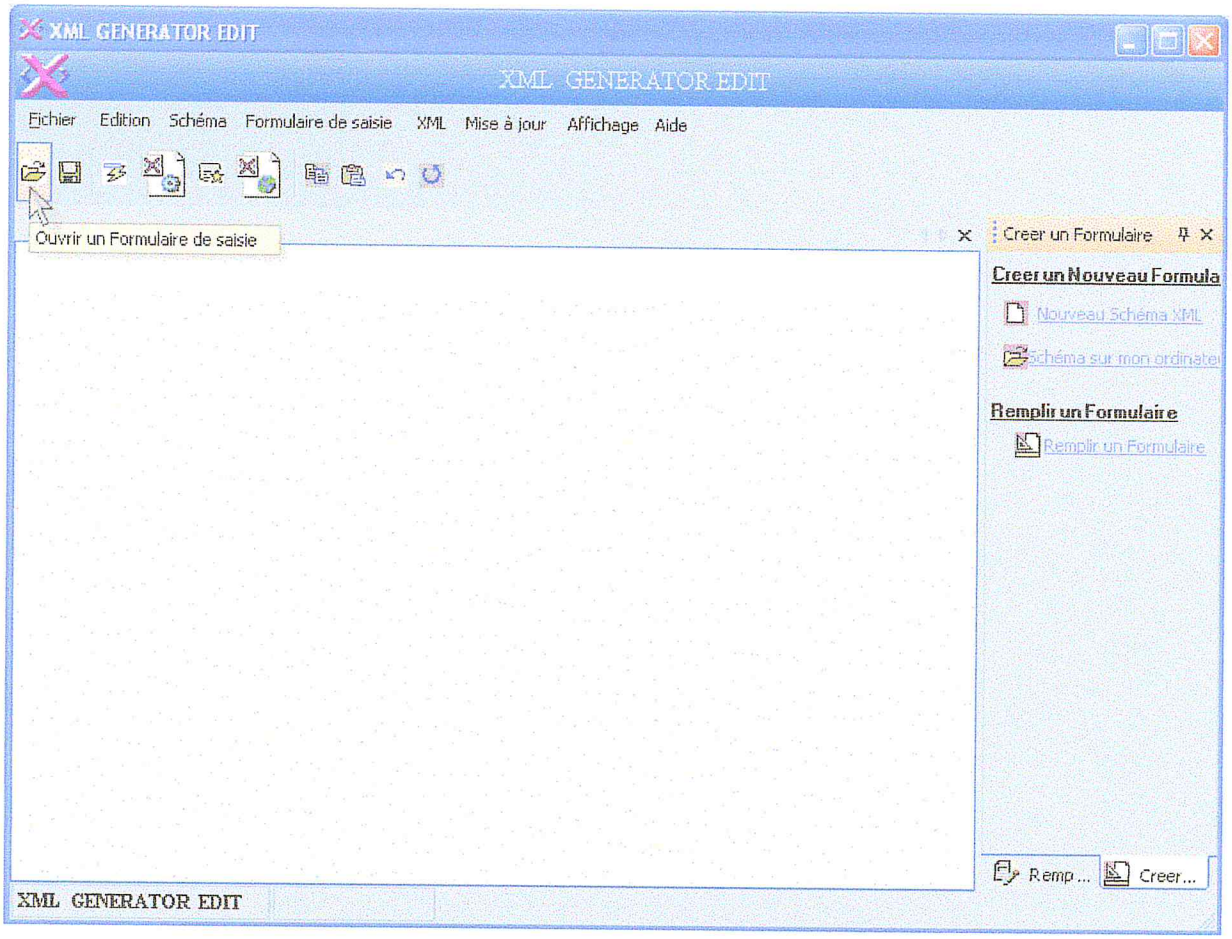


Figure 8- Interface de XML_GENERATOR_EDIT

-Barre de titre : Elle contient :

- Le logo de l'application.
- Le nom de l'application : XML_GENERATOR_EDIT.
- Le nom du document ou du formulaire dans le cas ou l'utilisateur l'agrandisse.
- Des icônes qui permettent de manipuler la fenêtre du document :

Fermer provisoirement.
Agrandir. Réduire
Fermer définitivement.

- La barre de menu : Elle contient les menus de l'application. Ils représentent les opérations offertes par le système. Chaque menu contient un ensemble d'options avec des raccourcis clavier. L'interface de XML_GENERATOR_EDIT comporte les menus suivants :

- Le menu Fichier : Il contient les options suivantes :

  - Option *Ouvrir un schéma* : Permet de spécifier le schéma XML à partir d'un emplacement afin de générer un nouveau formulaire.

  - Option *Ouvrir un formulaire* : Permet à l'utilisateur d'ouvrir un formulaire crée précédemment.

  - Option *Remplir un formulaire* : Permet à l'utilisateur d'ouvrir la boite remplir un formulaire qui contient à son tour :
    - Option plus de formulaire : Permet d'ouvrir un modèle de formulaire existant sur l'ordinateur.
    - Option formulaire sur mon ordinateur : Permet à l'utilisateur d'ouvrir un formulaire créé précédemment.

  - Option *créer un formulaire* : Permet à l'utilisateur d'ouvrir la boite créer un formulaire qui contient à son tour :
    - Option nouveau schéma : Permet l'édition d'un nouveau schéma .
    - Option schéma sur mon ordinateur : Permet à l'utilisateur d'ouvrir un schéma créé précédemment.

  - Option enregistrer : Permet à l'utilisateur d'enregistrer le formulaire.

  - Option enregistrer sous : Permet à l'utilisateur d'enregistrer le formulaire dans un autre emplacement.

  - Option *Quitter* : Permet à l'utilisateur de quitter d'une façon définitive l'application.

- Le menu Edition : Il contient les options suivantes :
  - Option copier : Permet à l'utilisateur de copier du texte.
  - Option coller : Permet à l'utilisateur de coller du texte.
  - Option sélectionner tout : Permet à l'utilisateur de sélectionner du texte.

- Le menu Schéma : Il contient les options suivantes :
  - Option Valider le schéma : Permet à l'utilisateur de valider le schéma XML.
  - Option Générer le formulaire de saisie : Permet à l'utilisateur de générer le formulaire de saisie à base du schéma XML.

- Le menu Formulaire de saisie : Il contient les options suivantes :
  - Option Validation des entités utilisateurs: Permet à l'utilisateur de valider les données saisies.

  - Option personnaliser le formulaire de saisie : Permet à l'utilisateur d'afficher une boite de dialogue qui lui permettra de modifier les propriétés du formulaire telles que : Le titre, la couleur du fond, la couleur des intitulés, la police des intitulés et l'association d'image au formulaire.

- Le menu XML : Il contient l'option suivante :
  - Option Générer le document XML : Permet à l'utilisateur de générer le document XML à base du formulaire.

- Le menu Mise à jours : Il contient les options suivantes :

  - Option Modifier : Permet à l'utilisateur faire des modifications dans son formulaire contient à son tour :
    - Option Intitulé : Permet de modifier les intitulés des formes de saisies.
    - Option Type : Permet à l'utilisateur de modifier le type des éléments et des attributs.
    - Option commentaire : Permet à l'utilisateur de modifier les commentaires liés aux champs de saisies.

  - Option Ajouter : Permet à l'utilisateur d'ajouter des éléments ou des attributs à son formulaire, elle contient à son tour :
    - Option Au dessous : Permet à l'utilisateur d'ajouter des éléments ou des attributs au dessous de l'élément ou de l'attribut voulu.
    - Option Au dessus : Permet à l'utilisateur d'ajouter des éléments ou des attributs au dessus de l'élément ou de l'attribut voulu.
    - Option Supprimer : Permet à l'utilisateur de supprimer un élément ou un attribut.

- Le menu Affichage : Il contient les options suivantes :
  - Option Barre d'outils : Permet à l'utilisateur d'afficher ou de masquer la barre d'outils.
  - Option Barre d'état : Permet à l'utilisateur d'afficher ou de masquer la barre d'état.

- Le menu Aide : Il fournit des informations sur le système.

- **La barre d'outils :** La barre d'outils contient des raccourcis aux menus cités dans la barre de menu.

- **Barre d'état :** Elle fournit des informations sur l'opération en cours.

4. Fonctionnalités du système

Le système XML_GENERATOR_EDIT est réalisé afin de répondre aux objectifs attendus. Il permet :

- La génération d'un formulaire de saisie structuré.
- Une saisie contrôlable des données.
- La production d'un document XML valide par construction.
- Les opérations de mise à jour sur les documents générés et les formulaires renseignés.
- L'adaptation du formulaire.
- La réutilisation d'un formulaire.

Afin de présenter ces fonctionnalités, nous allons baser notre démonstration sur des captures écrans réalisées sur le système :

4.1 Génération d'un formulaire de saisie structuré

La génération d'un formulaire doit passer par les étapes suivantes :

- Etape1 : Spécification du schéma

Afin de générer un formulaire, l'utilisateur doit spécifier un schéma XML et l'analyser. Le système lui propose deux façons pour l'éditer :

- Soit l'utilisateur crée un nouveau Schéma, et cela en cliquant sur l'option ***créer un formulaire*** du menu ***Fichier***, et cliquer ensuite sur ***nouveau schéma*** se trouvant dans la boite créer formulaire. Un nouveau document sera affiché et l'utilisateur peut saisir le Schéma.
- Soit, le Schéma existe déjà et l'utilisateur doit cliquer sur l'option ***Ouvrir Schéma*** du menu ***Ficher***, le spécifier dans la boite de dialogue ***Ouvrir***, qui est à la disposition de l'utilisateur dés qu'il choisit l'option *Ouvrir Schéma* du menu *Nouveau formulaire*.

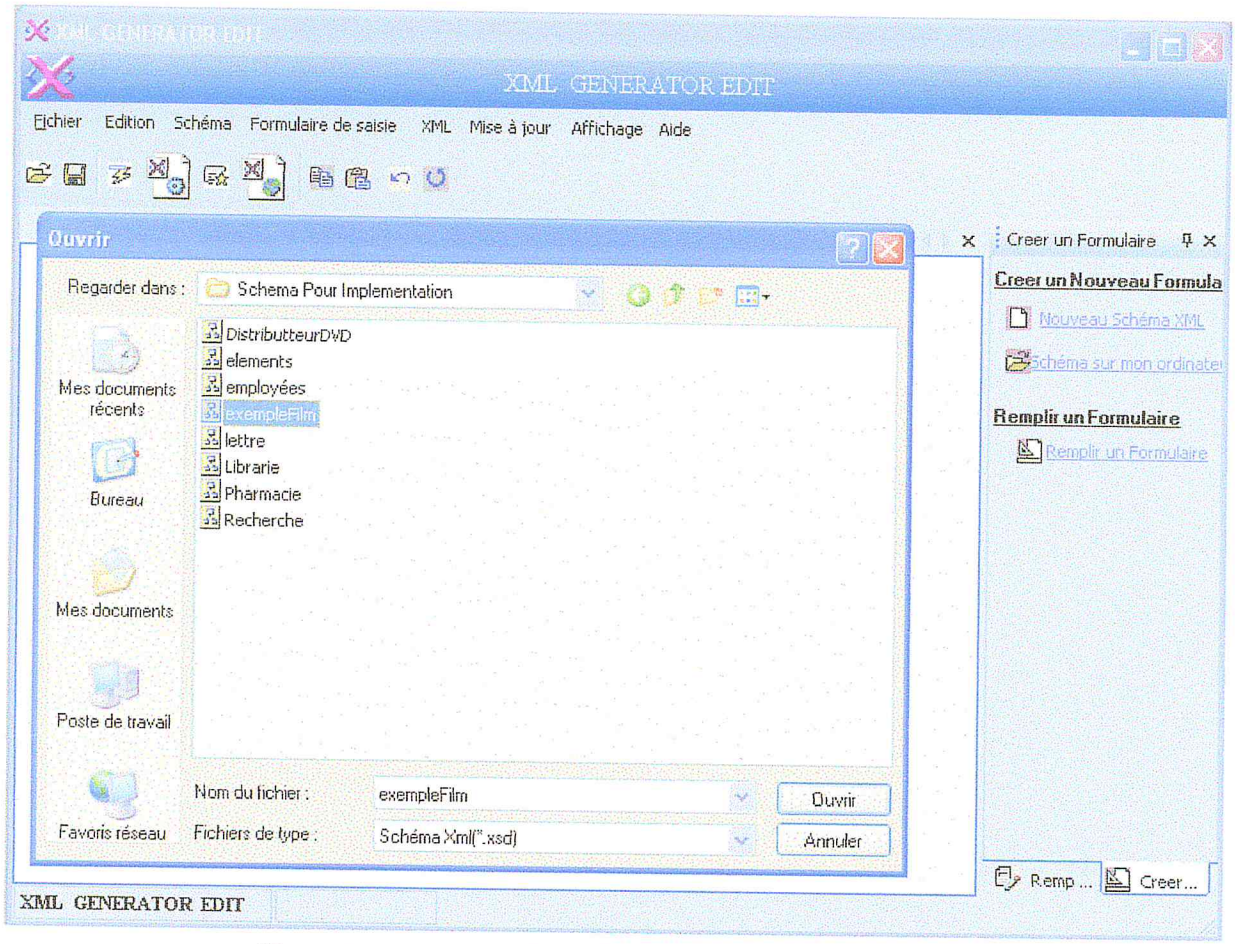


Figure 9- Spécification d'un Schéma XML « Ex : Schéma Film »

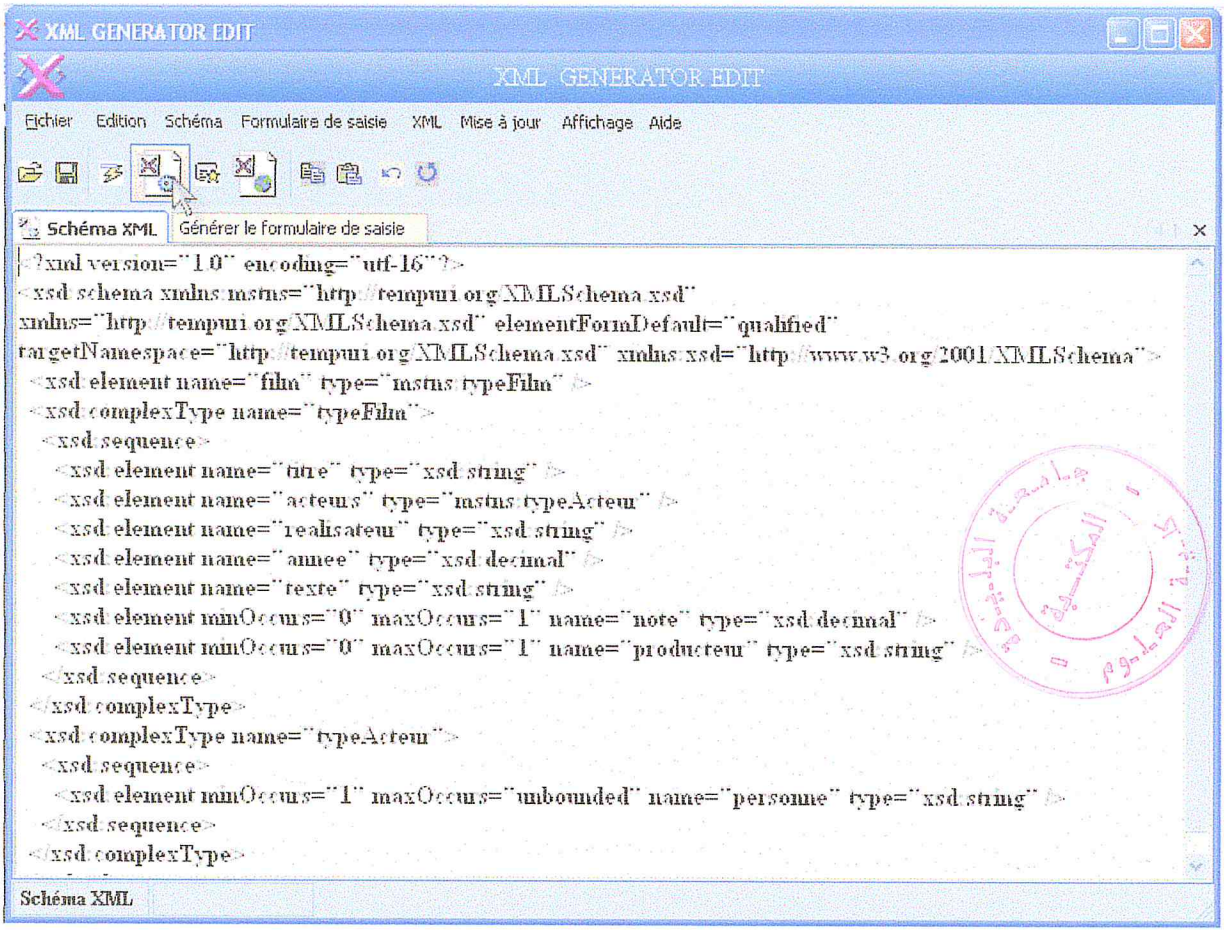


Figure 10- Affichage du Schéma XML « Ex : Schéma Film »

- Etape 2 : Analyse du Schéma XML

L'analyse du Schéma XML représente la condition de base pour pouvoir continuer le processus de la génération du formulaire, et par conséquent, la génération du document XML valides. Pour ce faire, l'utilisateur doit cliquer sur l'option ***Valider schéma*** du menu ***Schéma.*** Une boite de dialogue indiquant l'état du Schéma (valide ou non valide) est affichée. Dans le cas positif, le processus de génération peut continuer son exécution. Cependant, si le Schéma est non valide, les lignes des erreurs sont affichées dans une boite appelée résultat de validation du Schéma XML.

