Rapport projet de fin d'étude ASR

Accès à l'information XML par des requêtes XQuery au travers de son XSchema

Réalisé par :

DAB Marwa

MGARRECH Oussama

Encadré par :

Mme LOPES GANCARSKI Alda

2011/2012

Remerciements

Avant d'entamer la présentation de ce travail, il nous est particulièrement plaisant d'adresser nos meilleures reconnaissances et gratitudes à Mme Alda GANCARSKI, qui nous a aidé à la réalisation de ce projet, en suivant du près notre travail et en nous guidant pour aboutir au bon fonctionnement de notre application.

Nous tenons également à remercier le département Informatique de Télécom SudParis, généralement, et son coordinateur Mr. Michel SIMATIC en particulier, pour l'intérêt qu'il a attribué au bon déroulement et au suivi de ce projet.

On leur remercie également de nous offrir la possibilité de mener un tel projet qui était et serait certainement à grand apport par rapport à notre formation.

Résumé

Le présent travail s'intègre dans le cadre du Projet de Fin d'Etudes à Télécom SudParis. Intitulé « Accès à l'information XML par des requêtes XQuery au travers du XSchema», il présente comme objectifs essentiels de concevoir et réaliser une application qui automatise l'accès à l'information structurée dans des documents XML, faisant ainsi abstraction du langage de requête XQuery, en utilisant les outils technologiques Java.

L'interface utilisateur à développer doit permettre l'accès aux données XML, et la visualisation de la structuration des données.

Mots clés : Document XML, collection de document, XML, XSchema, XQuery

Abstract:

This work is part of the Project End of Studies at TelecomSudParis. "Access to XML information by XQuery through the XSchema", it presents as main objectives to design and build an application that automates the access to structured information in XML documents, making abstraction of the XQuery language, using Java technology tools. A final user interface to be developed must allow access to XML data, and visualization of the structure and data schema.

Keywords: XML document, document collection, XML,XSchem, XQuery

Table des matières

Remerciments……i

Résumé……ii

Liste des figures……v

Liste des tableaux……vi

Liste des Abréviations……vii

Introduction générale……1

Chapitre 1 Etat de l'art et analyse de l'existant……2

Introduction……2

1.1 Concepts généraux de structuration de données……2

1.1.1 Structuration des documents……2

1.1.2 Le Langage XML (eXtensible Markup Language) ……2

1.1.3 Définition de la structure d'un document XML……3

1.1.4 Langage d'interrogation pour XML : le XQuery……5

1.2 Etude de l'existant……6

1.2.1 Query By Example……6

1.2.2 Critique de l'existant : ……7

1.3 Solution Envisagée……7

1.3.1 Solution Conceptuelle……7

1.3.2 Solutions techniques……7

Conclusion……8

Chapitre 2 Analyse des besoins et Conception de l'Application……9

Introduction……9

2.1 Contexte et Problématique……9

2.1.1 Contexte……9

2.1.2 Problématiques……9

2.2 Identification des Besoins……10

2.2.1 Besoins fonctionnels……10

2.2.2 Besoins non fonctionnels……10

2.3 Conception Architecturale du Système……11

2.3.1 Architecture du fonctionnement du système……..11
2.3.2 Patrons de Génération de Requêtes XQuery. ……12
2.3.3 Architecture de la Couche Présentation……15
2.3.4 Modélisation UML du système……15
Conclusion : ……18
Chapitre 3 Développement de l'application……19
Introduction : ……19
3.1. Environnement de travail : ……19
3.1.1. Environnement matériel……19
3.1.2. Outils de développement logiciel……19
3.2. Fonctionnement de l'application : ……20
3.2.1 Le choix de la collection XML et du fichier XSD : ……21
3.2.2 La validation de la collection XML par le fichier XSD : ……22
3.2.3 Présentation de la grammaire de la collection XML ……26
3.2.4. Choix des éléments et affichage de la requête XQUERY……28
3.2.5. Affichage des résultats .. ……29
3.2.6. Enregistrement du résultat dans un fichier XML : ……31
Conclusion : ……31
Conclusion Générale……32
Bibliographie et Netographie……33
Annexe 1 ……35
Partie I : Les composants d'un document XML ……35
Partie II : Modèle du XML Schéma ……36
Annexe II : Manuel d'utilisation de l'interface « XQUERY» ……38

Liste de figures

Figure 1.1: L'interface QBE de Kexi 6
Figure 2.1: Architecture du Système.....11
Figure 2.2: Bloc A. Génération des Requêtes non conditionnées pour un seul Elément 12
Figure 2.3: Génération des Requêtes non Conditionnées pour N éléments 13
Figure 2.4: Bloc B. Génération des Requêtes Conditionnées pour un Seul Elément.....14
Figure 2.5: Génération des Requêtes Conditionnées pour N Eléments.....14
Figure 2.6: Couche Présentation du Système.....15
Figure 2.7 **:** Diagramme de cas d'utilisation.....16
Figure 2.8: Diagramme de séquence.....18
Figure 3.1: L'interface graphique « Acceuil ».....21
Figure 3.2: La collection XML 22
Figure 3.3: Le fichier XSD..... 23
Figure 3.4: Validation réussite de la collection XML par le fichier XSD..... 24
Figure 3.5: Fichier XML modifié..... 24
Figure 3.6: Echec de la validation de la collection XML par le fichier XSD..... 25
Figure 3.7: L'arborescence de la collection XML..... 26
Figure 3.8: Collection XML modifiée..... 27
Figure 3. 9: Arborecsence de la collection après modification 28
Figure 3.10: Exemple de requête..... 29
Figure 3.11: Résultat de la requête XQuery..... 30
Figure 3.12: Le résultat sous forme d'un fichier XML..... 31
Figure A2.1: Lancer l'application 38
Figure A2.2: Affichage de la requête..... 38
Figure A2.3: Réinitialisation des choix de l'utilisateur **…..**..... 39
Figure A2.4: Enregistrements des résultats..... 39
Figure A2.5: Le menu « Fichier » de l'interface « XQUERY »**….**..... 40

Liste des tableaux

Tableau 2.1 : Liste des cas d’utilisation……..... 17

Liste des abréviations

API : Application Program Interface

DTD : Document Type Definition

QBE :Query By Example

UML : Unified Modeling Language

SQL : Structured Query Language

SGBD : système de gestion de base de données

XML : Extensible Markup Language

XSD : XML Schema Definition

Introduction générale

Le besoin de la structuration des données s'impose dans la plupart des applications et systèmes informatiques. Ainsi, la conception de ces systèmes et des services qui les régissent est basée souvent sur l'élaboration d'un modèle relationnel de données qui fait de l'application un ensemble des fonctionnalités agissant avec des entités homogènes.

Cependant, l'utilisation du modèle relationnel ne peut pas être adéquate avec des systèmes qui mettent en œuvre des données hétérogènes ou de structure mal-définie, cas qui n'est pas toujours exclu à rencontrer.

C'est dans ce contexte que le recours à la représentation eXtensible Markup Language (XML) se propose comme un outil absorbant l'irrégularité des données et permettant de manipuler des données dites semi-structurées. Sauf que la particularité de cette nouvelle forme de structuration de l'information présente des complexités incontestables au niveau de l'interrogation.

La popularité de l'usage de ces fichiers dans des divers applications d'une part, et la complexité de la manipulation du langage de requête XQuery, d'autre part, mène à penser à créer un outil qui fera abstraction du langage d'interrogation pour offrir la possibilité d'une manipulation graphique.

Ayant été engagés dans ce projet comme 'Projet de Fin d'Etudes', nous avons réussi à proposer une version de cet outil, et ce rapport présentera les différentes phases de réalisation du projet ainsi que les fonctionnalités qu'on a élaborées.