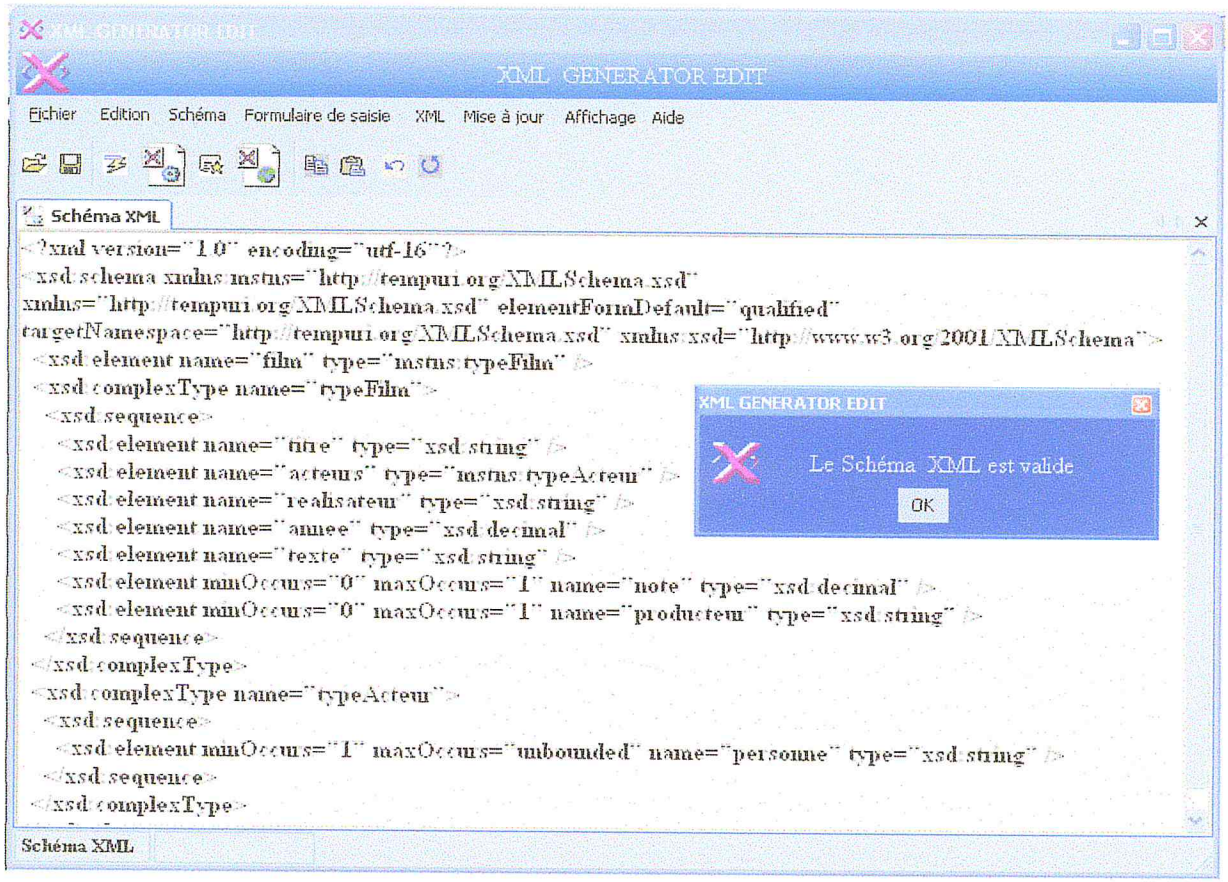


Figure 11- Analyse du Schéma XML « Ex : Schéma Film valide»

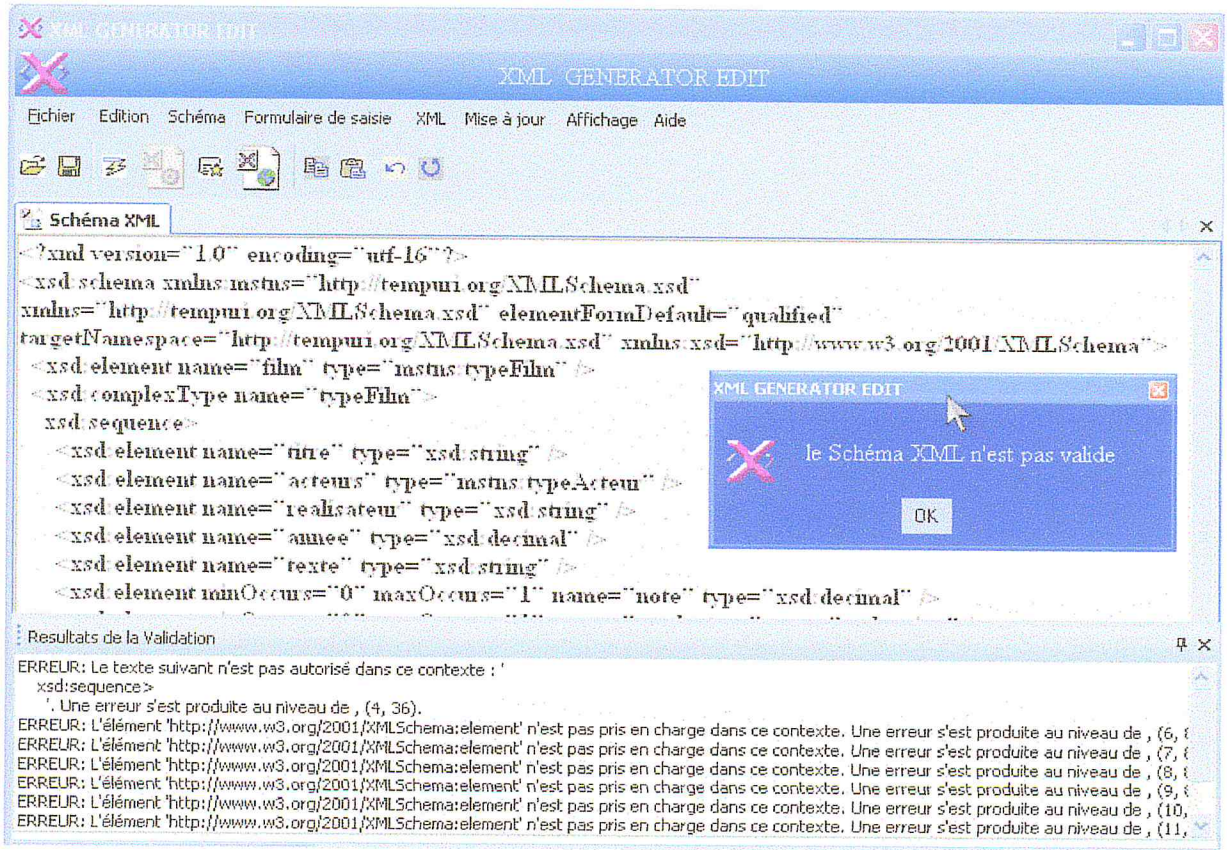


Figure 12- Analyse du Schéma XML « Ex : Schéma Film non valide»

Etape 3 : Génération du formulaire de saisie

Pour générer le formulaire, l'utilisateur doit cliquer sur l'option ***Générer le formulaire de saisie*** du menu ***Schéma.*** Une boite de dialogue comportant les propriétés du formulaire de saisie sera affichée, et permettra ainsi à l'utilisateur de personnaliser l'affichage du formulaire. Autrement notre système gardera les propriétés d'affichage par défaut.

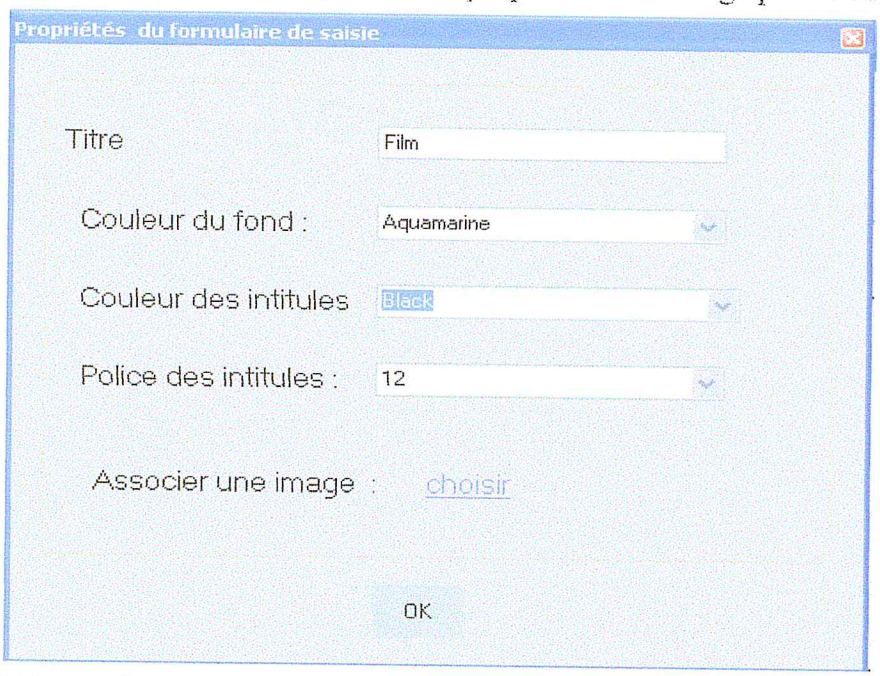


Figure 13- Personnalisation des propriétés d'affichage du formulaire de saisie« Ex : Schéma Film».

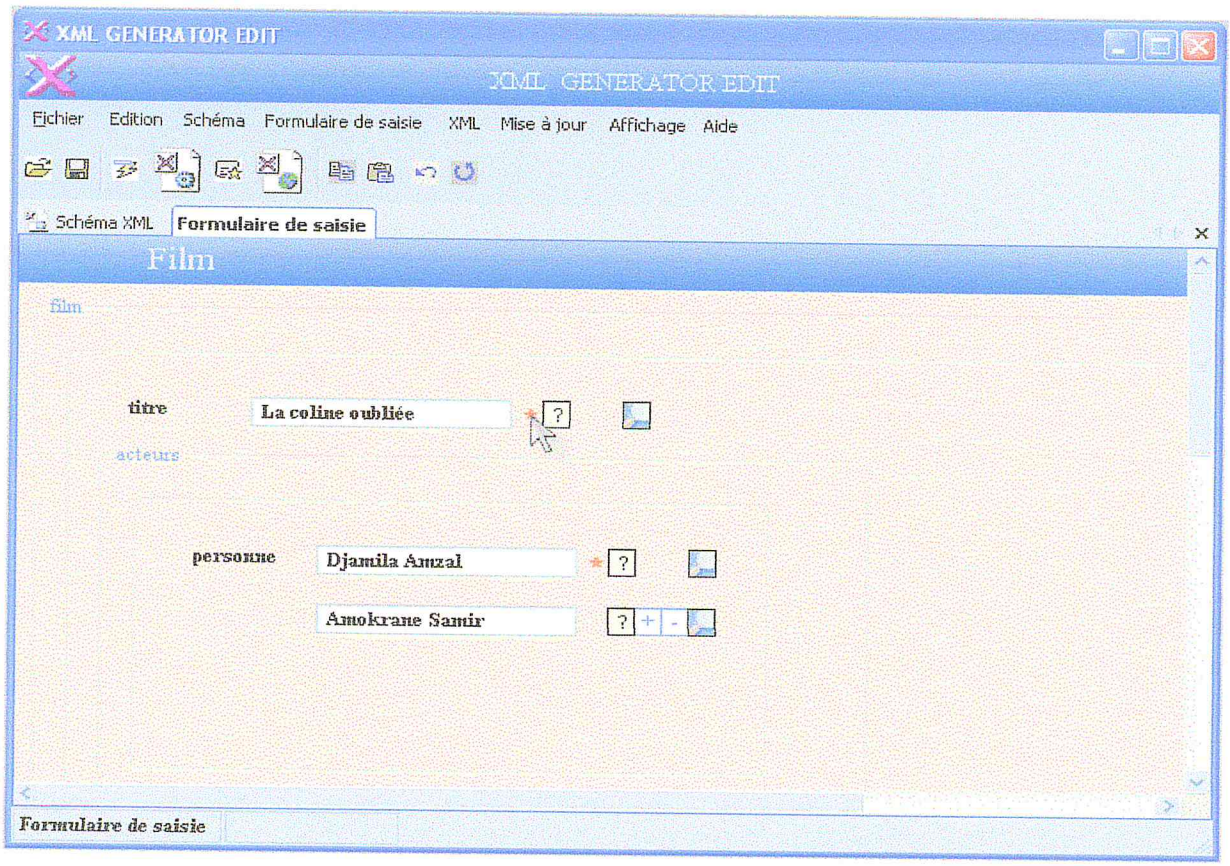


Figure 14- Affichage du formulaire de saisie « Ex : Schéma Film partie 1»

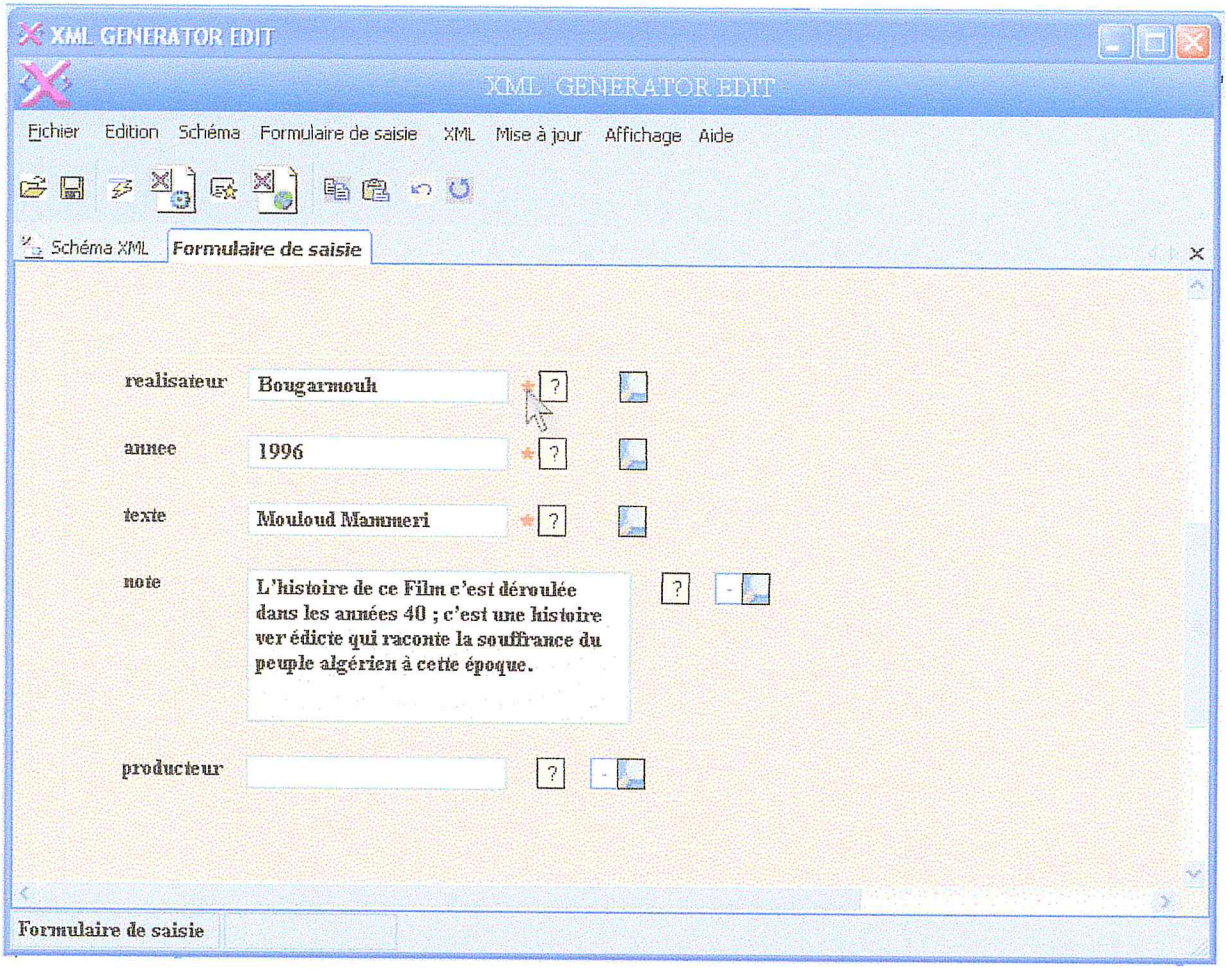


Figure 15- Affichage du formulaire de saisie « Ex : Schéma Film partie 2»

4.2 Contrôle des données saisies

4.2.1 Saisie des données

Une fois le formulaire est généré, l'utilisateur peut entamer la saisie des données. Cette dernière s'explique dans le remplissage des champs de saisie et l'activation des boutons d'options (case à cocher ou bouton radio).

Notre système met à la disposition des utilisateurs le bouton aide « ? » afin de les aider à connaître le type des données à saisir dans chaque champs, les boutons « + » et « - » comme indicateurs d'occurrences et « * » pour indiquer que le champs est obligatoire. Pour cela il suffit juste de poser le curseur sur le bouton concerné pour que le message d'aide sera affiché.

4.2.2 Les contrôles de validation

L'insertion d'une instance valide est vérifiée via les contrôles de validation imposés par notre système. Dans le cas, ou la saisie n'est pad valide, des boites de dialogues explicatives seront affichées à l'utilisateur. Ils contiennent une description complète de l'état de la validation. XML_GENERATOR_EDIT signale les erreurs selon les types des contrôles. Les points suivants montrent dans quel cas de saisie non valide le système informe l'utilisateur :

4.2.2.1. Contrôle sur les champs obligatoires :

L'utilisateur devra remplir tous champs marqués par une étoile rouge, faut de quoi un message d'erreurs sera affiché.

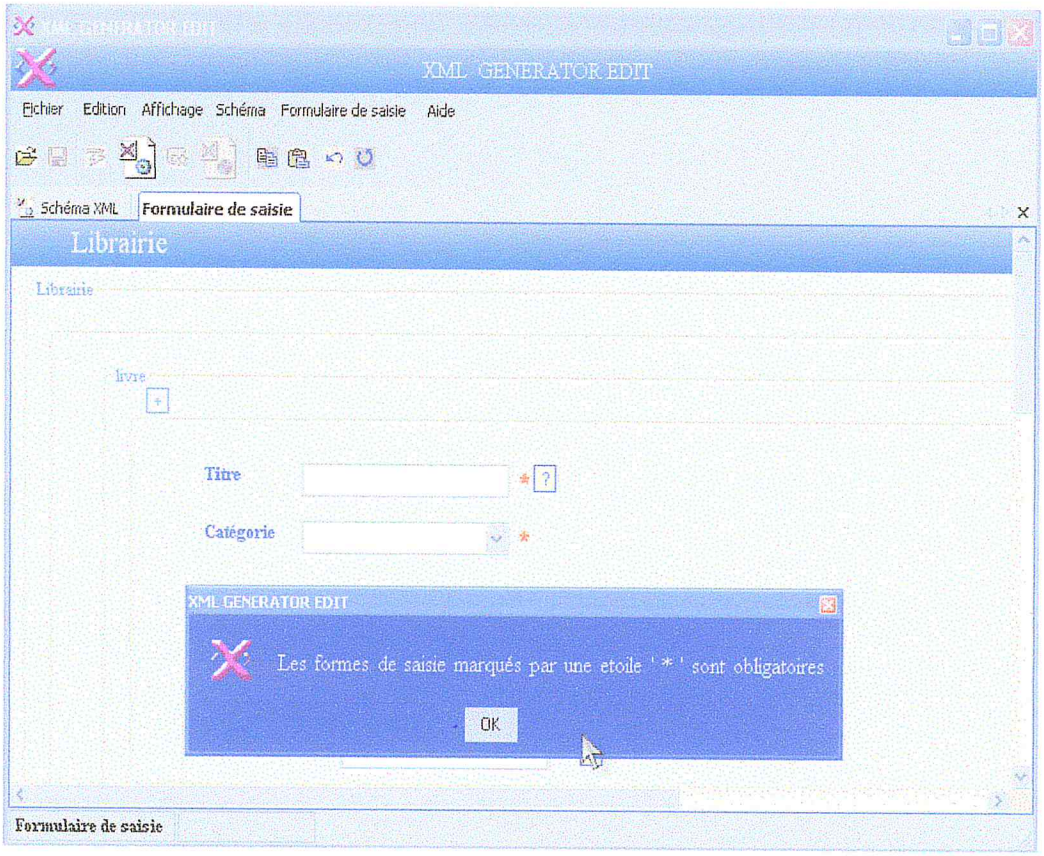


Figure 16- Contrôle sur champs obligatoire « Ex :Schéma Librairie ».

4.2.2.2. Contrôle de validité de types de données intégrés:

L'utilisateur devra remplir le formulaire suivant les types de données exprimés dans le schéma associé. Si non un message d'erreur sera affiché indiquant les champs erronés, en utilisant la couleur rouge.

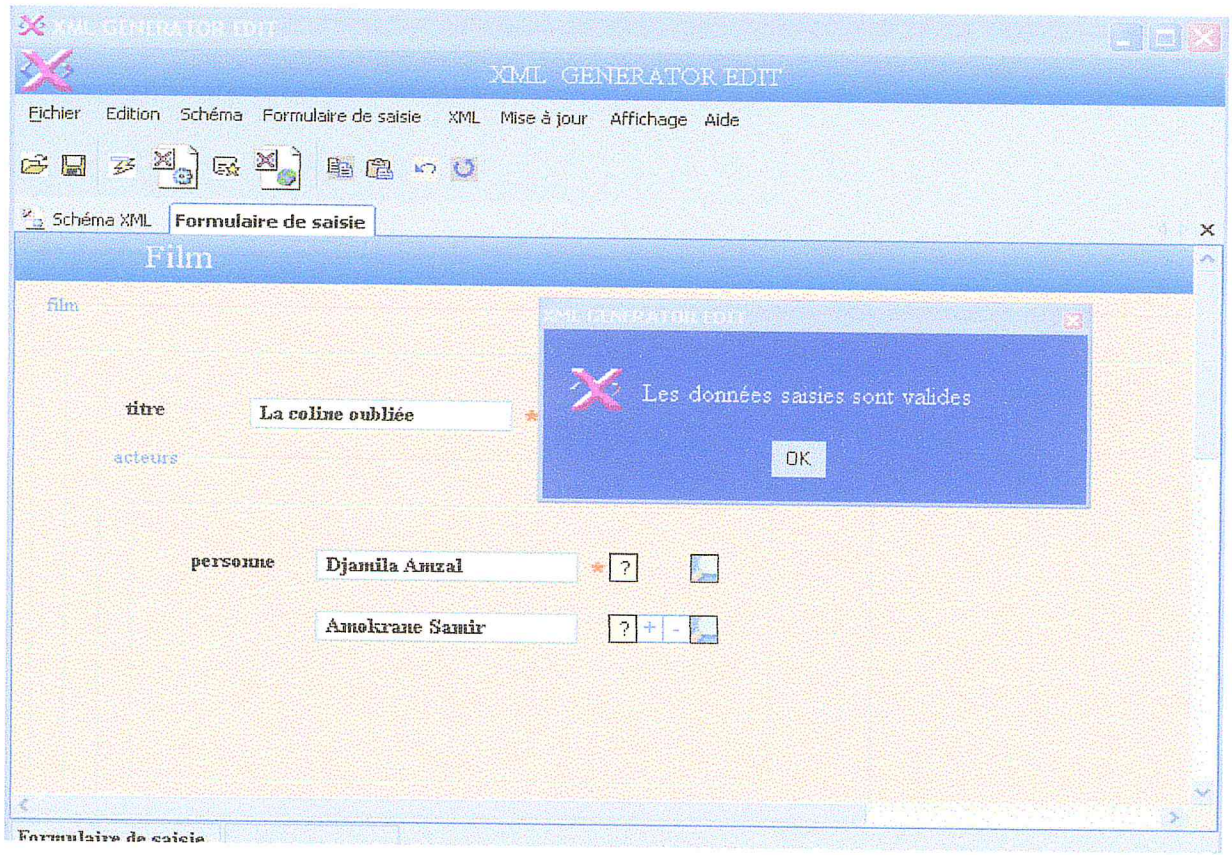


Figure 17- Affichage d'une boite de dialogue données valides « Ex : Schéma Film ».

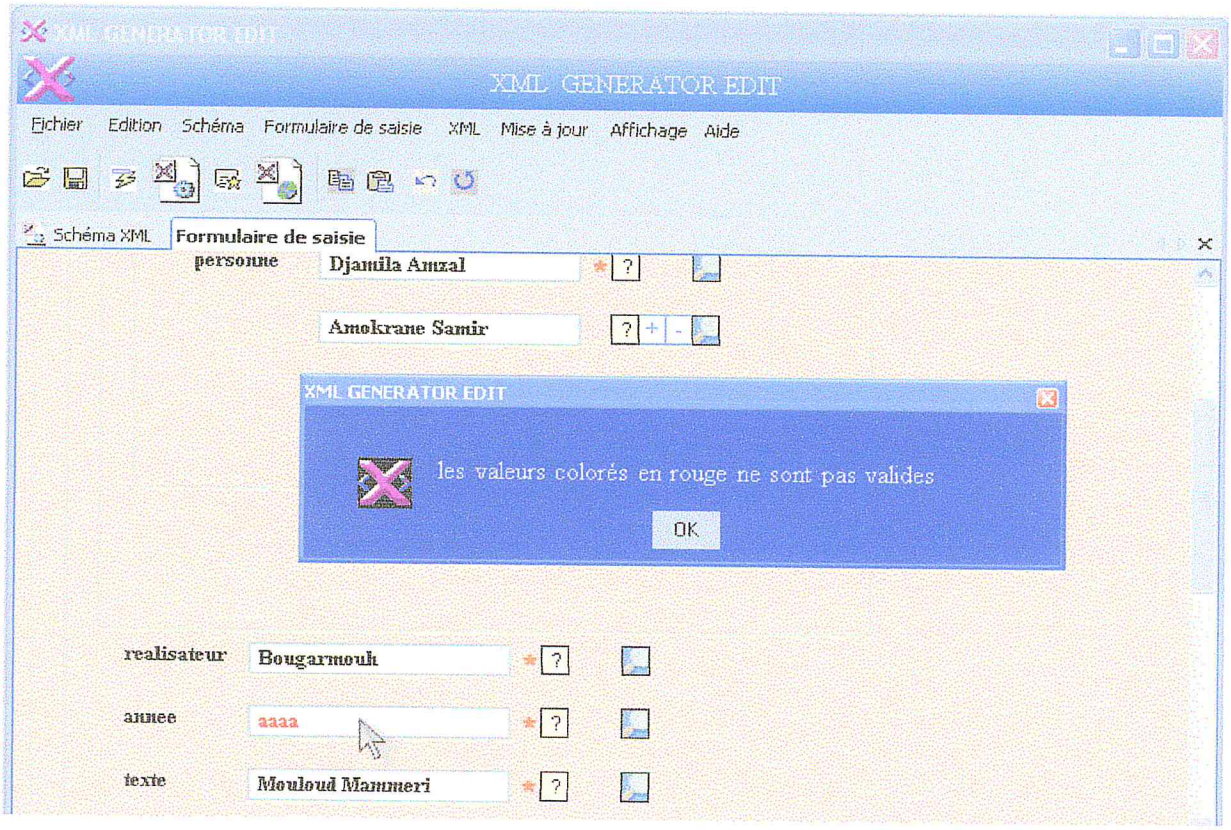


Figure 18- Affichage d'une boite de dialogue données non valides « Ex : Schéma Film ».

4.2.2.3. Contrôle de validité de types de données dérivés :

L'utilisateur devra remplir le formulaire suivant les types de données et les facettes applicables exprimés dans le schéma associé. Autrement un message d'erreur sera affiché indiquant les champs erronés, en utilisant la couleur rouge.

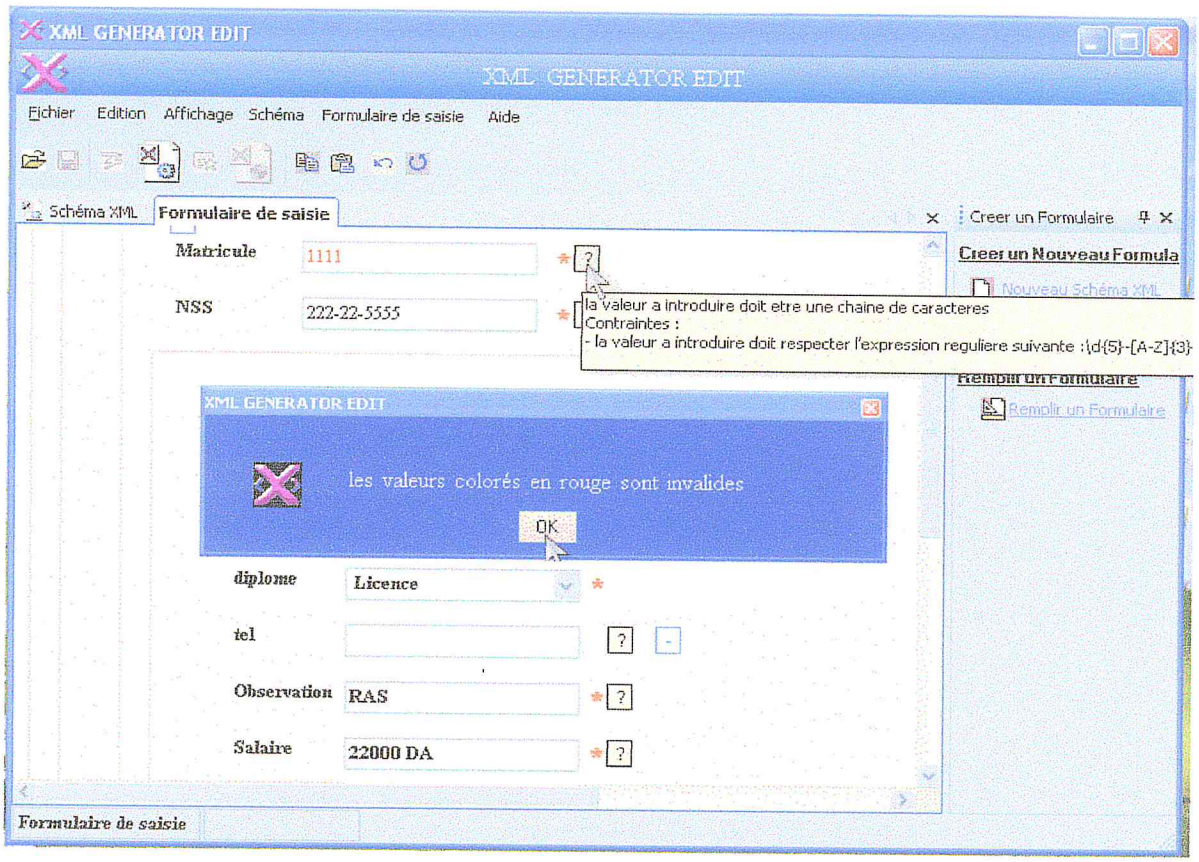


Figure 19- Contrôle de validité selon les types de données dérivés (dans ce cas facette *pattern*.).

4.2.2.4. Contrôle sur les champs de type ID :

L'utilisateur ne doit pas remplir les champs de saisies soulignés en rouge avec des valeurs semblables, si non un message d'erreur sera affiché.

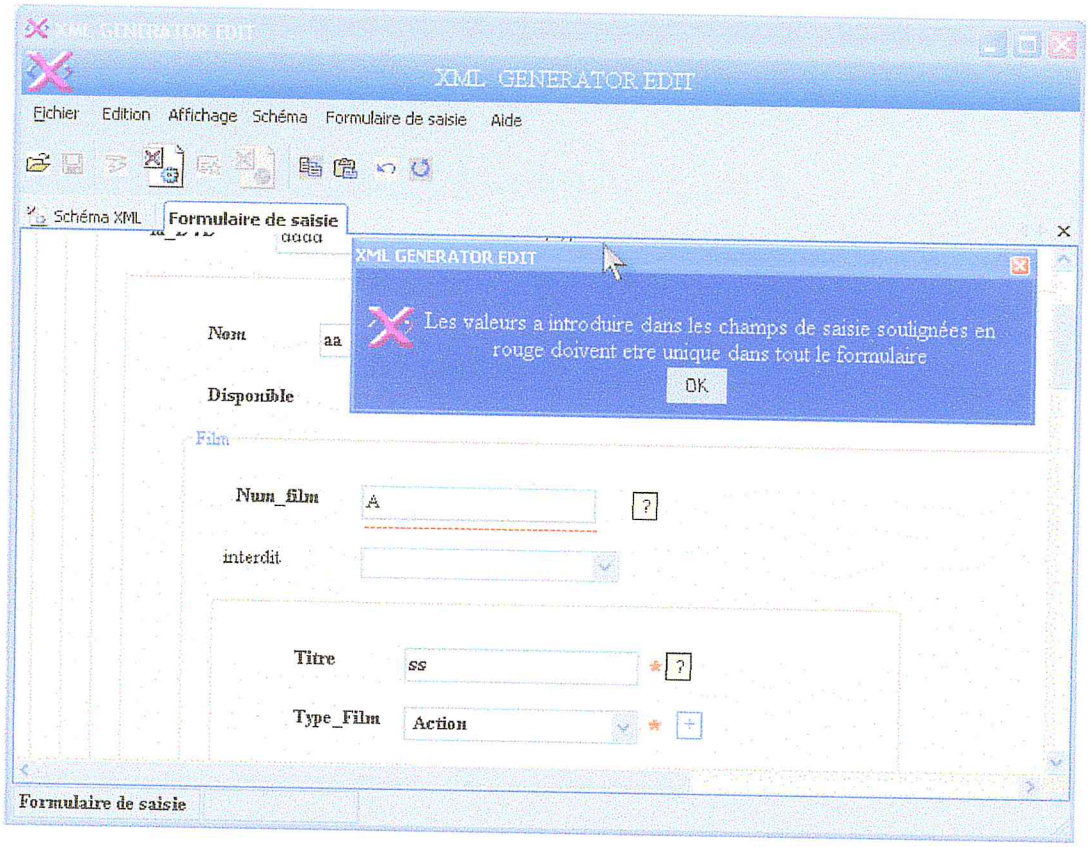


Figure 20- Contrôle de type ID.

4.2.2.5. Contrôle sur les champs de type IDREF et IDEREFS :

L'utilisateur doit remplir les champs de saisies soulignés en bleu avec des valeurs qui correspondent à celles des champs de saisies soulignés en rouge, autrement un message d'erreur sera affiché.

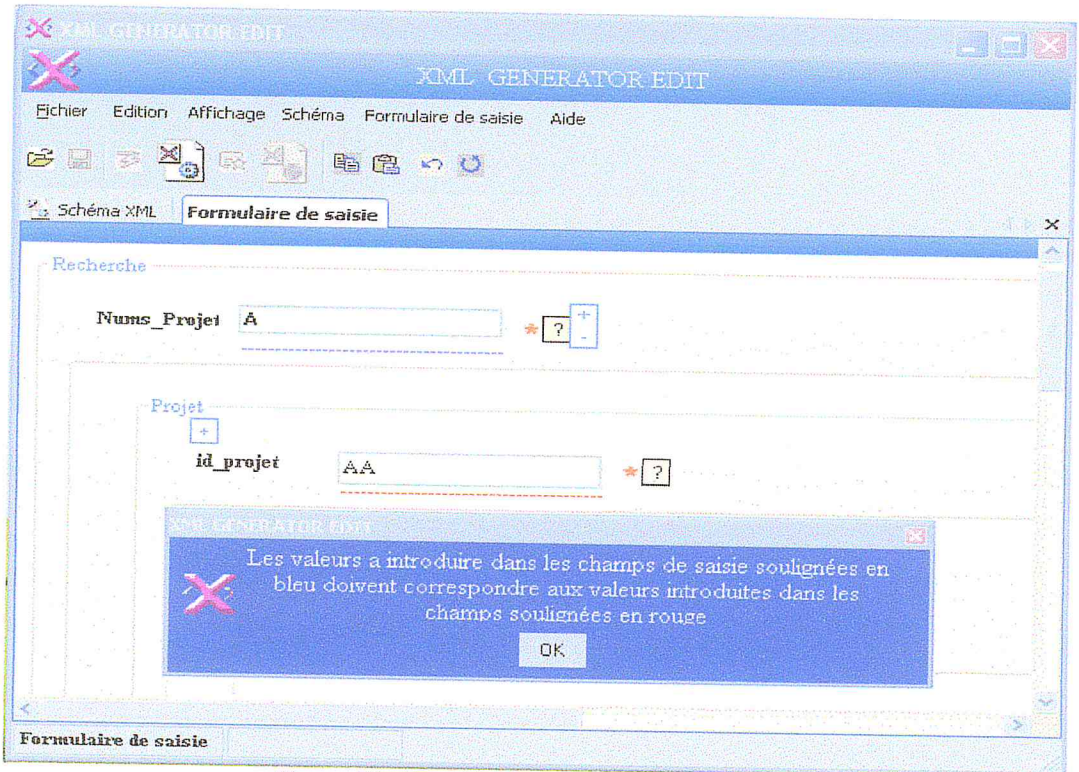


Figure 21- Contrôle de type IDREF et IDREFS.

4.2.2.6. Contrôle sur les champs de type Key:

Les jeux de valeurs destinés à être les clés d'un élément ne doivent pas être nuls, et doivent être unique, faut de quoi il y aura un message d'erreur.

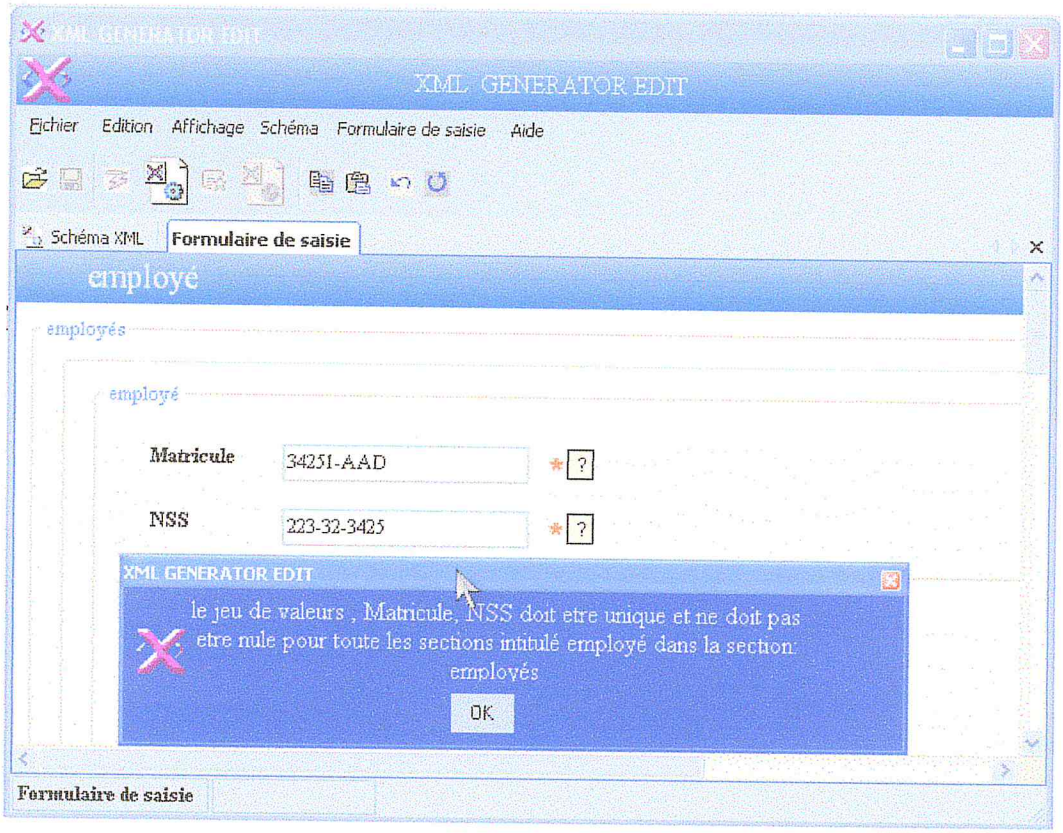


Figure 22- Contrôle de type Key.

4.2.2.7. Contrôle sur les champs de type unique :

Les jeux de valeurs destinés à être des clés uniques d'un élément doivent être unique, autrement il y aura un message d'erreur.