Structuré en trois grands chapitres, ce rapport détaille :

- Les concepts généraux mis en œuvre, l'état de l'art et la solution proposée dans le premier chapitre intitulé « Etat de l'art et Analyse de l'Existant ».

- Les différents besoins de l'application, la description de son architecture et sa conception dans le deuxième chapitre intitulé « Analyse des Besoins et Conception de l'Application ».

- Les différents résultats obtenus, dans le troisième chapitre intitulé « Réalisation ».

Chapitre 1

Etat de l'art et analyse de l'existant

Introduction

Dans ce premier chapitre, nous nous intéressons à introduire le cadre de notre projet: définir les différents concepts traités, présenter l'état de l'art en étudiant l'existant à priori dans le domaine d'émergence de notre application pour finir par la présentation des apports et solutions qu'amène notre application.

1.1 Concepts généraux de structuration de données

1.1.1 Structuration des documents

1.1.1.a Définition

Décrire la structure d'un document consiste à identifier et décrire chacun des éléments textuels, ou non textuels, qui le constituent. Elle définit l'ensemble des liens hiérarchiques et/ou logiques qui lient les éléments composant un document.

1.1.1.b Documents Structurés

Les documents structurés sont des documents qui contiennent de l'information à propos de leur structure logique, sémantique et intellectuelle. La structuration des documents se fait moyennant plusieurs langages en l'occurrence le HTML, etc.

1.1.2 Le Langage XML (eXtensible Markup Language)

1.1.2.a Présentation

Le XML est un meta- langage de balisage permettant de marquer des documents afin de les structurer. Il sert à coder le sens et la structure d'un document plutôt que sa forme. Ce langage est extensible car il permet de créer des vocabulaires permettant de décrire une structure de données particulière ou propre à un métier.

1.1.2.b XML et la structuration des documents

La norme XML est appropriée pour définir des documents structurés pour deux raisons principales:

1. Le modèle de données est hiérarchique, ce qui est également le cas des documents textuels, hypermédias, et de bon nombre de structures de données.

2. Les documents XML peuvent être validés selon une grammaire, permettant ainsi de définir les structures possibles et donc les traitements à effectuer. **(Voir « Parie I » de « Annexe I » pour plus de détails sur les spécifications du langage XML).**

1.1.2.c Intérêts

Le langage XML présente plusieurs intérêts parmi lesquels on note :

- Structurer l'information.
- Rendre explicite la structuration d'un document.
- Assurer l'échange entre humains et programmes informatiques.
- Traiter l'information structurée en XML grâce à des nombreux outils.

1.1.3 Définition de la structure d'un document XML

La définition de la structure d'une collection de document XML sert à produire un schéma décrivant la structure de cette collection et à déclarer ses différents éléments et attributs ainsi que leurs types. Il s'agit donc de créer un standard à suivre pour instancier les documents appartenant à la collection qui respectent le schéma correspondant.
La définition de ce schéma descriptif se fait en utilisant le Document Type Definition (DTD) ou bien le XSchema.

1.1.3.a Présentation du XSchema

Le XML Schema Language Definition (XSD) est un langage de description de format de document XML au travers un schéma que l'on appelle XML Schema (XSchema). Ce dernier permet de définir la structure et le type de contenu d'un document XML.

Un schéma XML définit les blocs de construction d'un document XML. notamment:

- les éléments et les attributs qui peuvent apparaître dans un document.
- les éléments qui sont des éléments enfants.
- l'ordre des éléments enfants.
- le nombre d'éléments enfants.
- si un élément est vide ou peut contenir du texte.

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

Le XSD intègre plusieurs types de données parmi lesquels on cite : le type chaine de caractères, entier, date, heure …

(voir « Parie II » de « Annexe I » pour plus de détails sur les spécifications du langage XShema).

1.1.3.b Comparaison entre le XSchema et le DTD

Une instance d'un *XSchema* est amène plusieurs différences par rapport à l'instance DTD correspondente .

- **Limitation des DTDs**

  Le DTD présente plusieurs limites par lesquelles on cite :

- Nécessite d'utiliser un outil pour manipuler un tel fichier, différent de celui utilisé pour un document XML.

- Ne supporte pas les espaces de nom: il n'est pas possible d'importer des définitions de balises définies par ailleurs dans un fichier XML défini par une DTD.

- Le « typage » des données est extrêmement limité.

- **Caractéristiques de XSchema**

Les schémas XML seront très bientôt utilisés dans la plupart des applications Web comme un remplacement pour les DTD pour les raisons suivantes:

- Les schémas XML sont extensibles à des ajouts futurs.
- Les schémas XML supportent l'héritage: les éléments peuvent hériter du contenu et des attributs d'un autre élément.
- Les schémas XML sont écrits en XML.
- Les schémas XML prennent en charge les types de données.

-Les schémas XML prennent en charge les espaces de noms.

1.1.4 Langage d'interrogation pour XML : le XQuery

1.1.4.a Présentation

XQuery est le langage de manipulation des documents XML servant à la consultation de l'information contenue dans un document XML. XQuery est basé sur le **XPath** (un langage pour localiser une portion d'un document XML)

XQuery est conçu pour interroger des données XML, pas seulement sous forme de document XML, mais tout ce qui peut apparaître comme XML dans le sens large de bases de données. XQuery est au XML ce que SQL est aux tables de la base de données.

1.1.4.b Spécification et Syntaxe

- **Le langage minimal de XQuery:** se base sur la norme XPath 2 (qui spécifie le langage de requête XML proprement dit), ayant comme cœur l'expression FLWOR (For , Let , Where ,Order by ,Return) :

L'expression FLWOR est une instruction de boucle, avec de nombreuses fonctionnalités, qui est assez similaire au SELECT de SQL. Grâce au **Where**, il est possible d'écrire des jointures internes ou externes.[6]

Voici un exemple de requête XQuery basée sur l'expression FLWOR :

```
for $x in doc("books.xml")/bookstore/book
where $x/price>30
return $x/title
```

Ici la clause « for » choisit tous les éléments « book » sous l'élément de « bookstore » dans une variable appelée $x. La clause « where » permet de choisir les « book » dont le prix supérieur à 30 et la clause « return » spécifie ce qui devrait être rendu. Ici il rend Le résultat de cette requête contient les éléments de titre.

1.1.4.c. XQuery et notre projet

Il s'avère d'après l'étude de XQuery que manipuler des données XML en utilisant des requêtes XQuery n'est pas simple. La complexité de ce langage de requête d'une part, et le besoin d'avoir un moyen d'accès direct, simple et performant aux documents structurés d'une autre part sont les deux raisons qui poussent à développer ce projet. Celui-ci s'agit en fait d'offrir à l'utilisateur un accès graphique aux données XML sans passer par des requêtes

XQuery. L'outil à développer sert donc à automatiser l'extraction des données structurées dans les documents XML, et à camoufler à l'utilisateur final de l'outil les procédures d'accès aux documents et de sélection de données.

1.2 Etude de l'existant

1.2.1 Query By Example

Inventé par Moshe Zloof pour le compte de la compagnie IBM, en 1977, Query By Example(QBE)[5] , est un type d'interface utilisateur servant à effectuer des recherches dans des bases de données relationnelles.