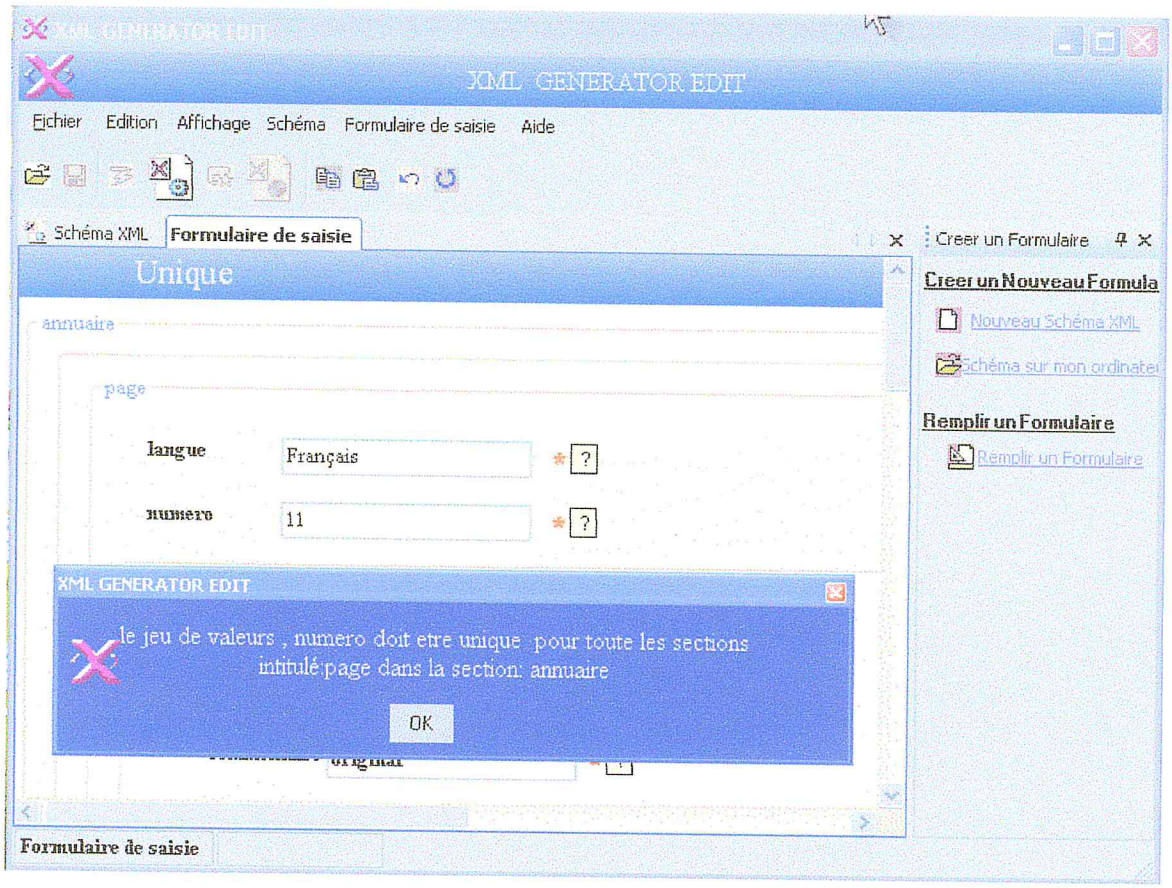


Figure 23- Contrôle de type unique.

4.3 La Génération d'un document XML Valide

Afin de générer le document XML, XML_GENERATOR_EDIT récupère toutes les données introduites du fichier formulaire renseigné et la hiérarchie du fichier modèle.
Pour générer le document XML, l'utilisateur doit cliquer sur l'option ***Générer le document XML*** du menu ***Formulaire de saisie.*** Une boite de dialogue comportant les propriétés du document XML est affichée, et permettra ainsi à l'utilisateur soit générer selon ces besoins le codage des caractères, la référence à une feuille de style et le nom et l'emplacement du formulaire renseigné avec l'extension « xge » . Autrement notre système gardera les propriétés d'affichage par défaut. Ensuite c'est la boite de dialogue permettant d'enregistrer le document XML avec l'extension « xml »qui sera affichée pour indiquer ou sera enregistré le fichier XML généré. Notre système permet deux types d'affichages, mode texte et mode Web pour mieux voir la représentation de la structure du document XML final on verra dans ce qui suit un exemple explicatif.

Au niveau de la génération du document XML, les caractères «< », « > », « ' », « " »et « & » sont remplacés par leurs valeurs prédéfinies par notre système.

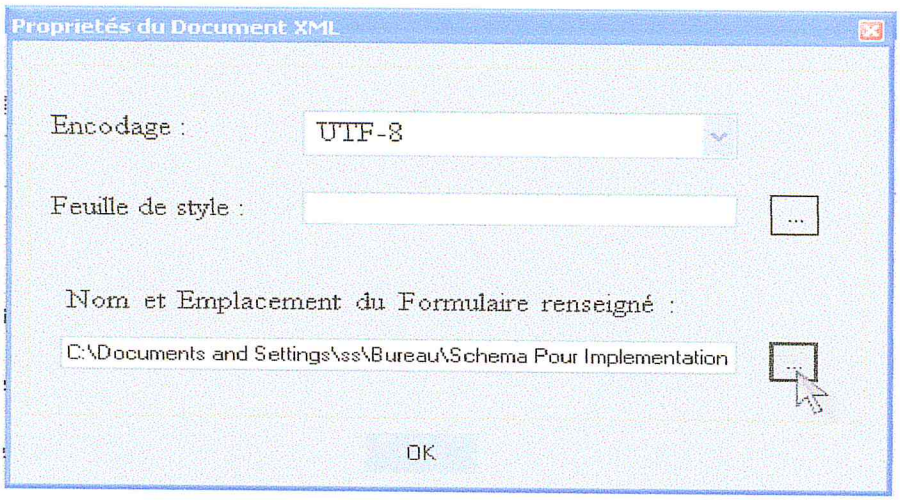


Figure 24- Propriétés d'affichage du document XML.

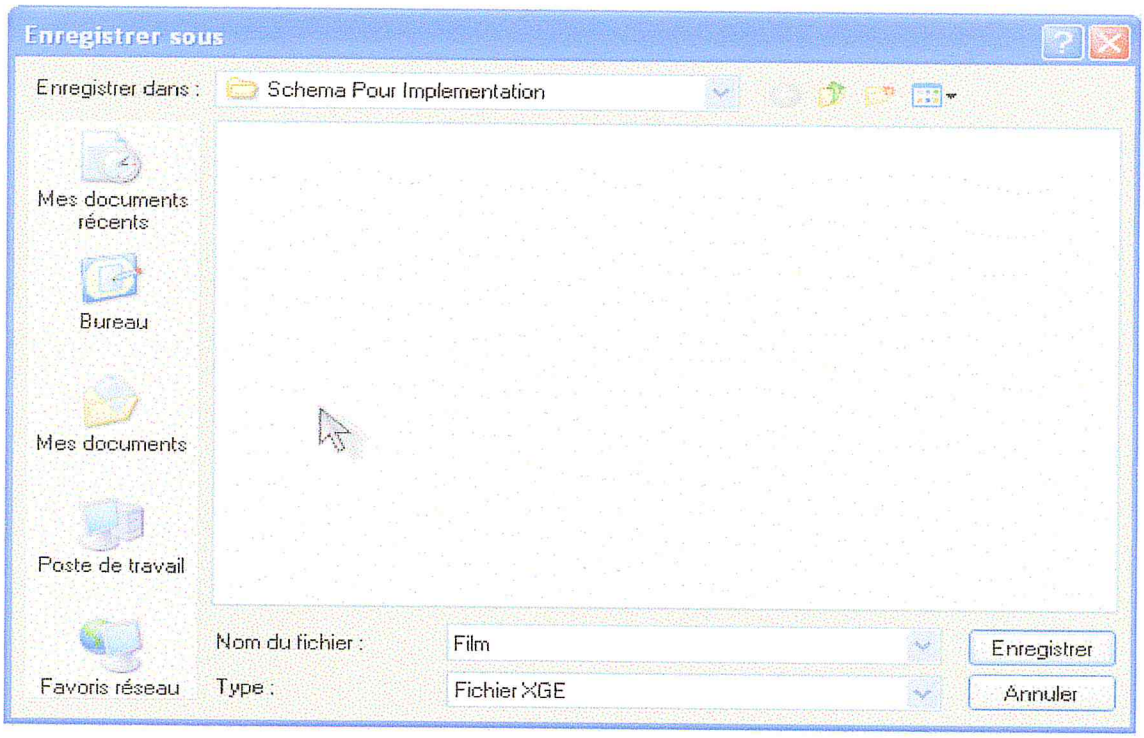


Figure 25- Enregistrement du formulaire renseigné.

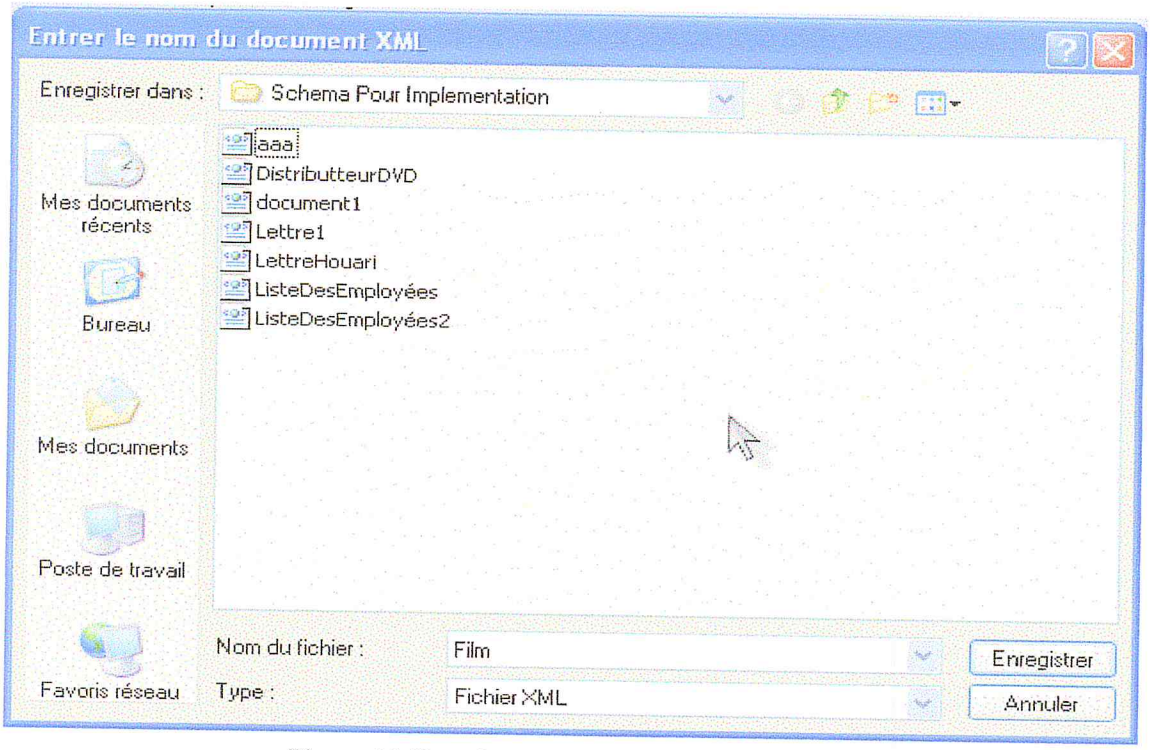


Figure 26- Enregistrement du formulaire renseigné.

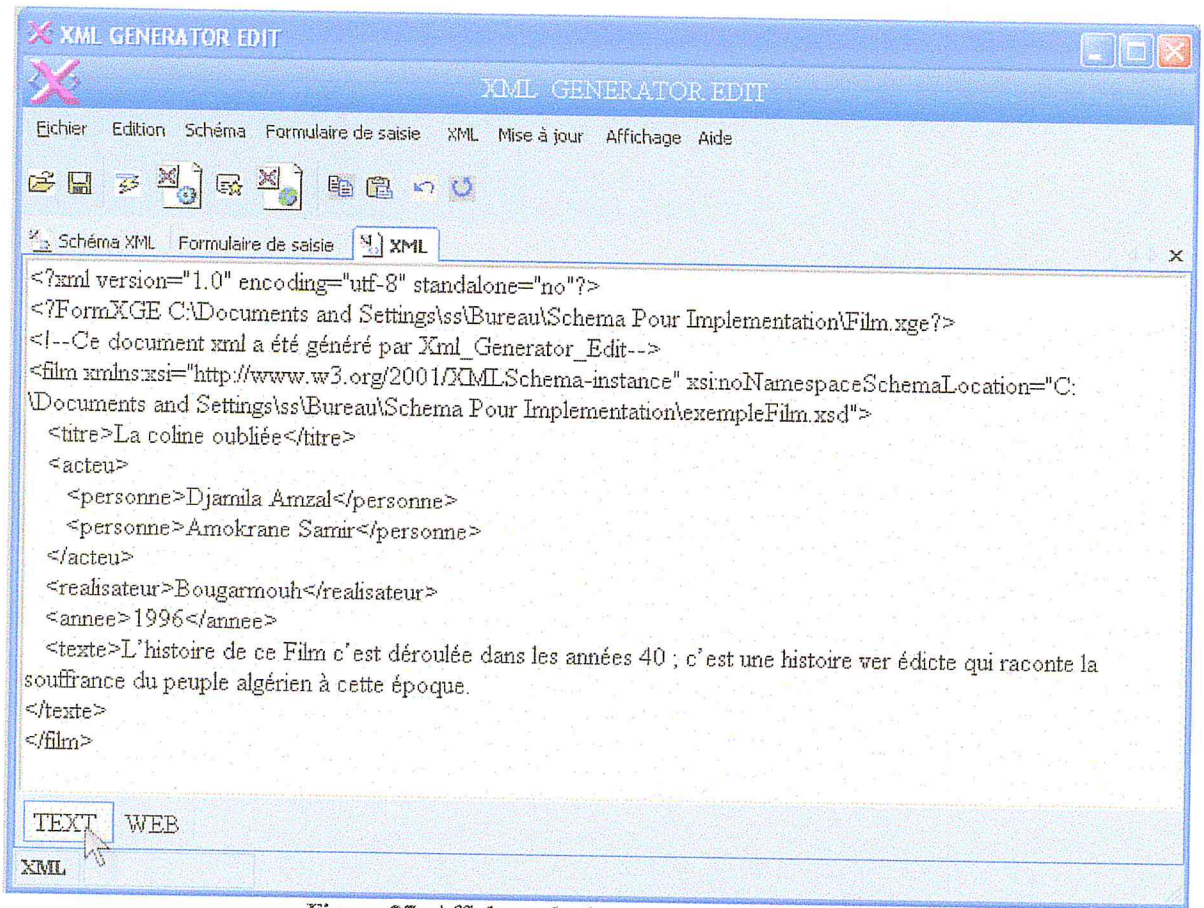


Figure 27- Affichage du document XML en mode texte.

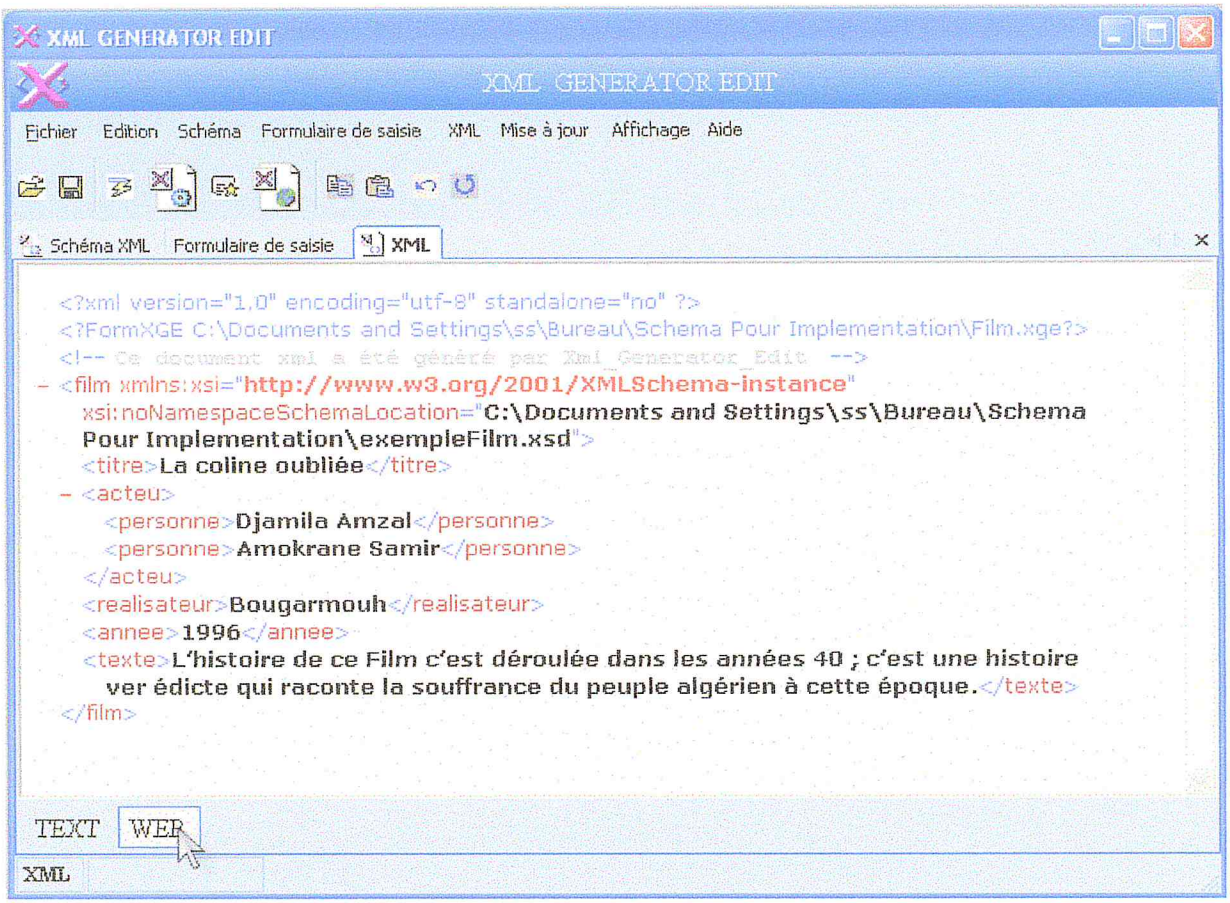


Figure 28- Affichage du document XML en mode Web.

4.4. Les opérations de mise à jour

Notre système permet à l'utilisateur aimerait effectuer des mises à jour sur la structure de son document XML via le formulaire de saisie associé, et cela en cliquant avec le bouton droit de la souris pour qu'un menu comportant :

- Modifier l'intitulé.
- Modifier le type de données.
- Ajouter un élément de type simple ou un attribut selon l'emplacement du champ de saisie, qui comporte à son tour au dessous ou au dessus pour donner l'emplacement exact du nouvel élément
- Supprimer un élément ou un attribut.
- Modifier l'occurrence qui comporte champs optionnel ou obligatoire.
- Modifier les commentaires associés aux champs de saisie.

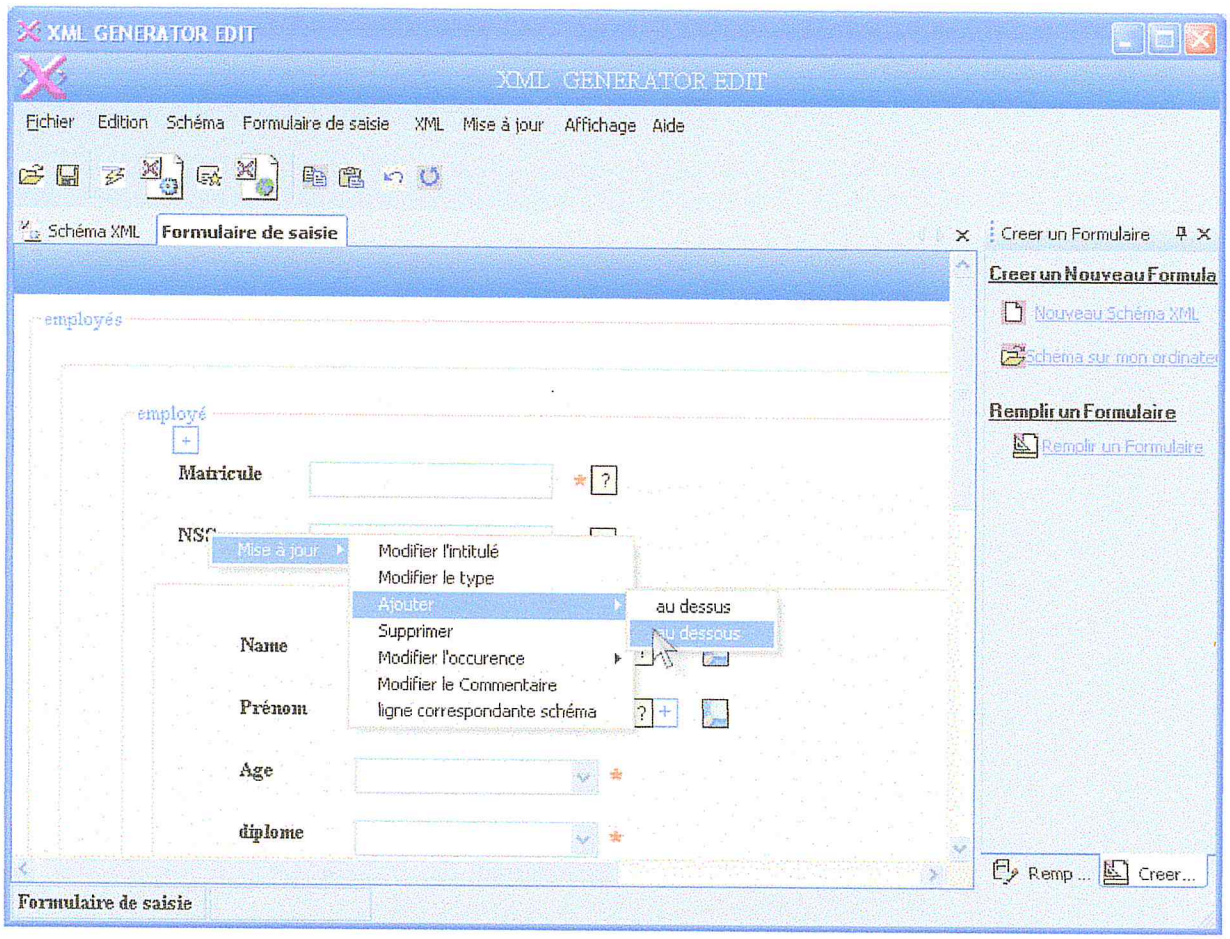


Figure 29- Opération de mise à jours (ajout d'un attribut).

Une fois que l'utilisateur a sélectionné l'opération à effectuer un boite de dialogue sera affichée pour indiquer les changements à faire.

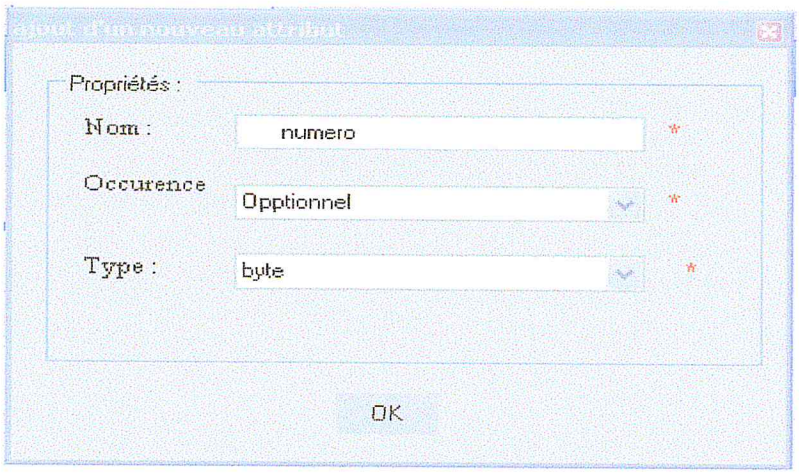


Figure 30- Propriétés du nouvel attribut à ajouter.

Les documents XML qui correspondent aux différents traitements de mise à jour seront générés à la suite de chaque formulaire mis à jours.

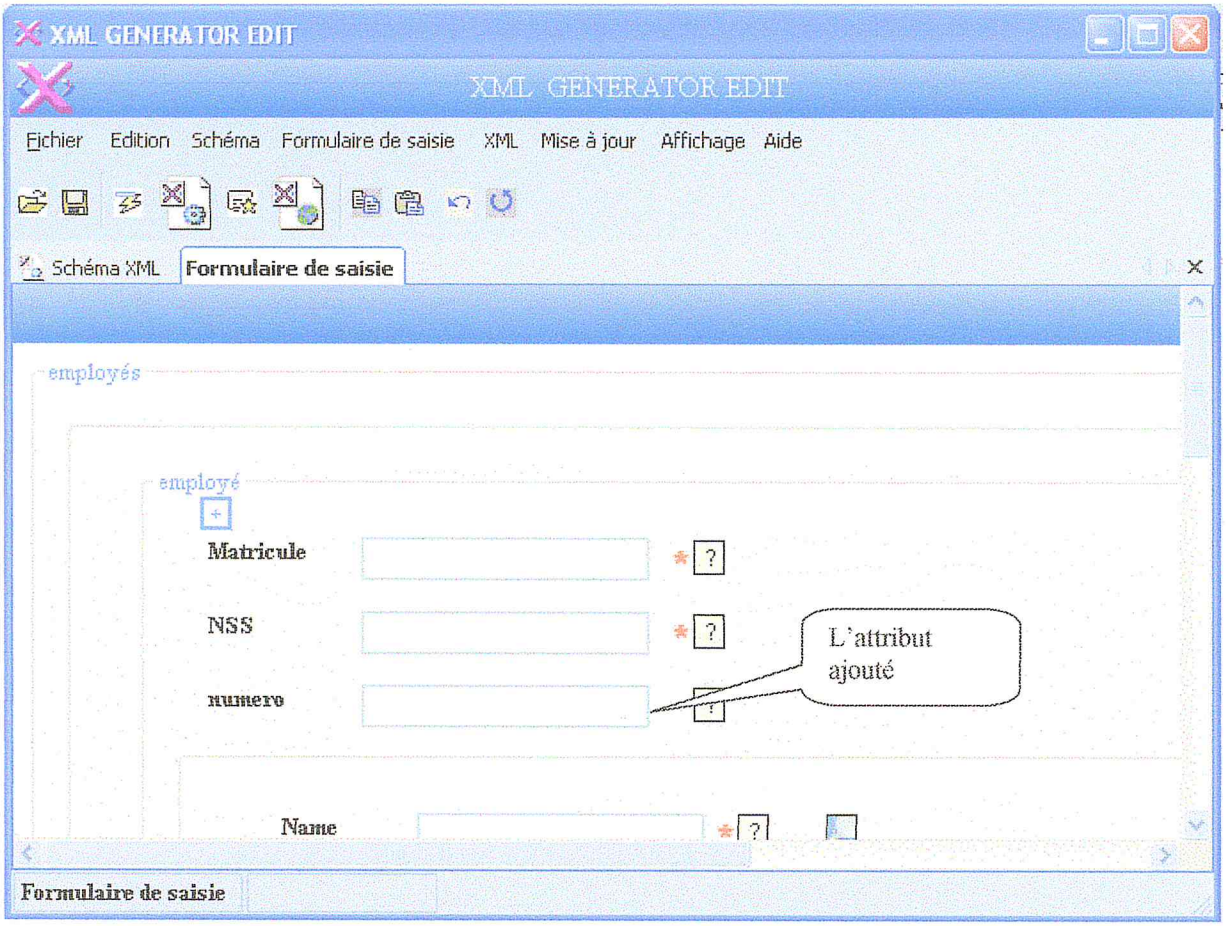


Figure 31- Modification du formulaire de saisie avec l'ajout d'un nouvel attribut à ajouter.

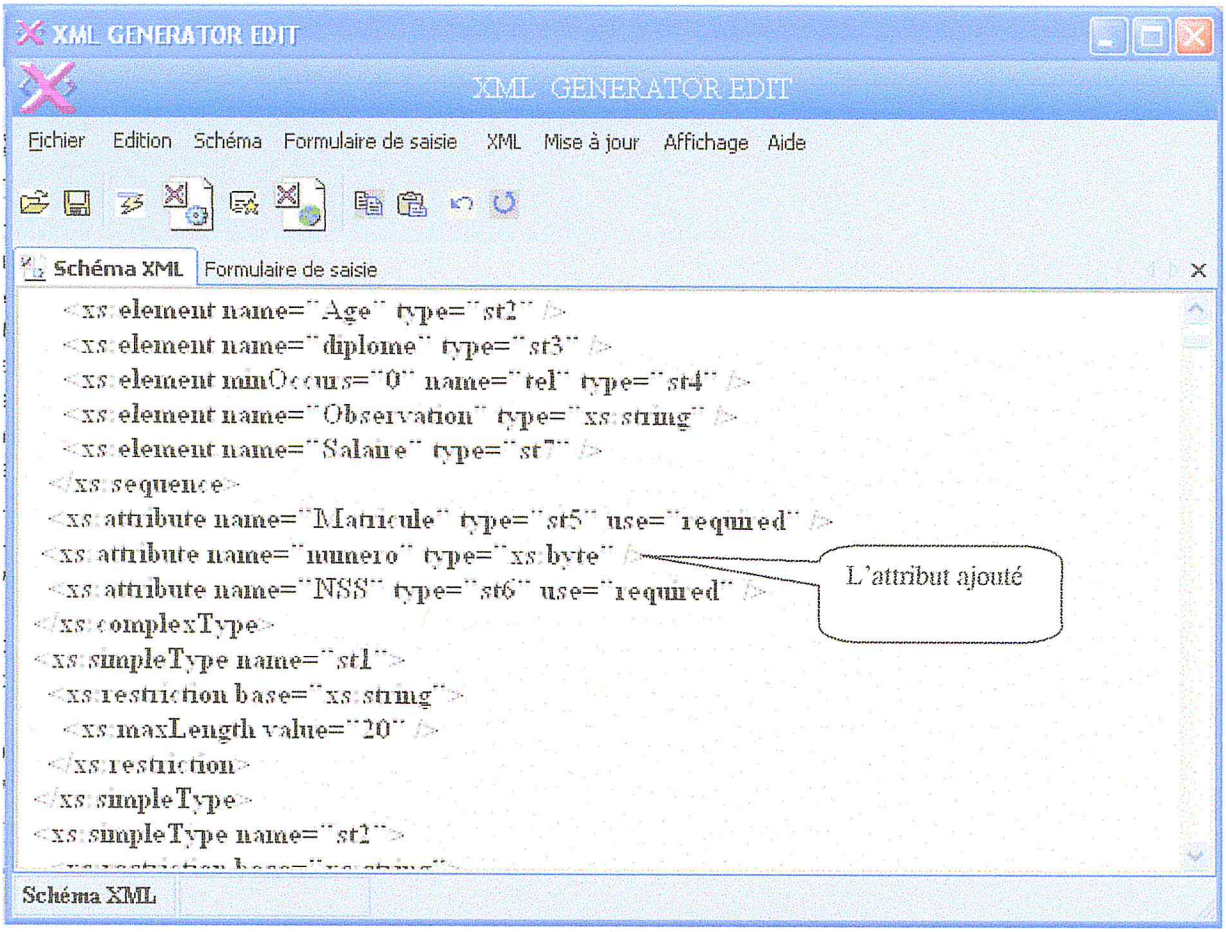


Figure 32- Modification du document XML associé.

4.5 Adaptation du formulaire

XML_GENERATOT_EDIT a mis à la disposition des utilisateurs l'option de personnaliser le formulaire de saisie du menu formulaire de saisie avant et après sa génération. Cette option permet de modifier : Le titre, la couleur, du fond, la couleur des intitulés et la taille leurs polices de caractères.
Dans ce qui suit nous allons présenter un exemple de personnalisation pour le formulaire Film

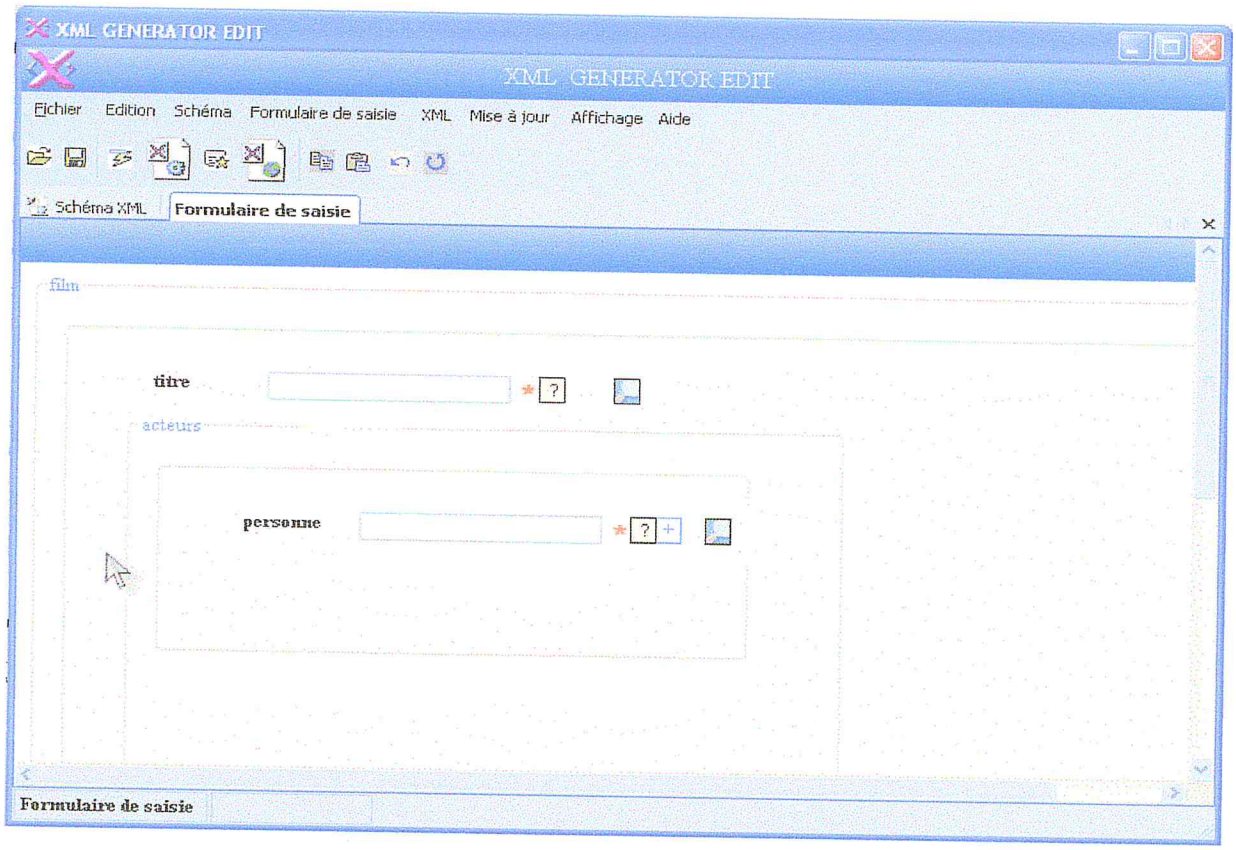


Figure33 : Formulaire de saisie avant personnalisation

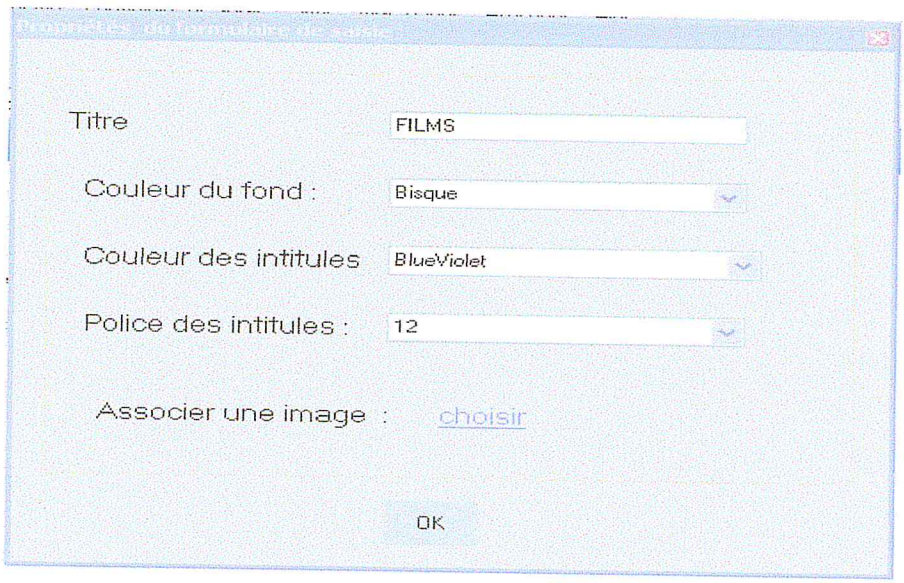


Figure 34- Personnalisation du formulaire de saisie.

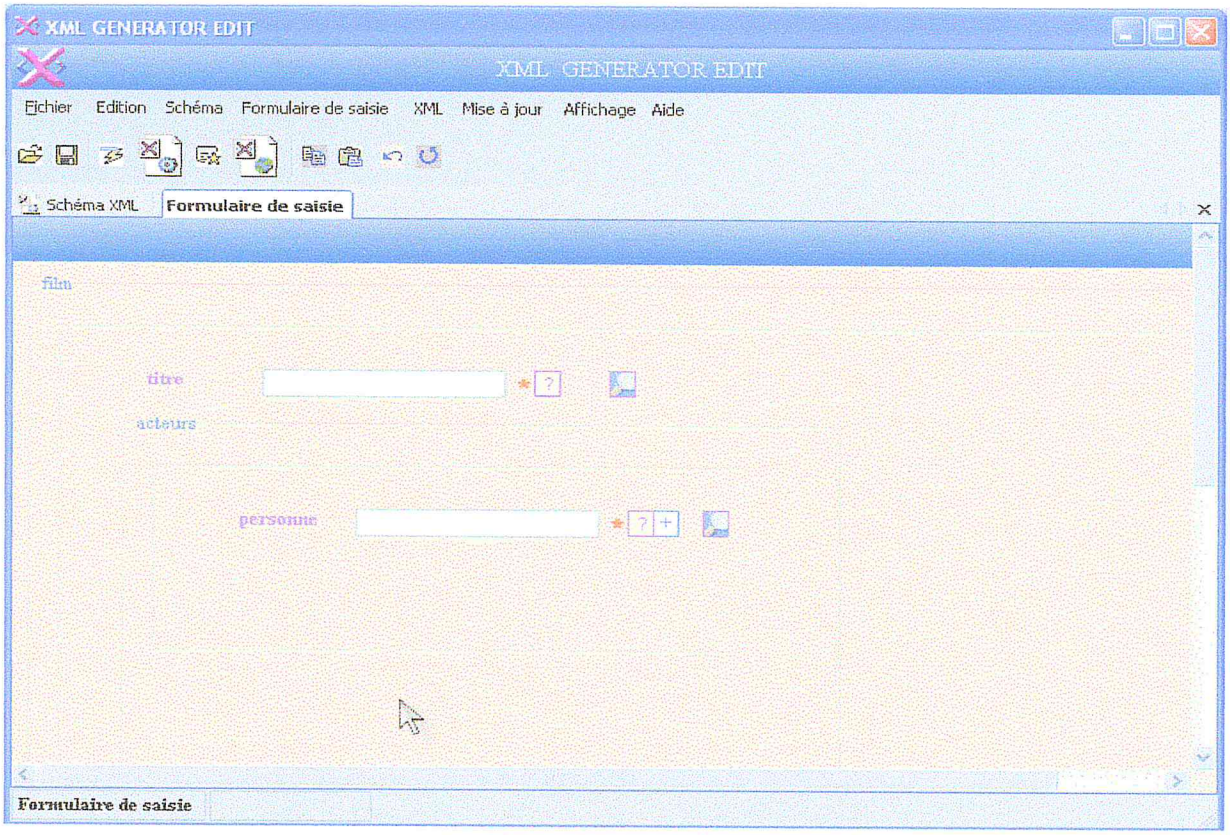


Figure35 : Formulaire de saisie après personnalisation.

En plus des personnalisations prés cités notre système offre à l'utilisateur la possibilité d'adapter la taille des champs à ses propres besoins. Pour cela il doit cliquer sur le bouton zoom pour qu'une boite de dialogue comportant la hauteur et la largeur du champ s'affichera, il lui suffira juste de modifier les deux valeurs pour que la taille du champ correspondant changera.