Le principe d'une interface QBE est que l'utilisateur présente un exemple du résultat de recherche attendu, puis le soumet au SGBD. Celui-ci recherchera alors toutes les données qui correspondent à cet exemple. Les tables de la base de données sont présentées à l'écran, et l'utilisateur peut les manipuler en vue de créer l'exemple. [3]

Un exemple de cette interface utilisateur est présenté dans la figure 1.1:

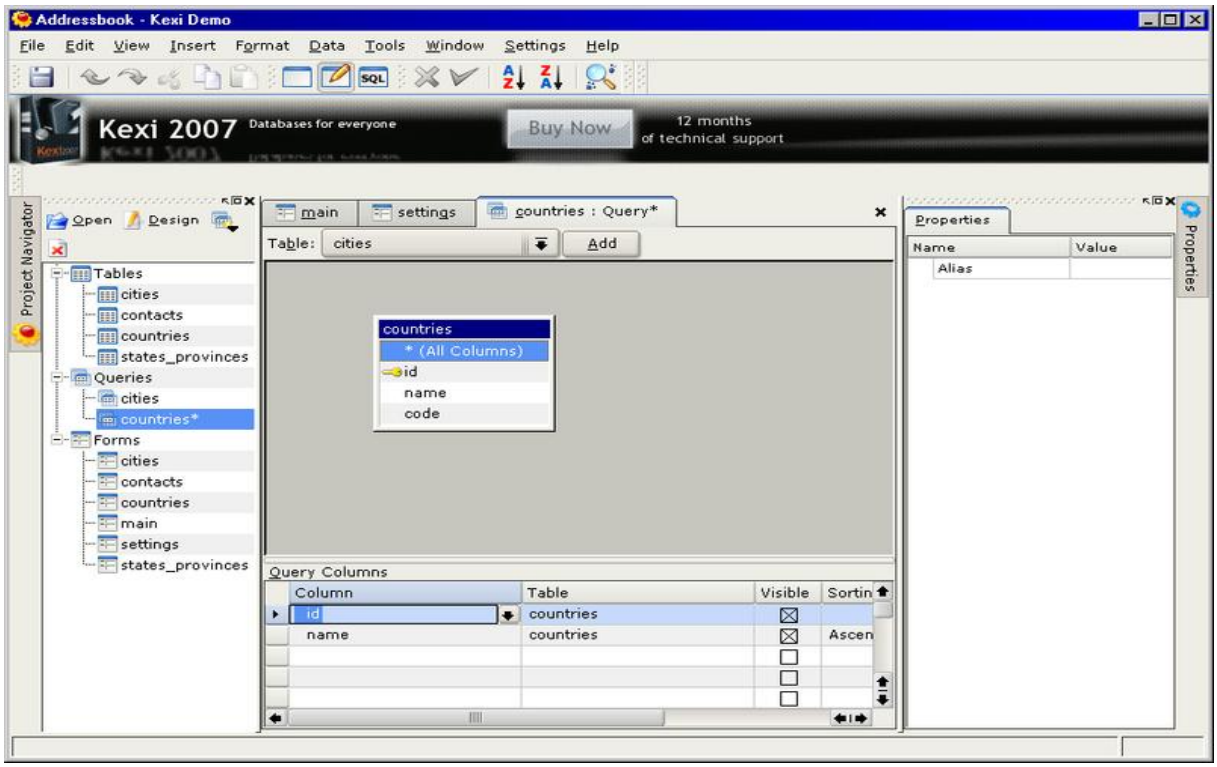

Figure 1.1 : L'interface QBE de Kexi

Donc il ne s'agit pas de manipuler un langage de requêtes, mais tout simplement de définir une image de la réponse que l'on veut obtenir, pour voir figurer les données répondant à l'interrogation demandée.

1.2.2 Critique de l'existant :

- QBE permet de faire abstraction du langage de requête SQL et offre à l'utilisateur une interface modélisant les différentes tables de la base de données interrogée ainsi que les différents champs à extraire depuis les tables.
- Limites: QBE est restreinte aux données structurées dans des bases de données relationnelles. Il ne traite pas les données semi-structurées, que présentent notamment les fichiers XML.
- Pour ce type particulier de données, l'état de l'art ne présente pas des applications répondant aux particularités des données XML.

1.3 Solution Envisagée

1.3.1 Solution Conceptuelle

Le même concept général de l'outil QBE est mis en œuvre pour notre application, sauf que la notre se présente comme un outil manipulant des données semi-structurées.

Des nouvelles notions liées aux données XML substituent donc à celle du QBE: il s'agit de traiter une collection de documents au lieu d'une base de données, des fichiers XML au lieu des tables et le langage de requête XQuery au lieu du SQL. Notre solution est adéquate pour un sous-ensemble simple et souvent le plus utilisé de requêtes XQuery et si l'utilisateur le veut, il peut lui-même éditer sa requête.

1.3.2 Solutions techniques

Le choix technologique est un élément décisif pour aboutir à une solution technologique pertinente et pour garantir la bonne qualité logicielle et l'évolutivité de l'application.
Nous allons alors expliciter le choix technologique ainsi que les moyens architecturaux employés dans cette partie.

1.3.2.a Le langage de programmation Java :

Le choix du langage Java pour la réalisation de l'application provient des plusieurs faits :

- Java présente des API qui permettent de manipuler des fichiers XML et des fichiers XSD

- Java permet également de répondre aux exigences graphiques de l'application (les composants JTable et JTree de Swing)

1.3.2.b Le choix des API

- API du traitement des fichiers XML: le choix était fixé sur JDom qui permet de traiter un document XML et d'extraire ses différents composants afin d'explorer son arborescence et sa structuration.

- API d'exécution des requêtes XQuery : Saxon. Contenant un parseur XQuery, Saxon est utilisé dans notre projet comme un processeur de recherche des données selon une requête appliquée sur une collection déterminée de documents XML.

- API graphiques : c'est l'API Swing qui a été déployée. Particulièrement, on a eu recours à l'usage des composant JTable et JTree pour la visualisation du schéma suivi par la collection et les documents qui la composent et pour permettre de définir l'image des données que l'on souhaite obtenir en résultat.

Conclusion

Au cours de ce chapitre, nous avons défini les concepts autour des documents structurés, les moyens d'établir ces documents et d'en définir le schéma ainsi que le langage qui permet de les interroger. Nous avons également présenté l'existant, en montré les limites avant de proposer des solutions conceptuelles et techniques.

Chapitre 2

Analyse des besoins et Conception de l'Application

Introduction

Afin de mieux spécifier notre application, nous nous attardons dans le présent chapitre à présenter le contexte du projet, et identifier et analyser les différents besoins .Ensuite ,nous passons à l'élaboration de la conception architecturale et la modélisation UML du système en vue d'une perception plus concrète de l'application.

2.1 Contexte et Problématique

2.1.1 Contexte

L'application est utilisée dans un contexte de recherche de données XML. Il s'agit de créer une interface utilisateur permettant l'interrogation des collections des documents XML sans avoir à saisir des requêtes XQuery.

2.1.2 Problématiques

L'application doit permettre de visualiser le schéma des fichiers traités ainsi que les différents éléments d'information et leurs types. Différemment des tables relationnelles, les documents XML peuvent présenter des éléments complexes ou simples. C'est à partir de cela que l'application doit pouvoir matérialiser les relations entre les différents champs et la structure générale que respectent les données interrogées.

La visualisation complète et correcte de l'image des données n'est qu'un premier enjeu de l'application, un autre enjeu est de pouvoir traduire l'interrogation graphique de l'utilisateur en une requête XQuery pour arriver enfin à présenter les données XML correspondantes en résultat.

2.2 Identification des Besoins

2.2.1 Besoins fonctionnels

Par rapport à l'utilisateur final, qui est le principal acteur dans le système, notre outil doit permettre de répondre aux besoins suivants:

- En présence d'un fichier XSD, valider les documents XML faisant partie de la collection de façon à vérifier leur compatibilité avec un XSchema unique.

- Générer l'interface décrivant cette collection et permettant l'accès aux différents éléments des documents qui existent.

- Générer les requêtes XQuery permettant d'extraire les données demandées par l'utilisateur et interroger la collection des documents en passant par un moteur de recherche.

- Exposer les données extraites à l'utilisateur.

- Permettre d'enregistrer les résultats obtenus sous forme d'un document XML

2.2.2 Besoins non fonctionnels

2.2.2.a L'ergonomie

Le système doit être doté d'un niveau acceptable d'ergonomie afin de pouvoir satisfaire aux utilisateurs de l'application.

2.2.2.b Evolutivité

Notre travail doit garantir la possibilité de modification de l'application et du développement de ses fonctionnalités.

2.2.2.c Temps de réponse

Le temps de réponse d'une application désigne le temps au bout duquel le système peut répondre à une requête en entrée. Il traduit la rapidité réalisée par le système, et il y a toujours tendance à réduire son coût par l'assurance d'un accès rapide des utilisateurs aux données et informations qu'il demande.

2.3 Conception Architecturale du Système

Dans cette partie, on va s'occuper de la description détaillée et de la modélisation UML de notre application. Ainsi, ça va permettre de bien comprendre le mécanisme du déroulement du programme.

2.3.1 Architecture du fonctionnement du système

Pour répondre aux besoins précédemment identifiés et décrits, nous concevons le fonctionnement de notre système comme il est décrit par l'architecture montrée dans la Figure 2.1 :

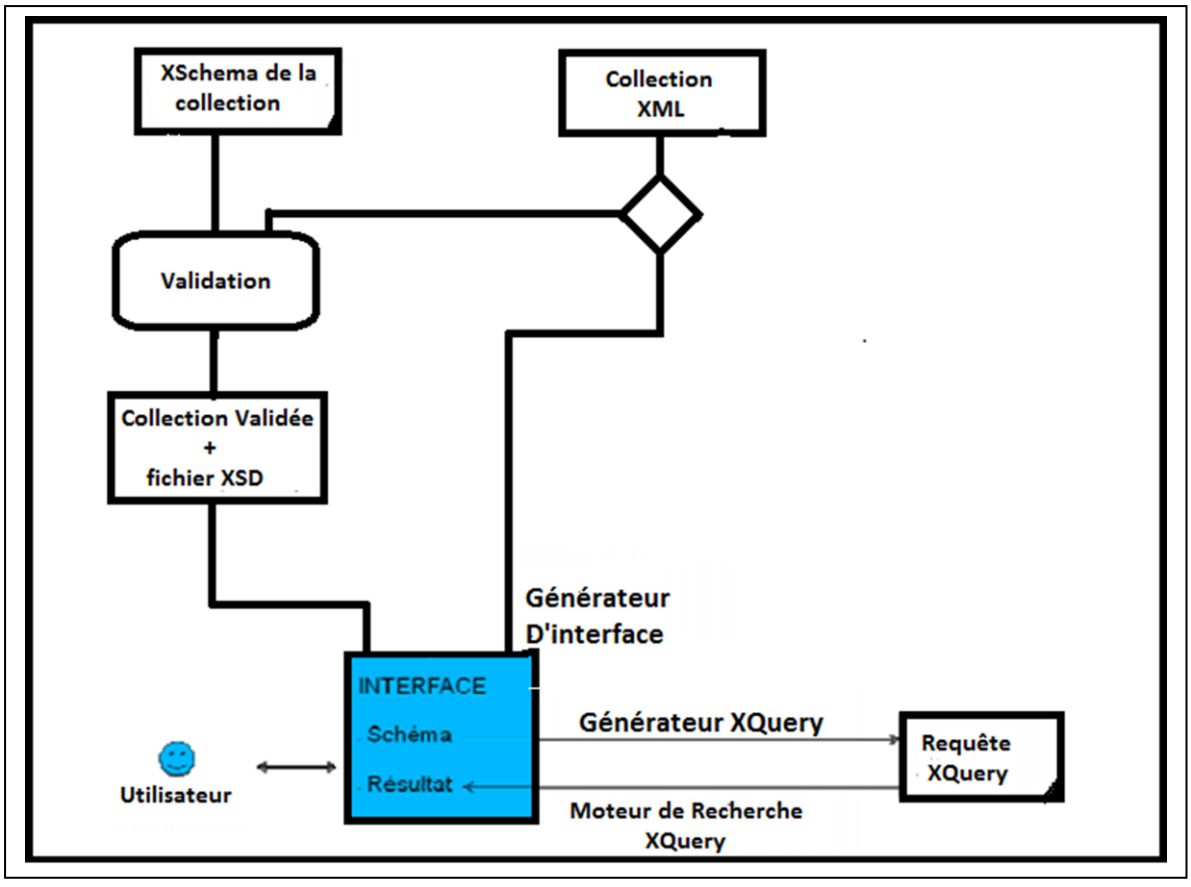

Figure 2.1 : Architecture du Système

Notre application comporte essentiellement deux modules principaux:

- le Générateur d'Interface ;
- le Générateur de Requête.

En premier temps, c'est le premier module qui intervient. Il s'agit de générer l'interface présentant le schéma qui décrit la collection et permettant d'exprimer les interrogations utilisateur ainsi que les résultats correspondants. Ce module génère l'interface utilisateur à partir de la collection passée en entrée. La présence du fichier XSD est optionnelle et elle permet en premier lieu de valider la collection de documents et en deuxième lieu de donner les informations sur les types des données XML visualisées.

Le deuxième module, quant à lui, consiste à récupérer les choix de l'utilisateur et les traduire en requêtes XQuery. L'exécution de ces requêtes se fait moyennant l'API du moteur de recherche Saxon [4] qui permet d'interroger les documents en question et d'en extraire les données correspondantes. Une autre fonctionnalité de ce module consiste à afficher la requête XQuery générée, pour donner à l'utilisateur possibilité de la modifier et de la ré-exécuter.

2.3.2 Patrons de Génération de Requêtes XQuery

- la Syntax de la requite XQuery: Requête Flower (For Let Where Order by Return)

  - **For:** sert à définir la source ou les éléments sous lesquels la recherche sera faite.
  - **Where** : sert à exprimer la restriction appliquée sur les éléments à extraire.
  - **Order by :** définit l'élément suivant lequel les résultats seront triés .
  - **Return :** définit l'élément à retourner

- Requête simple
  - Cas d'un seul élément

    Une seule requête est créée pour chaque document XML interrogée. Le bloc suivant résume le processus de construction de la requête.