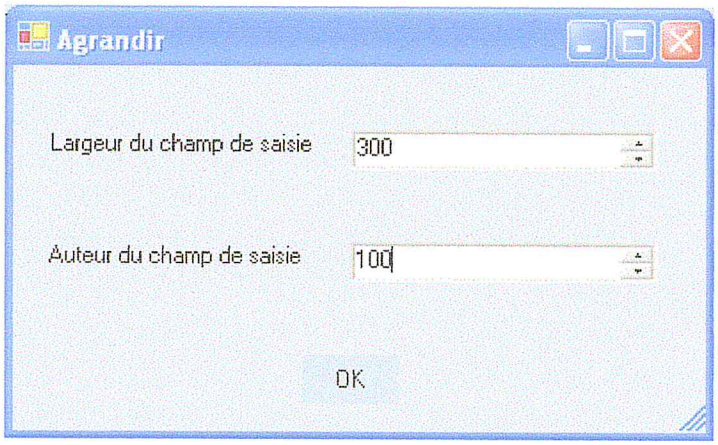


Figure36 : La boite de dialogue agrandir champ de saisie.

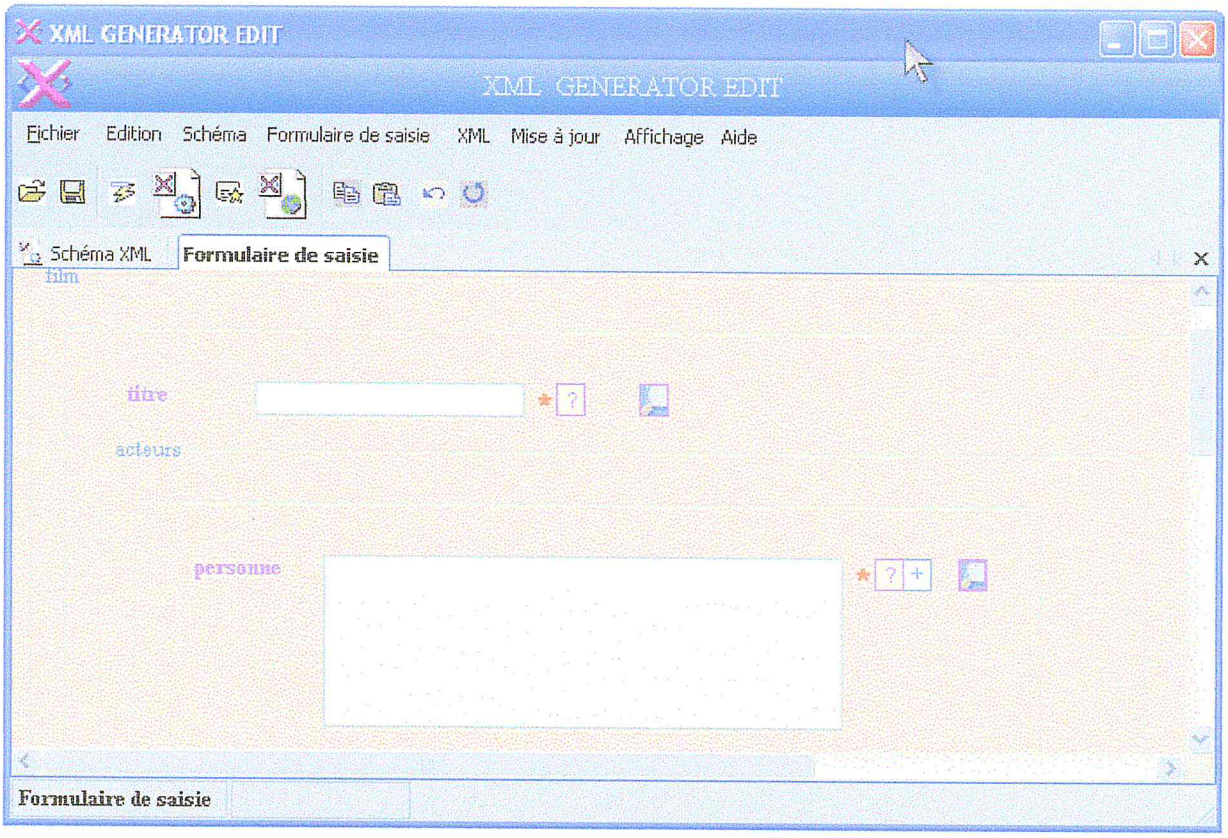


Figure37 : Formulaire de saisie après agrandissement du champ de saisie personne.

4.6 Réutilisation d'un formulaire

L'utilisateur peut ouvrir un formulaire existant et cela en activant la boite ***Remplir un formulaire*** dans le menu ***Fichier*** de l'interface du système. Il aura deux option : Ouvrir un formulaire existant déjà sur ordinateur ou ouvrir un modèle de formulaire. Une boite de dialogue ouvrir s'affichera pour spécifier le formulaire à ouvrir avec l'extension « xge » définie par défaut.

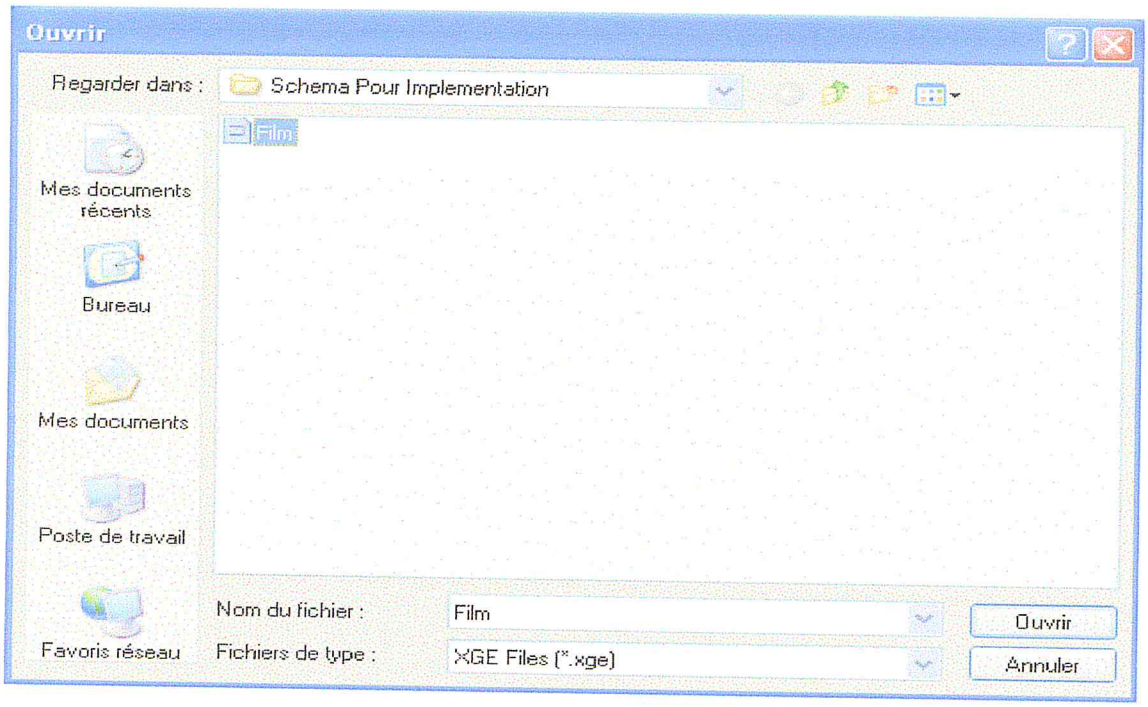


Figure37- Spécification du formulaire de saisie à ouvrir.

Dés que l'utilisateur valide son choix, le formulaire sera ouvert avec toutes les saisies introduites précédemment s'il y en a.

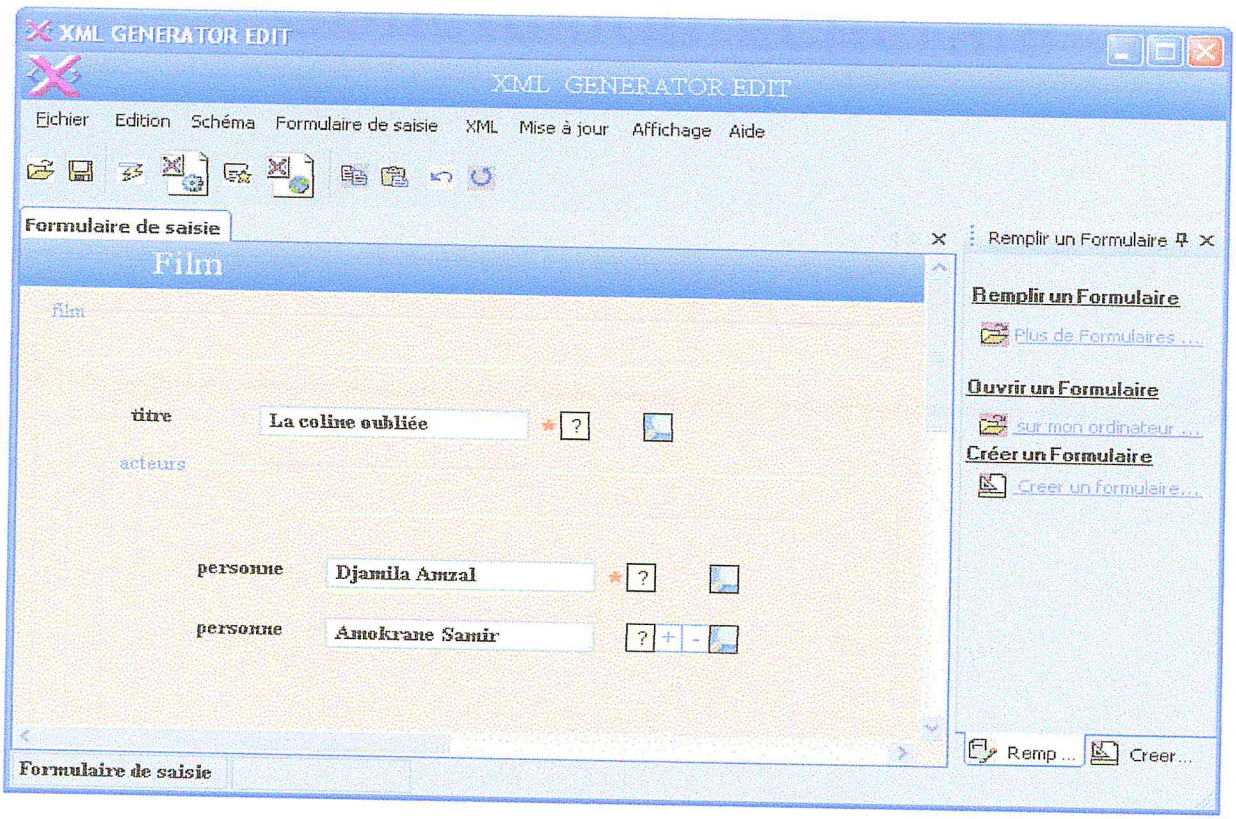


Figure37- Ouverture d'un formulaire existant

L'objectif du travail a été la réalisation d'un système capable d'offrir un mécanismes de production de documents XML Schéma valide sans contrainte de profil et à moindre coût, contrairement à ce qui existe actuellement comme éditeur XML. Après une étude détaillée de tous les concepts qui ont intervenus dans la conception du système (XML, DTD, schémas, formulaire) nous avons mis en place le système XML_GENERATOR_EDIT .

C'est un système assez complet. Il assure une bonne intégration des deux fonctionnalités d'Analyse et de validation. Il répond aux exigences et aux espérances qui ont été portées sur sa faisabilité. Il offre un certain nombre de fonctions :

- L'édition des Schémas XML.
- La génération des formulaires.
- L'adaptation des formulaires.
- La réutilisation des formulaires.
- les opérations de mise à jour de la structure des documents XML.
- La mise à jour des instances introduites.
- La génération de documents XML .

Bien que notre système ai pu répondre aux objectifs attendus, il serait tout de même intéressant de lui envisager quelques extensions. Pour cela, on a penser en premier lieu à la création d'interfaces dynamiques qui rendront l'écriture des documents XML aussi simple en offrant à l'utilisateur la possibilité de choisir les formes de saisies et de les adaptées selon ces besoins .

Deuxièmement, et en partant des documents XML valides, il faut adopterai le processus inverse afin de pouvoir générer des schémas XML valides qui seront utilisés a leurs tours pour la production d'autres instances du document XML. Mais il faut noter que ces extensions doivent obéir à l'objectif principal qui est la production de document XML valides par construction.

Nous espérons que ce travail a pu satisfaire les besoins qui ont mené à son élaboration et que les utilisateurs des documents XML trouvent en XML_GENERATOR_EDIT un outil assez complet, respectant la commodité, la convivialité, la réduction des coûts et répondre ainsi à leurs exigences.

Annexes et Bibliographie

1 - Présentation de la plate-forme Microsoft .NET :

1.1- La plate-forme .NET :

Cette section présente les grandes lignes de l'architecture de la plate-forme .NET. La plate forme .NET se compose de plusieurs fonctionnalités et services de la base, comme l'illustre la figure A.1, l'un des objectifs de cette nouvelle plate-forme est de simplifier le développement Web.

1.2- Technologie de base de la plate-forme .NET :

Les technologies de base qui composent la plate-forme .NET sont les suivantes :

- **.NET** Framework :

  Cette technologie se fonde sur un nouveau Common Language Runtime. Celui-ci fournit un ensemble commun de services pour les projets crées avec Visuel Studio.NET, indépendamment du langage. Ces services fournissent des blocs modulaires de base pour les applications de tous types, utilisables à tous les niveaux des applications.

  Microsoft Visuel Basic, Microsoft Visuel C++, Microsoft Visuel C# et d'autres langage de programmation Microsoft ont été améliorés pour tirer profit de ces services.

- **.NET** Building Block Services :

  C'est un ensemble de services programmables distribués disponibles à le fois en ligne et hors connexion. Un service peut être appelé sur un ordinateur autonome non connecté à Internet ; il peut également être fourni par un serveur local fonctionnant au sein d'une entreprise. .NET Building Block Services peut être utilisé à partir de n'importe quelle plate-forme prenant en charge SOAP. Au nombre de ces services, citons : les calendrier, les annuaire, la notification et la messagerie…etc.

- Visuel Studio **.NET** :

  Constitue un environnement de développement de haut niveau, destiné à la création d'application sur le .NET Framwork. IL fournit des technologies clés afin de simplifier la création.

- .NET Entreprise Server :

  Les produits .NET Entreprise Server permettent une évolutivité, une fiabilité, une gestion de l'intégration. Ils offrent en outre un grand nombre de fonctionnalité par exemple : Microsoft SQL Server 2000, Microsoft Commerce Server 2000.

La figure suivante décrit les technologies de base de la plate forme Microsoft.NET

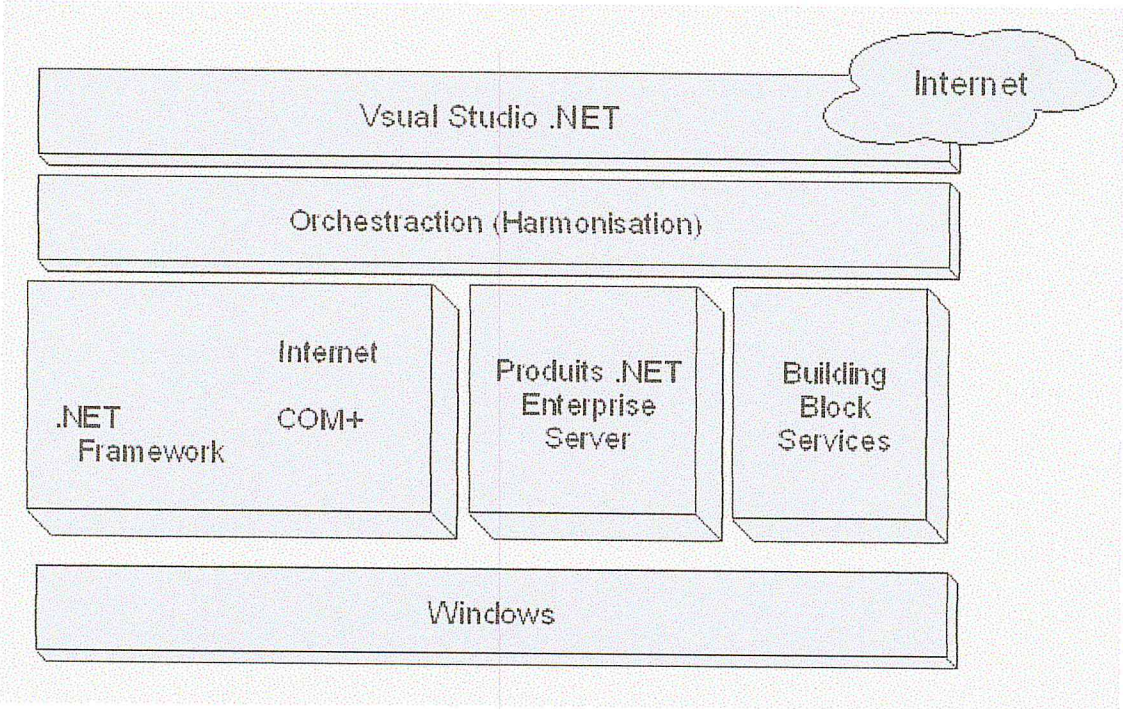


Figure A.1 Plate forme Microsoft .NET

1.3- Les avantages de la nouvelle technologie :

La plate-forme .NET offre les avantages suivants :

- Modèle de programmation cohérent et indépendant du langage, utilisable à tous les niveaux d'une application ;
- Interobabilité parfaite entre technologies ;
- Migration aisée à partir des technologies existantes ;
- Prise en charge totale des technologies Internet fondées sur des standards et indépendantes des plate-forme, telles que http, XML et SOAP.
- Grâce au Common Language Runtime, tous les langages compatibles avec la plate-forme .NET vont utiliser les mêmes fichiers d'exécution. Il n'est donc plus nécessaire de distribuer des bibliothèques d'exécution spécifiques à un seul langage, parce que les fichiers d'exécution .NET seront installés automatiquement dans les versions de Microsoft.

2 -Présentation du .NET Framework :

Le .NET Framework fournit tous les services communs nécessaire pour l'exécution de nos applications ; Ces services sont disponibles dans tous les langages compatibles avec .NET grâce à la spécification CLS (Common Language Specification)

Cette figure décrit ces services :

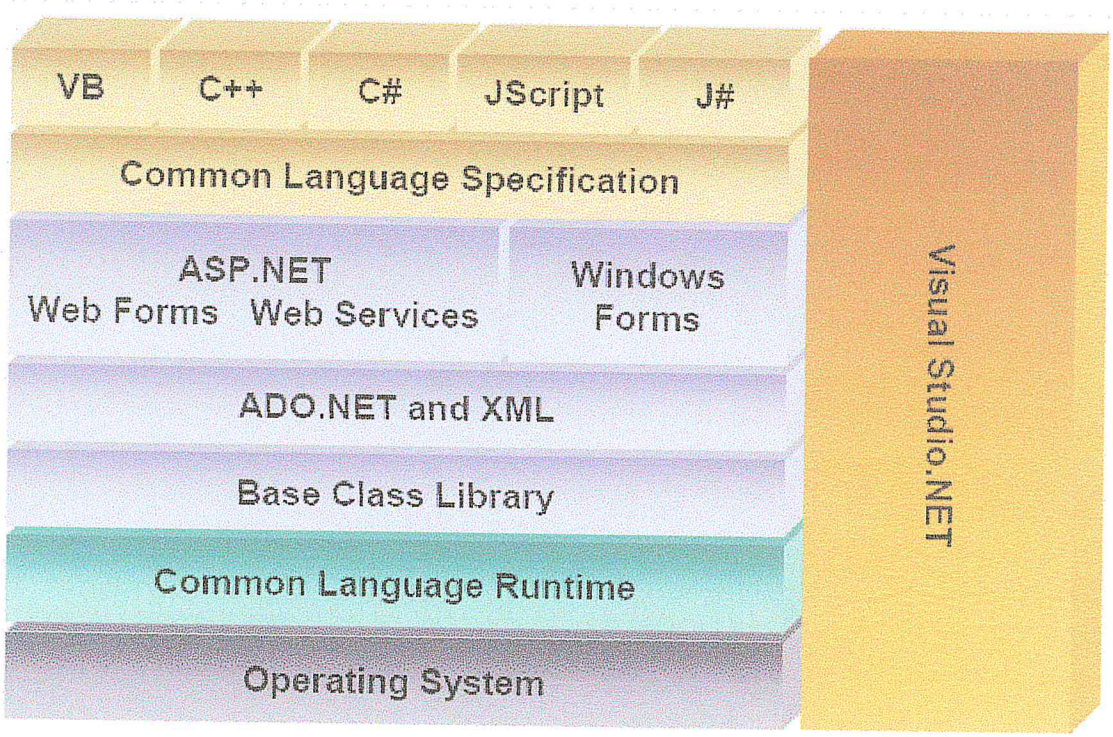


Figure A.2 Services communs pour l'exécution des applications

2.1 - Création de composants dans le .NET Framework :

Avant l'avènement de COM, les applications étaient des entités totalement séparées, sans aucune intégration ou très peu. Grâce à COM, nous pouvons intégrer des composants au sein d'une même application et à plusieurs, en exposant des interfaces communes. Dans le .NET Framework, les composants possèdent une base commune. Il n'est plus nécessaire d'écrire le code visant à permettre aux objets d'interagir directement les uns avec les autres. Dans cet environnement le .NET Framework prend totalement en charge les classes, l'héritage, les méthodes, les propriétés, le polymorphisme, les constructeur et d'autres construction orientées objet.

2.2- Spécification CLS (Common Language Specification) :

La spécification CLS définit les standards communs que doivent respecter les langage et les développeurs pour que leurs composants et applications puissent être largement utilisés par d'autres langages compatibles avec le modèle .NET . Elle permet aux développeurs Visuel Basic .NET, Visuel C++ ou d'autres langages de créer des applications dans le cadre d'une équipe multi-langage, avec l'assurance que l'intégration des différents langages s'effectuera sans problème. CLS permet même aux développeurs Visuel Basic .NET ou Visuel C++ …etc d'hériter de classes définies dans des langage différents.

2.3 Visuel Studio .NET:

Dans le .NET Framework, Visuel Studio .NET fournit les outils servant au développement rapide d'applications.

2.4 Les Langages du .NET Framework :

Cette section présente les langages que Microsoft fournit avec Visuel Studio .NET pour le .NET Framework.

- **Visuel Basic .NET :** Nouvelle version de Visuel Basic avec des innovations substantielles en terme de langage.
- **C#:** Il s'agit du premier langage moderne orienté composant.
- **Extensions C++:** offre plus de puissance et de contrôle.
- **J# .NET :** est un langage pour les développeurs java qui souhaitent créer des applications et des services pour le .NET .
- **Langages tiers :** divers langage tiers prennent en charge le .NET : COBOL, Pascal, SmallTalk… etc.

3 Présentation des composants .NET Framework :

Les composants du .NET Framework sont les suivants :

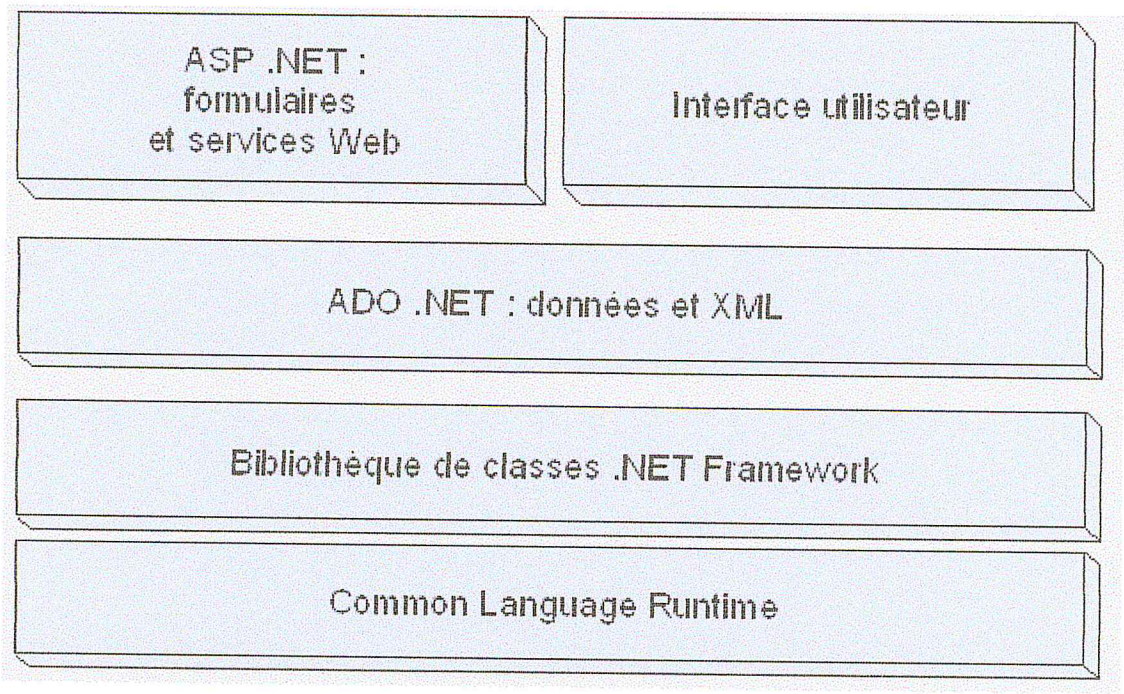


Figure A.3 Composants du .NET Framework

3.1- Common Langage Runtime :

Il simplifie le développement d'application, fournit un environnement d'exécution robuste et sécurisé, prend en charge plusieurs langages, simplifie le déploiement et la gestion des applications et offre un environnement géré

3.1.1- Composants du Common Langage Runtime :

Les fonctionnalités du Common Langage Runtime sont décrites dans la figure suivante :

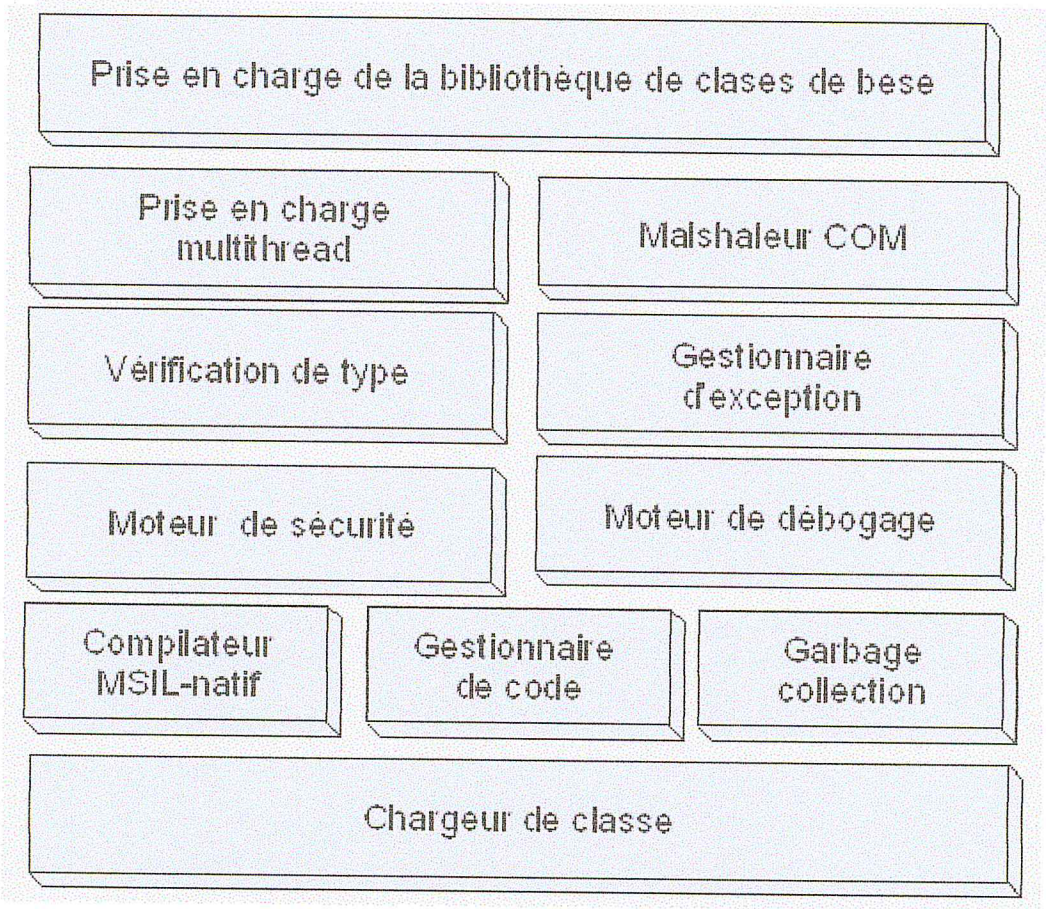


Figure A.4 Composants du Common Langage Runtime.

- **Chargeur de classe :** charge en mémoire l'implémentation d'un type chargeable et le répare à l'exécution.
- **Compilateur MSIL(**Microsoft Intermediate Language **)- natif :** convertit MSIL en un code natif.
- **Gestionnaire de code :** gère l'exécution du code.
- **Garbage collection :** fournit une gestion automatique de la durée de vie de tous nos objets.
- **Moteur de sécurité :** fournit une sécurité par preuve, fondée sur l'origine du code en plus de l'utilisateur.
- **Moteur de débogage :** permet de débogue l'application et de tracer l'exécution du code.
- **Vérification de type :** n'autorisera pas les conversions non sécurisées ou les variables non initialisées.

- **Gestionnaire d'exceptions :** fournit un traitement structuré des exception.
- **Prise en charge multithread :** fournit des classes et des interface qui permettent la programmation multithread.
- **Marchaleur COM :** fournit le marshaling à partir et à destination de COM.
- **Prise en charge de la bibliothèque de classes.NET Framework :** intègre du code au runtime qui prend en charge la bibliothèque de classes.

3.2-Bibliothèque de classes .NET Framework :

Elle fournit de nombreuses nouvelles fonctionnalités puissantes du runtime et d'autres services essentiels de haut niveau via une hiérarchie d'objets qui s'appelle un espace de nom.

La figure suivante décrit ces espaces de nom :

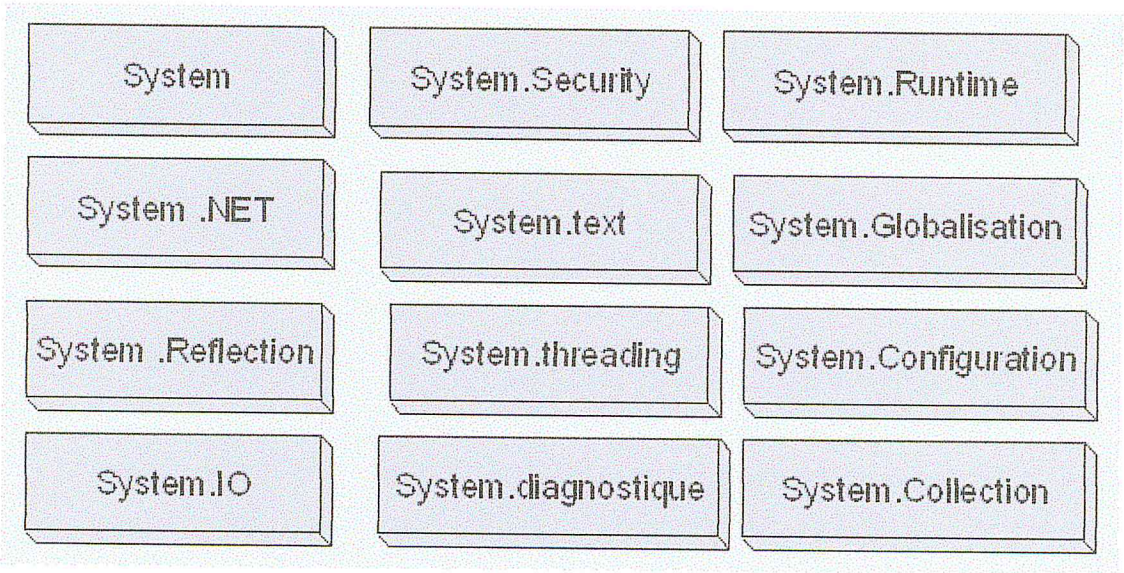


Figure A.5 Bibliothèque de classes .NET

- **System :** contient des classes fondamentales et de base qui définissent les types de données, les événements, les interfaces, les attributs…etc.
- **System.Collection :** fournit des listes, des tables et d'autres méthodes de regroupement de données.
- **System.IO :** il s'agit d'entrée /sortie et flux de fichiers.
- **System.NET :** fournit une prise en charge des sockets et de TCP/IP.

Pour plus d'information consulter la documentation du SDK Microsoft .NET Framework.

3.3- ADO .NET données et XML :

ADO.NET est la nouvelles génération de la technologie ADO (ActiveX Data Object) Son but est l'amélioration du modèle de programmation déconnecté, ainsi elle est riche de XML.

- **System.Data :** comprend la classe **DataSet** qui représente des tables mutiples et leur relations.
- **System.XML :** il comprend un outil d'écriture et un analyseurXML.

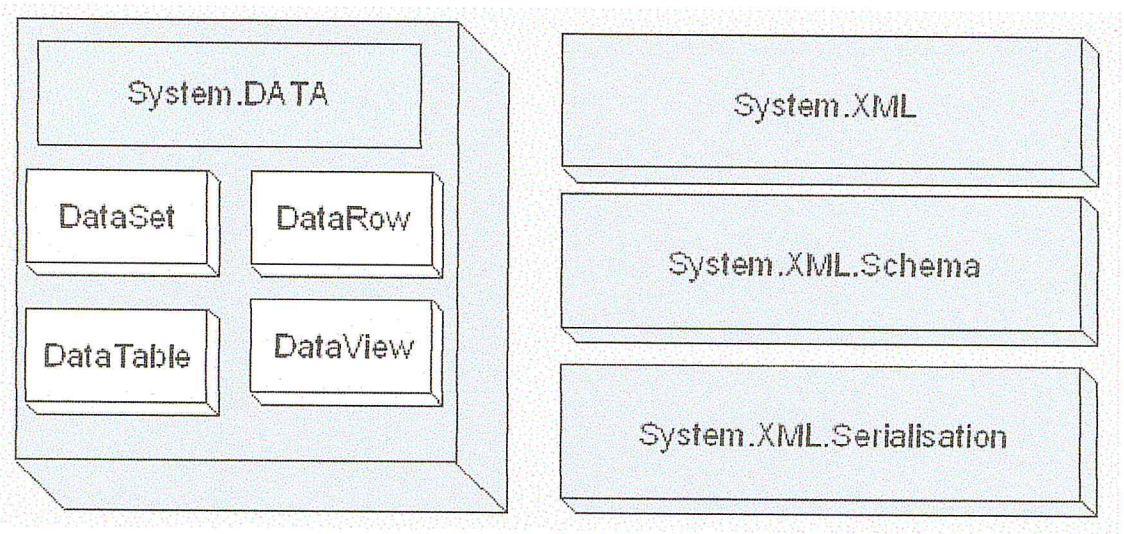


Figure A.6 ADO.NET

3.4- ASP .NET formulaires et services Web (Active Server Pages) :

ASP.NET est un cadre de programmation élaboré sur la base du Common Language Runtime et qui peut être employé sur un serveur pour créer des applications Web puissantes. Les formulaires ASP.NET sont des outils d'emploi pour la création d'interface utilisateur Web dynamiques.

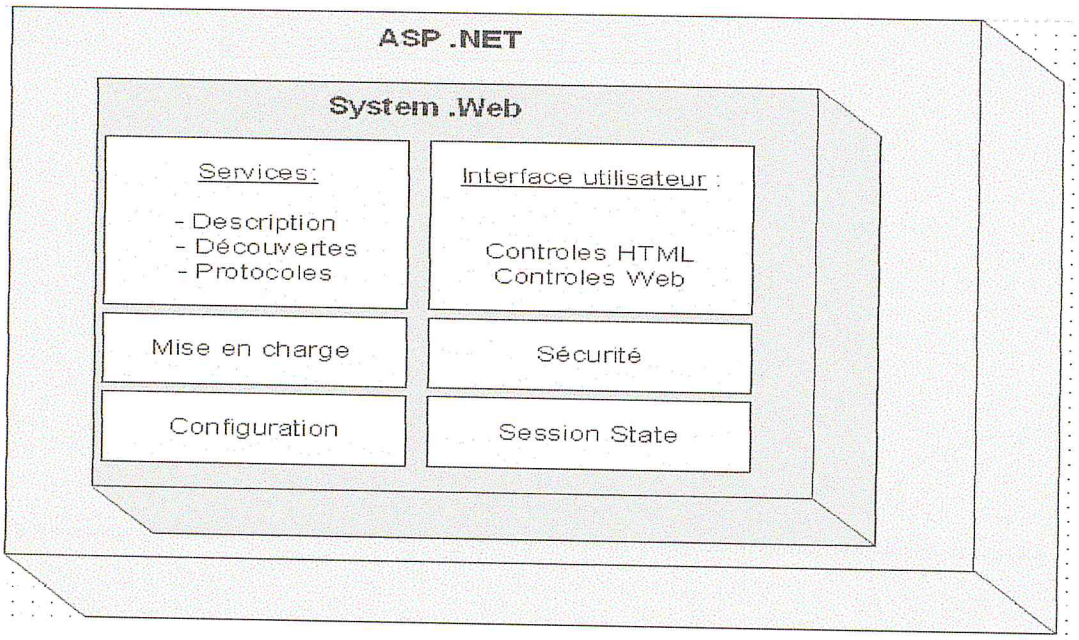


Figure A.7 Présentation des formulaire et services Web : ASP .NET

Dans **System.Web,** certais services tels que la mise en charge, la sécurité ou la configuration sont partagés par les services Web et l'interface.

3.5 Interface utilisateur :

La figure suivante explique comment le .NET gère l'interface Framework des applications Windows traditionnelles :

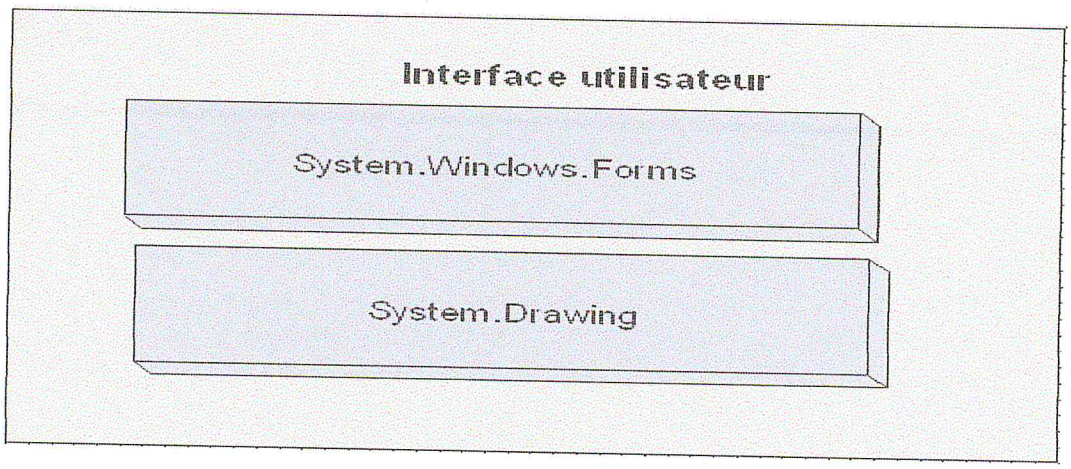


Figure A.8 Interface utilisateur.