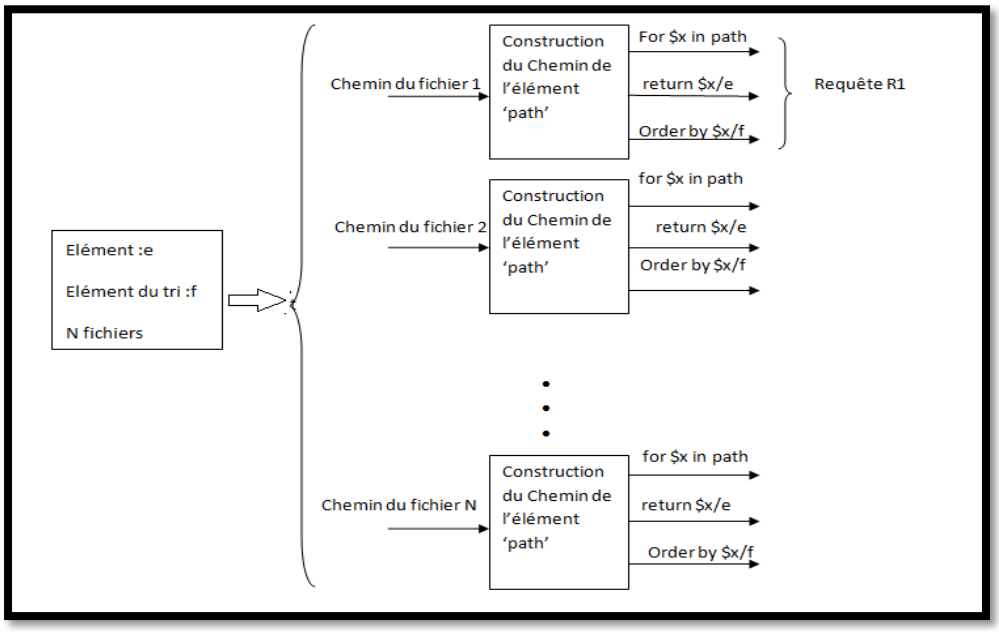

Figure 2.2 : Bloc A. Génération des Requêtes non conditionnées pour un seul Elément

❖ Cas de plusieurs éléments

N éléments étant choisis, on génère donc M requêtes par élément ce qui permet d'obtenir en total N*M requêtes.

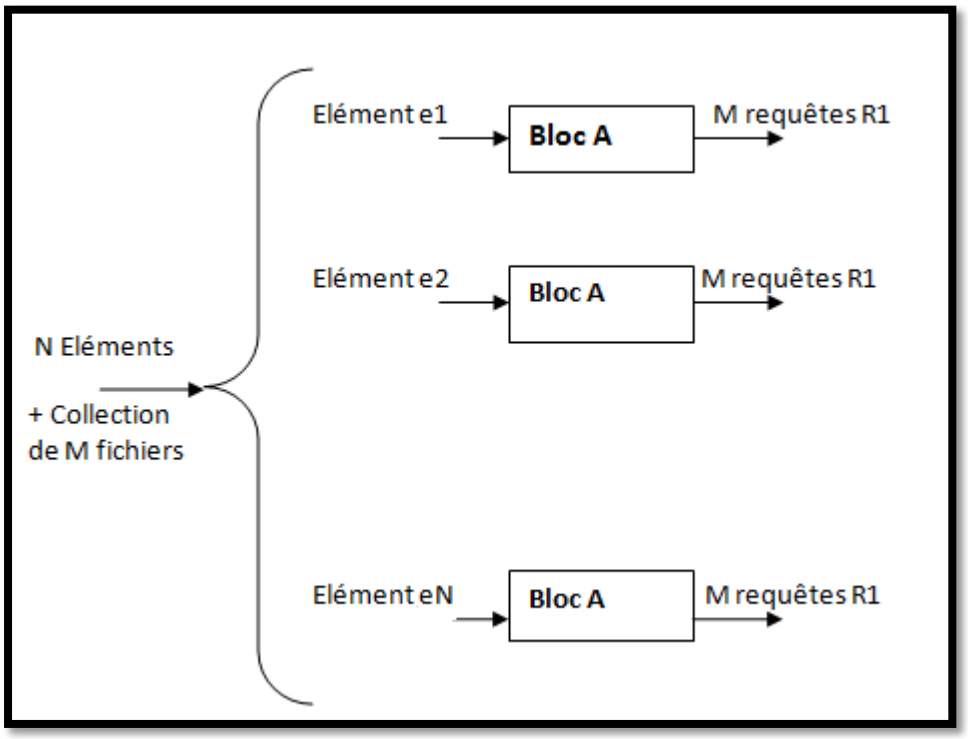

Figure 2.3 : Génération des Requêtes non Conditionnées pour N éléments

❖ Cas d'un seul élément

Une seule requête est créée pour chaque document XML interrogée. La condition peut concerner n'importe quel autre élément e', qui n'est pas nécessairement celui qu'on veut afficher (e). Le bloc suivant résume le processus de construction de la requête.

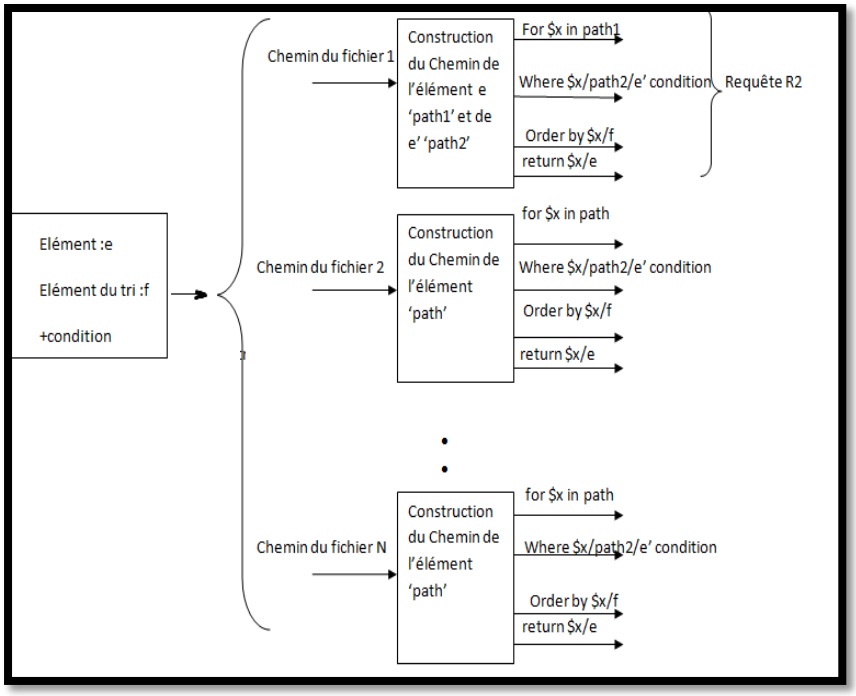

Figure 2.4 : Bloc B. Génération des Requêtes Conditionnées pour un Seul Elément

- Cas de plusieurs éléments

  On part du même ensemble des documents XML, sauf que plusieurs éléments (e1,e2,…,eN) sont à rechercher. Différentes conditions sont à appliquer sur des différents éléments (e'1,e'2,…,e'N).

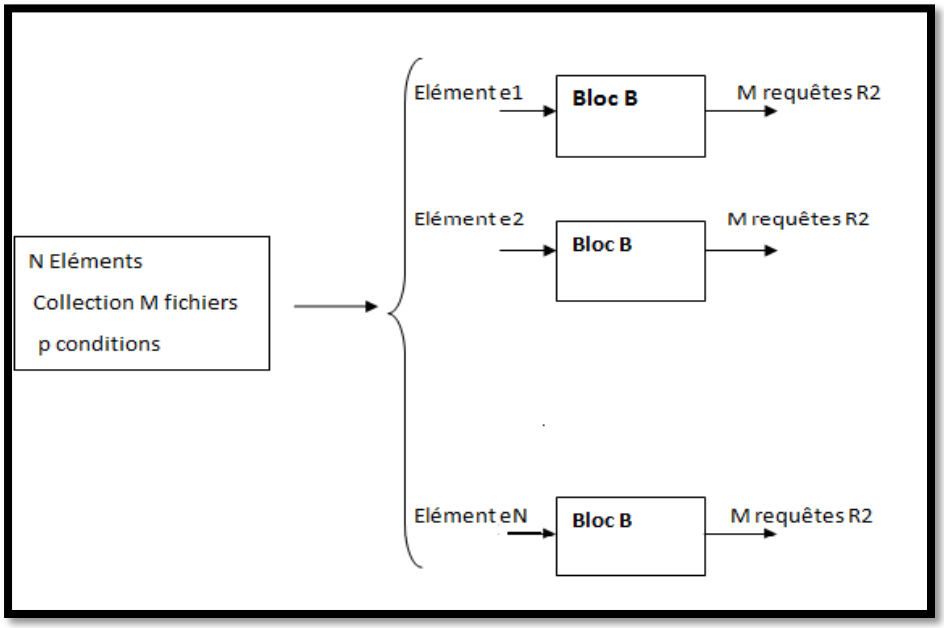

Figure 2.5 : Génération des Requêtes Conditionnées pour N Eléments

2.3.3 Architecture de la Couche Présentation

Graphiquement, notre application présente deux interfaces principales : interface « Accueil » qui donne accès à l'interface « XQUERY ». Le schéma montré par la Figure 2.2 décrit la composition de la couche présentation de notre application.