- **System.windows.Forms** représente l'interface utilisateur coté client ;tandis que **System.Drawing** représente la nouvelle génération de services GDI + (Graphic Device Interface Plus).

1. Introduction :

Il est parfois nécessaire de préciser les balises et attributs auxquels on a droit lors de l'édition d'un document XML. A cet effet, le XML permet d'utiliser un fichier afin de déclarer tous les éléments qui peuvent avoir lieu dans le document et de vérifier qu'un document XML est conforme à une syntaxe donnée .Ce fichier permettra à l'interpréteur de vérifier si les données fournies correspondent à celles attendues, il décrit la structure des documents y faisant référence grâce à un langage adapté.

La norme XML définit en conséquence une définition de document type appelée DTD (en anglais *Document Type Definition*), c'est-à-dire une grammaire permettant de vérifier la conformité du document XML. Cette norme n'impose pas l'utilisation d'une DTD pour un document XML, mais elle impose par contre le respect exact des règles de base de la norme XML vu que ce dernier a été conçu pour être utilisé avec ou sans DTD. Opérer sans DTD permet d'inventer un balisage sans avoir à le définir de façon formelle, avec le risque de perdre le contrôle automatique de la structuration de documents.

La DTD est l'ensemble des règles et des propriétés que doit suivre le document XML. Ces règles définissent généralement le nom, le contenu des balises et le contexte dans lequel elles doivent exister. Cette formalisation des éléments est particulièrement utile lorsqu'on utilise de façon récurrente des balises dans un document XML, elle impose des contraintes sur l'arborescence que doit prendre un document XML.

2. La syntaxe

2.1. La déclaration de la DTD dans le document instance

Une DTD est une grammaire nommée, écrite dans un fichier du même nom qui possède l'extension *.dtd*. Elle est appelée dans le prologue du document XML de la façon suivante :

```
<!DOCTYPE racine SYSTEM "URL.dtd">
```

Où racine est le nom donné à la DTD et 'URL.dtd' le chemin d'accès (URL) au fichier.

2.1.1 La déclaration de type de document :

Dans un document valide, on trouve généralement un moyen qui nous permet de faire référence à notre DTD et de trouver la racine du document (à l'aide d'un fichier appelé DTD. Cet outil est la déclaration de type de document. Elle permet de spécifier une DTD pour un document XML. Selon le type de la DTD, il existe trois types de déclaration [ERWS 02] :

- La déclaration d'une DTD externe.
- La déclaration d'une DTD interne.
- La déclaration d'une DTD mixte.

2.1.1.1 La déclaration d'une DTD externe :

La déclaration de l'utilisation d'une DTD externe doit se faire avant l'élément racine et elle est introduite dans un document XML par l'instruction DOCTYPE immédiatement suivi par le nom de l'élément racine, suivi par l'identifiant de la DTD (SYSTEM ou PUBLIC). Pour une référence externe, deux façons existent :

- Grâce à l'identifiant SYSTEM :

```
<!DOCTYPE Nom-racine SYSTEM "chemin-DTD-externe">
```

Les constituants de la balise sont :

<	!DOCTYPE	Nom-racine	SYSTEM	chemin-DTD-externe	>
Ouverture de la balise	Instruction XML	Le nom de l'élément racine du document. C'est un nom XML.	On fait appel à une DTD externe	Le chemin de la DTD.	Fermeture de la Balise

- Grâce à l'identifiant PUBLIC :

```
<!DOCTYPE Nom-racine PUBLIC "Non-identifiant-public" " FPI-DTD-externe">
```

Les constituants de la balise sont :

<	!DOCTYPE	Nom-racine	PUBLIC	Non-identifiant-public	FPI-DTD-externe	>
Ouverture de la balise	Instruction XML	Le nom de l'élément racine du document. C'est un nom XML.	la DTD est publiée et accessible au plus grand nombre d'utilisateurs. C'est une ressource disponible pour tous sur un serveur Web distant.	Le Nom de la DTD (identifiant public)	Le chemin de la DTD externe.	Fermeture de la balise

Ce type de déclaration est utilisé lorsqu'on fait référence à une DTD publiée à un usage élargi c'est-à-dire une DTD standard.

Nom-identifiant-public : le nom de l'identifiant public. Il se compose de quatre éléments séparés par le caractère « // »et désignant dans l'ordre :

- type_enregistrement : un signe + si c'est selon la norme ISO 9070, un signe - sinon ;
- propriétaire : nom du propriétaire (entreprise ou personne) ;
- DTD description: description textuelle pour laquelle les espaces sont autorisés ;
- langue : un code de langue ISO 639.

2.1.1.2 La déclaration d'une DTD interne :

Elle doit être déclarée avant l'élément racine, elle est introduite par l'instruction DOCTYPE suivi par le nom que porte l'élément racine. Elle est encadrée par les deux crochets [et]. La syntaxe est la suivante :

```
<!DOCTYPE Nom-racine [déclarations]>
```

Les constituants de la balise sont :

<	!DOCTYPE	Nom-racine	déclarations	>
Ouverture de la balise	Instruction XML	Le nom de l'élément racine du document. C'est un nom XML.	Les règles de la DTD interne	Fermeture de la Balise

2.1.1.3 La déclaration d'une DTD mixte:

La déclaration de type de document a la forme suivante:

```
<!DOCTYPE Nom-racine SYSTEM "chemin-DTD-externe" [déclaration]>
```

Les constituants de la balise sont :

<	!DOCTYPE	nom_racine	SYSTEM	Chemin_ DTD_ externe	déclaration	>
Ouverture de la balise	Instruction XML	Le nom de l'élément racine du document. C'est un nom XML.	On fait appel à une DTD interne au site	Le chemin de la DTD.	L'ensemble des balises de la DTD interne	Fermeture de la Balise

2.2. La déclaration des éléments

Les éléments sont les principaux composants des langages à balises comme XML ou HTML. Les DTD acceptent en fait un type de base (PCDATA), qui peut être utilisé avec des connecteurs et des contraintes d'occurrences pour créer des modèles de contenu plus complexes. Il existe également deux types spéciaux pour les éléments : EMPTY et ANY. **[REMC 02]**

2.2.1. Les différents types d'éléments

2.2.1.1. Eléments vides

Les éléments vides sont déclarés avec le mot clé EMPTY.
exemple

```
<!ELEMENT élément EMPTY>
```

L'élément élément apparaîtra alors comme ceci dans un document instance : <élément/>

2.2.1.2. Eléments contenant du texte

Le seul type de base disponible dans les DTD est PCDATA, ce qui signifie *Parsed Character Data*. Un élément de type PCDATA contient en fait du texte qui sera analysé par le processeur XML. Nous utilisons la syntaxe suivante pour déclarer un élément nom contenant du texte :

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

2.2.1.3. Eléments ayant tout type de contenu

Un élément déclaré avec le mot clé ANY peut contenir n'importe quelle combinaison de données analysables par le processeur XML ; c'est-à-dire que tous les éléments sont autorisés. Par exemple, l'élément document peut contenir tous types d'éléments :

```
<!ELEMENT document ANY>
```

2.2.2. Les connecteurs

Les DTD fournissent deux connecteurs qui permettent de complexifier le contenu d'un élément.

2.2.2.1. Le connecteur de choix

Le connecteur de choix des DTD permet d'indiquer qu'un des éléments seulement doit être présent dans le document instance. Dans l'exemple suivant, l'élément voiture peut être soit du type essence ou diesel, mais pas les deux à la fois.

```
<!ELEMENT voiture (essence | diesel)>
```

2.2.2.2. Le connecteur de séquence

Le connecteur de séquence permet d'exprimer le fait qu'un élément est composé d'une suite d'éléments. Par exemple, l'élément lettre est composé d'un élément auteur, destinataire et corps dans cet ordre.

```
<!ELEMENT lettre (auteur, destinataire, corps)>
```

2.2.3. Les contraintes d'occurrences

En plus des types d'éléments et des deux connecteurs, des caractères spéciaux (*,+,?) permettent de spécifier les contraintes d'occurrences des éléments dans les documents instances. **[REMC 02]**

- Le caractère "?": il suit un élément ou un groupe d'éléments et indique qu'il peut y en avoir 0 ou 1 occurrence ; il indique le caractère optionnel de l'élément.
- Le caractère "*": il suit un élément ou un groupe d'éléments et indique qu'il y en a 0 ou plusieurs occurrences.
- Le caractère "+": il suit un élément ou un groupe d'éléments et indique qu'il doit y avoir au moins 1 occurrence.

Nous pouvons noter que les éléments peuvent être définis de façon récursive, c'est à dire qu'ils sont capables de se contenir eux-mêmes :

```
<!ELEMENT element (element1, element*)>
```

Fréquemment, la récursivité est moins directe, un fils ou un petit fils d'un nœud élément contient le nœud élément lui-même.

2.3. La déclaration des attributs

Les attributs fournissent une information supplémentaire sur les éléments. Les attributs sont toujours placés à l'intérieur de la balise ouvrante d'un élément, et ils apparaissent sous la forme d'une paire nom-valeur ; la valeur de l'attribut étant toujours entre guillemets. Dans une DTD, les attributs sont déclarés en utilisant la syntaxe suivante :

```
<!ATTLIST nomElément nomAttribut typeAttribut occurence>
```

2.3.1. Les types d'attributs

Les attributs peuvent être de différents types. Les voici représentés dans un tableau.

Type	Explication
CDATA	ce type correspond à du texte
ID	représente un identifiant pour l'élément, qui doit être unique dans le document
IDREF	représente une référence à un ID se trouvant dans le même document
NMTOKEN	un seul mot
ENTITY	la valeur de l'attribut est une entité
NOTATION	la valeur de l'attribut est le nom d'une notation

Tableau B1 - Types d'attributs

Signalons en outre qu'un attribut peut également être une liste de ID, IDREF, NMTOKEN ou ENTITY séparés par des espaces, grâce aux types IDS, IDREFS, NMTOKENS et ENTITIES respectivement.

2.3.2. Les contraintes d'occurrences

Le tableau suivant regroupe les contraintes d'occurrences possibles :

Valeur	Explication
Valeur	la valeur par défaut de l'attribut est 'valeur'
#DEFAULT valeur	la valeur par défaut de l'attribut est 'valeur'
#REQUIRED	l'attribut doit impérativement apparaître
#IMPLIED	l'attribut peut ne pas apparaître
#FIXED valeur	l'attribut a une valeur constante fixée

Tableau B2 - Contraintes d'occurrences pour les attributs

2.4. La déclaration des entités

Une entité est en réalité une référence à quelque chose, soit un nom de variable (entité interne), soit un alias (entité externe). Les entités sont très utilisées, notamment pour représenter :

- un caractère spécial à l'intérieur d'un document XML, comme c'est le cas pour 'eacute' qui représente le "é". Ceci permet au document d'être portable. Plusieurs entités de caractères spéciaux sont prédéfinies (&amp : &, &lt : <, &gt : >, &apos :', &quot : ").

- une phrase fréquemment utilisée dans le document XML. Le fait d'appeler une entité représentant un texte permet un gain de temps dans l'écriture du document XML. Il suffit d'écrire le nom de l'entité pour qu'une phrase apparaisse dans le document final. Cela permet également d'éviter les erreurs de frappe et de modifier facilement, rapidement et complètement une phrase (il suffit de modifier une fois la déclaration de l'entité dans la DTD).
- une ressource non XML, qui associée à une NOTATION peut être traitée et donc apparaître dans le document final. Une NOTATION est en fait un nom de format, un type de ressource. Le processeur XML devra faire appel à une application annexe qui l'aidera à formater cette ressource et donc la faire apparaître correctement dans le document. Ce type d'entité est dit "binaire" car elle n'est pas traitée comme étant du code XML. C'est le cas des images (bmp, gif ou jpeg), ou bien des codes ou scripts écrits dans un langage de programmation (C, Java, Perl). L'entité représente alors un fichier non XML, extérieur au document, qui pourra être interprété à l'intérieur du document. Il est nécessaire d'expliciter dans la DTD la localisation exacte du fichier d'exécution de l'application non XML avec la syntaxe suivante :

```
<!NOTATION txt SYSTEM "Chemin">
```

- une même entité déclarée externe peut-être utilisée dans plusieurs documents XML, ce qui permet une mise à jour générale et donc une standardisation pérenne.

3. Les points importants

3.1. La validité d'un document

Lorsque le processeur XML lit un document XML, il regarde en premier lieu les déclarations situées dans le prologue et notamment la déclaration de la DTD. Il va ensuite parcourir le document XML, en s'arrêtant sur chaque balise. Il comparera ensuite la composition effective de l'élément (balise) avec celle définie dans la DTD. Si des différences de structuration apparaissent, le processeur rendra comme résultat que le document n'est pas valide car il ne respecte pas les règles de structuration de la DTD. Ceci permet d'éviter les erreurs de compositions de documents instances XML.

4. Les limites

Les DTD sont des fichiers de structuration qui ont leur propre syntaxe. Ce langage n'est pas du XML. Il n'est donc pas supporté par les mêmes outils que les documents XML. Ainsi, il est nécessaire d'avoir un outil spécifique pour pouvoir développer des DTD, ce qui peut apporter un surcoût financier et de temps lors de la validation puisqu'une autre application doit être ouverte pour la DTD.

La syntaxe en elle-même est très limitée. En effet, la syntaxe des DTD repose sur le principe de grammaire. Comme nous l'avons vu dans le chapitre précédent concernant la syntaxe, un type primitif unique existe dans les DTD : PCDATA, qui représente le type texte. Ainsi, il est impossible de vérifier le type des éléments qui entrent dans le document XML.

La grammaire a une autre limitation qui concerne le nombre d'occurrences. Bien sûr, il y a les caractères spéciaux tels que *, ?, +. Mais ceux-ci ne sont pas suffisants. En effet, ils restent très généraux. Il est impossible de définir un nombre d'occurrences plus précisément, explicitement, ni de définir une plage d'intervalles d'occurrences.

Les fichiers XML peuvent contenir des données qui proviennent ou qui sont destinées à être stockées dans une base de données. Toutefois, outre le fait qu'une base de données est dans la plupart des cas typée, elle est naturellement structurée. Un enregistrement d'une table est généralement composé de plusieurs champs qui peuvent être eux-mêmes des références à une autre table de cette base. Or cette notion, tout comme celle d'héritage n'est pas présente dans les DTD. Ainsi il est nécessaire de fournir un gros effort pour établir la correspondance entre le contenu des documents XML et la base de données qui les stocke.

5. En résumé

Nous avons vu que la DTD est un mécanisme décrivant chaque objet pouvant apparaître dans le document. Il est très important d'avoir une bonne DTD et donc de passer du temps à sa conception. C'est elle qui nous permettra de construire des documents XML valides. Il peut-être aussi astucieux d'utiliser des DTD existantes et éventuellement de les modifier, c'est généralement plus efficace que de tout refaire à zéro.

Toutefois, il devient de plus en plus évident que les DTD ont atteints leurs limites et qu'elles ne correspondent plus aux besoins d'aujourd'hui. Le W3C a donc créé un groupe de travail qui a développé un autre langage de schémas pour les documents XML appelé XML Schema.

Références Bibliographique :

[SILA 00] : « *Introduction au XML* »de Simon St Laurent des éditions OSMAN EYROLLES MULTIMEDIA 2000
Ce livre propose une initiation claire et accessible au XML, complétée d'astuces pratique.

[KMPJ 01] :« *XML et les bases de données* » de – KEVIN WILLIAMS . MICHAEL BRUNDAGE . PATRICK DENGLER . JEFF GABRIEL . ANDY HOSKINSON . MICHAEL KAY . THOMAS MAXWELL . MARCELO OCHOA . JOHNNY PAPA . MOHAN VANMANE.
Traduit de l'anglais par FABRICE LEMAINQUE , PAOLA APPELIUS-ROY , YOLAINE ROCHETAIG , INGRID PIGUERON.
Wrox Press et Editions Eyrolles 2001 ; ISBN 2-212-009282-2
Même si ce livre aborde les concepts de base de la technologie XML, il est destiné essentiellement à tous les développeurs d'applications Web, intranet ou e-commerce et à tous les concepteurs et administrateurs de bases de données. Il explique comment stocker un document XML dans une base de données

[ALMI 01] :« *XML langage et applications* » de Alain Michard
Editions Eyrolles, 2001, ISBN 2-212-09206-7
Cet ouvrage est destiné en premier lieu aux concepteur de systèmes d'information, que ces systèmes soient de « simple » sites Web riches de quelques centaines de documents, ou qu'ils donnent l'accès à de nombreuses sources distribuées et hétérogènes. Il s'adresse aussi aux auteurs de documents qui voudront comprendre et éventuellement contrôler les productions XML générées par les éditeurs XML. Il s'adresse enfin aux étudiants en ingénierie documentaire et multimédia.

[ERWS 02] : « *XML in a Nutshell* » Elliote Rusty Harold, W. Scott Means, édition O'Reilly, 2002. ISBN : 2-841-77143-1 Un livre de référence XML: L'ambition de cet ouvrage est d'être une référence aussi complète et détaillée que possible sur les formats API utilisés lors de l'analyse, de transformation et de la présentation de données XML. Ce livre repose sur un équilibre entre la présentation détaillée des possibilités et l'aide à la réalisation. Par cette double approche, l'ouvrage doit faciliter au développeur exercé l'utilisation du langage. La précision du texte fait de ce volume un incontournable pour la programmation en XML

[GEGA 02] :« *XML des bases de données aux services Web* » : de Georges Gardin édition Dunod paris 2002 ISBN 2 10 0069330 »
Cet ouvrage couvre les fondements de XML, les services Web, les bases de données XML et l'intégration de données et l'application avec XML. Il essaie d'expliquer clairement toutes ces techniques et architectures complexes et intégrées nécessaires à l'urbanisation des systèmes d'information modernes, donc montre le rôle d'XML dans les systèmes d'information dans toutes les dimensions.

[REMC 02] : « *XML précis et concis* » de Robert ECKSTEIN et Michel CASABIANCA
Traduction de James Guérin des éditions O'REILLY® 2002 . ISBN : 2-84177-104-0
Ce manuel est à la fois une introduction claire à la terminologie et à la syntaxe, et une référence exhaustive qui décrit toutes les instructions XML, depuis les attributs réservés jusqu'aux différents types d'entités et de données en passant par les déclarations IGNORE et INCLUDE. Cet ouvrage couvre également la dernière version de XSLT (1.0), adoptée par le W3C. Ce langage surpuissant permet de manipuler et transformer les documents XML à volonté.

[PTPB 03] : « *XML ED TITTEL ; EDI SCIENCE* »
Original edition bay the McGraw-Hill Companies
Traduit de l'américaine par Patrick Pabre, pour les éditions Dunod, Paris, 2003
ISBN : 2-10-0006934-9.
Cet ouvrage explique la terminologie, les concepts et les techniques de balisage d'XML. Il fournit également les notions de base sur les feuilles de styles CSS, les DTD et la syntaxe d'XML.

Sites Web :

http://www.w3.org/XML : W3C recommendation 1.0 , 6 octobre 2000, Ce document est la dernière version de la recommandation XML.

http://www.w3.org/TR/2001/REC-xmlschema-0-20010502 : Ce document est une traduction de la recommandation XML Schema du W3C, XML Schéma tome 0 : Introduction, datée du 2 mai 2001.

http://www.w3.org/TR/2001/REC-xmlschema-1-20010502 : Ce document est une traduction de la recommandation XML Schema du W3C, XML Schéma tome 1 : Structures, datée du 2 mai 2001.

http://www.w3.org/TR/2001/REC-xmlschema-2-20010502. :Ce document est une traduction de la recommandation XML Schema du W3C, XML Schéma tome 2 : Types de données, datée du 2 mai 2001.