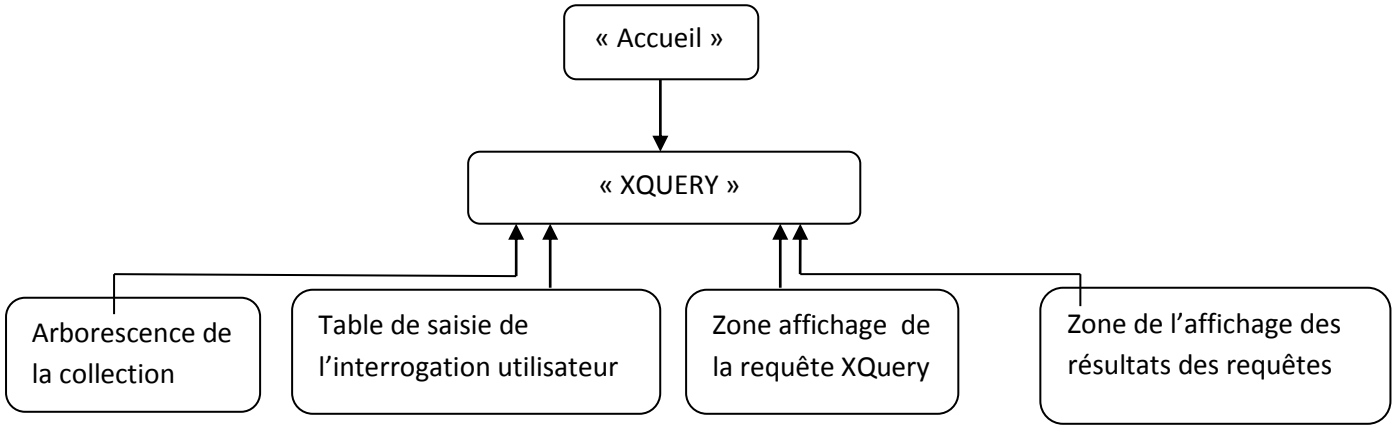

Figure 2.6 : Couche Présentation du Système

2.3.4 Modélisation UML du système

Pour la conception de notre système, nous avons eu recours au langage de modélisation Unified Modeling Language (UML). UML est un langage de modélisation graphique couramment utilisé pour la conception des projets de création des logiciels :

Parmi les diagrammes présentés par UML nous avons choisi de présenter deux types de diagramme : le premier est le diagramme de cas d'utilisation vu qu'il permet une bonne description des actions qu'on pourrait effectuer avec notre application ; le deuxième est le diagramme de séquence car il permet de représenter les interactions entre les différents objets selon un point de vue temporel. En effet, ce diagramme insiste sur l'ordre des envois des messages (l'interaction entre les objets), ce qui va assurer la bonne explication du fonctionnement de notre système.

Commonçons par le diagrammes de cas d'utilisation :

2.3.3.a Diagrammes de cas d'utilsation

Identification des acteurs :

On commence par identifier les acteurs susceptibles d'interagir avec l'application. Mais, tout d'abord il faut définir qu'est-ce qu'un acteur :

Un acteur représente l'abstraction d'un rôle joué par des entités externes (utilisateur, dispositif matériel ou autre système) qui interagissent directement avec le système étudié. [1]

Le système que nous désirons mettre en place comprend un seul acteur qui est **l'utilisateur.**

Identification des cas d'utilisation :

Un cas d'utilisation représente un ensemble des séquences d'actions réalisées par le système produisant un résultat observable intéressant pour un acteur particulier. Un cas d'utilisation modélisa un service rendu par le système. Il exprime les interactions acteurs/systèmes et apporte une valeur ajoutée notable à l'acteur concerné [1].

Passons maintenant à la présentation (Figure 2.3) et l'explication de notre diagramme de cas d'utilisation(Tableau 2.1) :

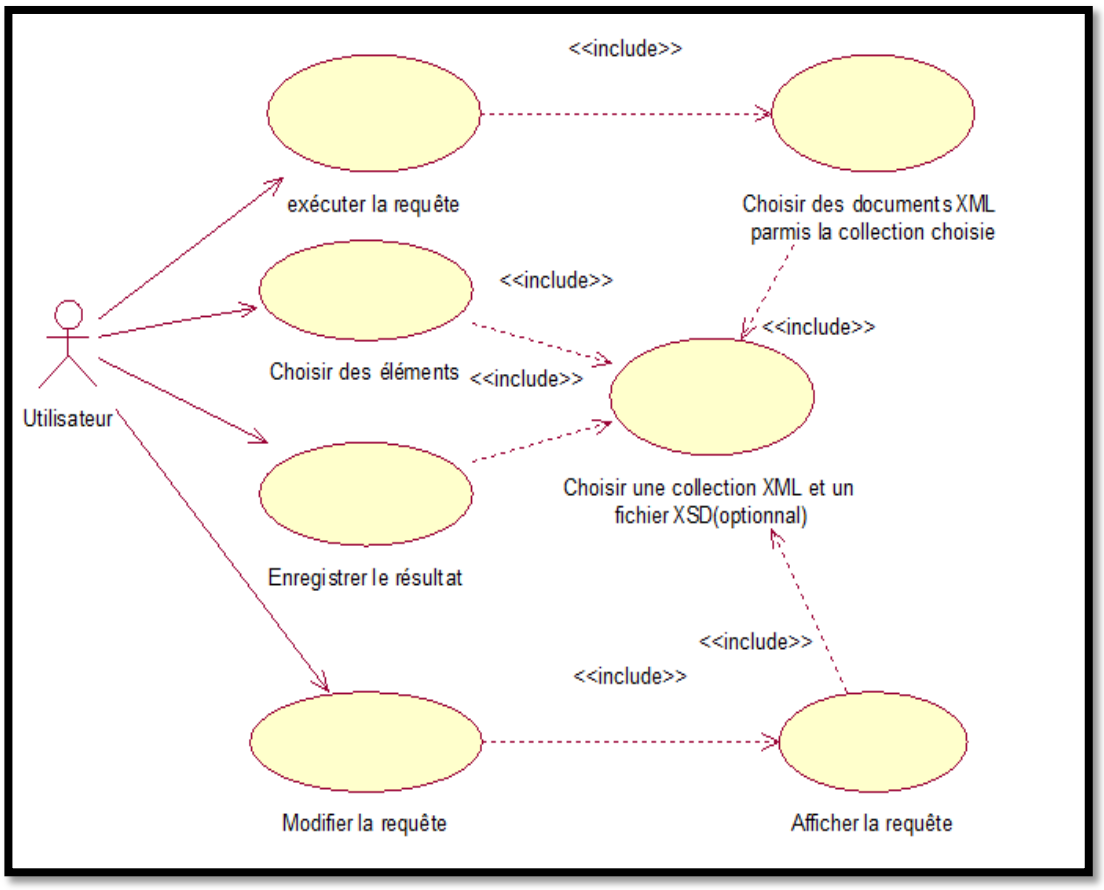

Figure 2.7 : Diagramme de cas d'utilisation

Cas d'utilisation	Fonctionnalités
Choisir une collection XML et un document XSD (optionnel)	Choix de la collection **XML** et du fichier **XSD** correspond (le fichier XSD est optionnel).
Choisir des éléments	Sélectionner des éléments (ou attributs) qui intéressent l'utilisateur parmi l'ensemble des éléments présentés Par la collection XML choisie.
Afficher la requête	Affichage de la requête **XQuery** afin de pouvoir la modifier.
Modifier la requête	Modification de la requête **XQuery** généré par l'application suite aux choix de l'utilisateur.
Choisir des documents XML	Choix des documents XML qui seront interrogés par la requête **XQuery** parmi l'ensemble des documents de la collection XML choisie
Enregistrer le résultat	Enregistrement du résultat de la requête dans un document **XML**.

Tableau 2.1: Liste des cas d'utilisations

Passons maintenant au diagramme de séquence :

2.3.3.b Diagrammes de séquence

- **Présentation des différents objets**

Concernant le diagramme de séquence, on a identifié trois objets qui sont :

Utilisateur : celui qui va utiliser l'application afin d'extraire l'information désirée depuis une collection XML et d'enregistrer le résultat dans un fichier texte.

Application : c'est notre application principale qui permette l'extraction de l'information depuis la collection XML.

Fichier XML: c'est un fichier dans lequel on va enregistrer le résultat de la requête XQuery exécutée.

- **Diagramme de séquence :**

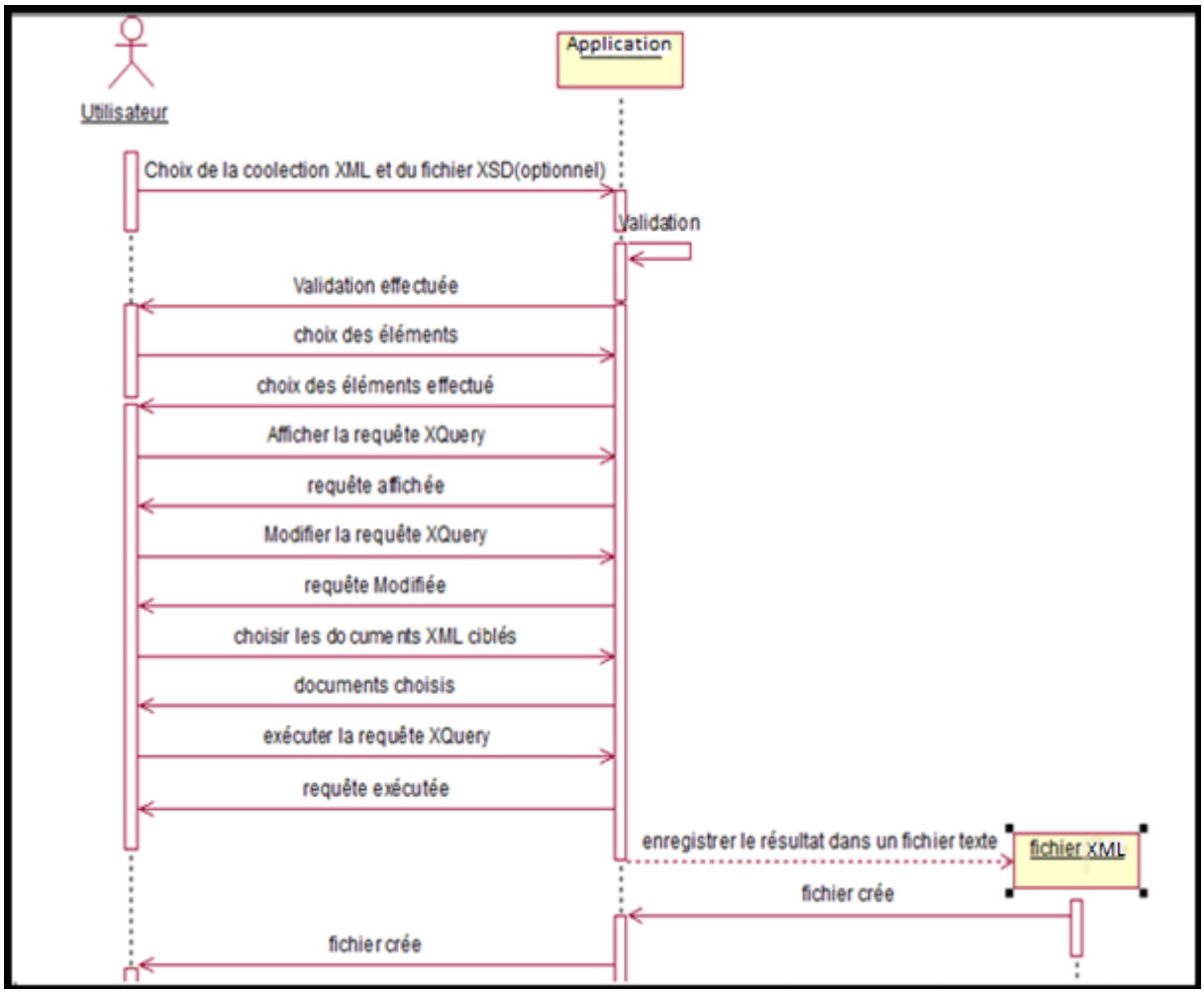

Figure 2.8 : Diagramme de séquence

Conclusion :

Ce chapitre nous a permis de comprendre le déroulement de l'application créée afin de bien saisir le mécanisme de ce programme. Il nous permet d'introduire le prochain chapitre qui s'occupera de la présentation et la description détaillée des interfaces de l'application.

Chapitre 3

Développement de l'application

Introduction :

Après avoir exprimé les besoins, spécifié et conçu l'application à réaliser, il ne reste qu'à l'implémenter. Dans ce chapitre nous présenterons l'environnement de travail, la réalisation de l'application ainsi qu'un test de ces fonctionnalités.

3.1. Environnement de travail :

Nous commençons cette première partie par la présentation de nos choix relatifs aux environnements de développement. Nous accordons un intérêt particulier au choix des outils et plateformes logicielles dont les performances conditionneront la pertinence de nos solutions de développement.

3.1.1. Environnement matériel

Nous avons utilisé pour la programmation, l'intégration et le test des modules logiciels à développer dans un ordinateur portable présentant les caractéristiques minimales suivantes :

- Modèle : Dell Inspiron
- Processeur : Intel ® Core™ 2 Duo CPU T6600 @2.20GHz 2.20 GHz
- Mémoire vive : 3.5 Go
- Disque Dur : 250 Go
- Carte réseau IP sans fil : 90-4C-E5-99-13-5A
- Système d'exploitation : Windows 7 Professionnel

3.1.2. Outils de développement logiciel

Notre application est censée fonctionner sur n'importe quel système d'exploitation : c'est pour cela que nous avons choisi la plateforme JAVA afin de la développer. En effet, c'est la portabilité de cette plateforme qui permettra à notre application d'être exécuté sur tout système d'exploitation tel que Windows, Linux, …

Pour développer sous la plateforme Java, nous avons utilisé les logiciels suivants :

- **NetBeans :**

NetBeans IDE est un environnement de développement intégré, libre, extensible, universel et polyvalent. Il permet de créer des projets de développement mettant en œuvre n'importe quel langage de programmation. Il permet la génération automatique des Interfaces graphique ce qui nous a aidé dans la réalisation de notre application.

- **JRE 6 : (Java Runtime Environment)**

JRE Désigne un ensemble d'outils permettant l'exécution de programmes Java sur toutes les plateformes supportées. JRE est constitué d'une JVM (Java Virtual Machine – Machine Virtuelle Java) qui représente le programme servant à interpréter le code Java et le convertit en code natif.

- **Rational Rose :**

Rational Rose est le leader Mondial comme outil de Modélisation UML
Ce logiciel propose par ailleurs des nombreux outils pour, faciliter la gestion des projets de développement.
Nous avons utilisé ce logiciel pour l'élaboration de deux types de diagrammes : diagramme de cas d'utilisation ainsi que le diagramme de séquence présentés dans le chapitre précédent.

3.2. Fonctionnement de l'application :

Notre application comporte deux Interfaces graphiques permettant en autre de réaliser les fonctions qui suivent :

3.2.1 Le choix de la collection XML et du fichier XSD :

Cette fonction est assurée par la première interface graphique intitulée **« Acceuil »** présentée par la Figure 3.1:

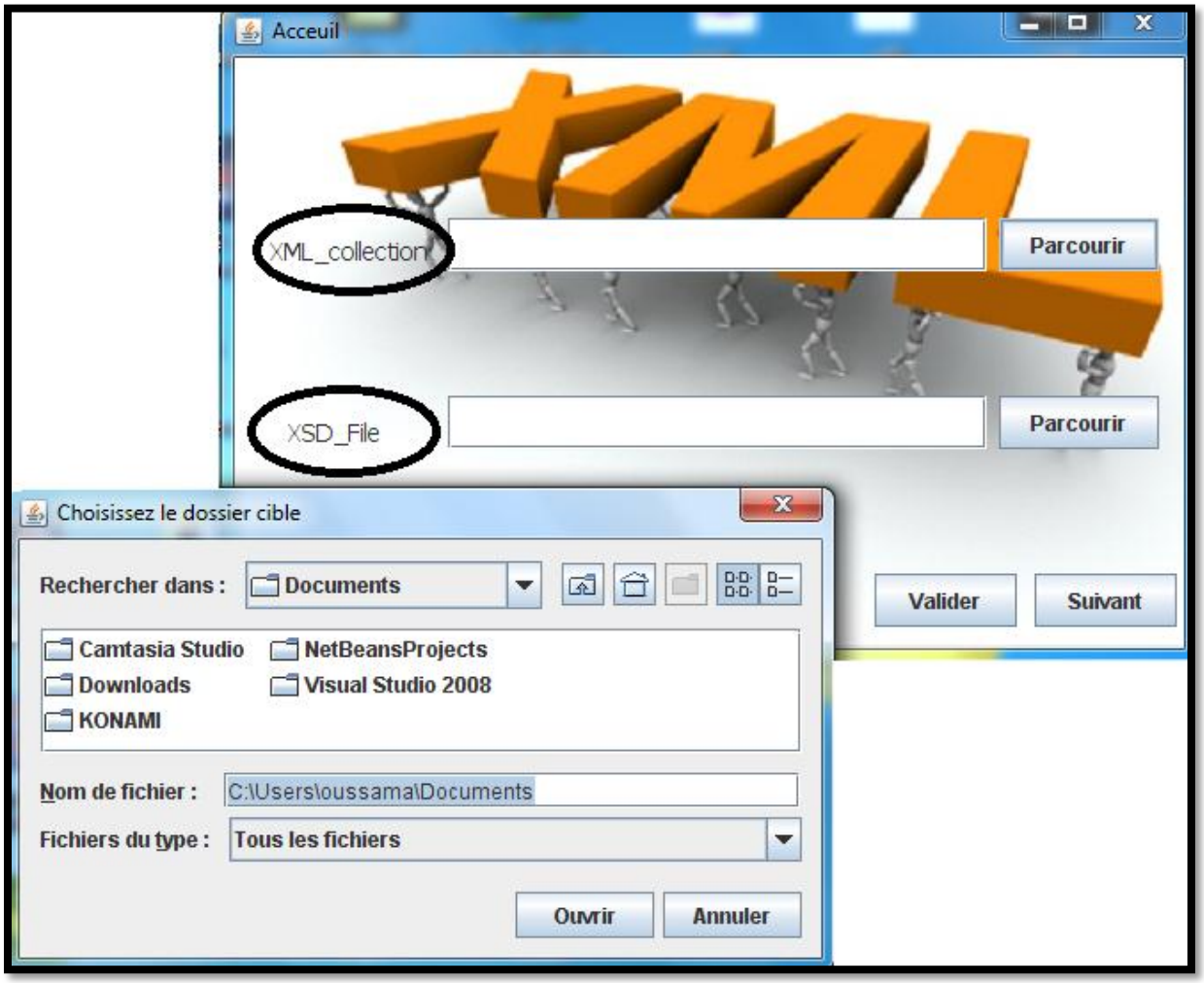

Figure 3.1 : L'interface graphique « Acceuil »

Comme la montre la figure cette interface présente deux champs dans lesquels on idique l'emplacement de la collection **XML** et du fichier **XSD** correspondant.

L'utilisateur peut vouloir indiquer seulement l'emplacement de la collection XML dans ce cas il appui directement sur le bouton « Suivant » pour passer à la deuxième interface graphique.

3.2.2 La validation de la collection XML par le fichier XSD :

Dans le cas où l'utilisateur indique le chemin du fichier XSD, on passe par la validation de la collection XML selon le fichier XSD afin de détecter les fichiers qui

présentent des anomalies par rapport à leur schéma et qui peuvent gêner l'utilisateur dans les étapes suivantes.

Donc l'utilisateur doit passer par le bouton « Valider » afin qu'il valide la collection XML avant d'appuyer sur le bouton « Suivant ».

Prenons un exemple pour expliquer d'avantages cette fonctionnalité : la Figure 3.2 montre la collection XML « Librairie » qui va servir d'exemple pour le reste de ce chapitre.

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

<livre categorie="policier">
<titre>7 jours et 7 nuits</titre>
<auteur>Pascal</auteur>
<pages>145</pages>
</livre>

<livre categorie="Science-fiction">
<titre>la nouvelle planete</titre>
<auteur>Nicolas</auteur>
<pages>155</pages>
</livre>

</Librairie>
```

Fichier 1

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

<livre categorie="autobiograhie">
<titre>Sept ans a Trenton</titre>
<auteur>Patrick</auteur>
<pages>135</pages>
</livre>

<livre categorie="roman">
<titre>Histoire de Trenton</titre>
<auteur>Sylvan</auteur>
<pages>125</pages>
</livre>

</Librairie>
```

Fichier 2

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

<livre categorie="autobiograhie">
<titre>L'enfance</titre>
<auteur>Patrick</auteur>
<pages>135</pages>
</livre>

<livre categorie="roman">
<titre>Les ennemis</titre>
<auteur>Sylvan</auteur>
<pages>125</pages>
</livre>

</Librairie>
```

Fichier 3

Figure 3.2 : la collection XML

Comme la montre la figure 3.2 notre collection XML contient trois documents XML . le schéma XSD correspondant à ces trois documents est présenté par la figure 3.3:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <xs:import namespace="http://www.w3.org/2001/XMLSchema-
instance" schemaLocation="xsi.xsd"/>
  <xs:element name="Librairie">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" ref="livre"/>
      </xs:sequence>
      <xs:attribute ref="xsi:noNamespaceSchemaLocation"
use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="livre">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="titre"/>
        <xs:element ref="auteur"/>
        <xs:element ref="pages"/>
      </xs:sequence>
      <xs:attribute name="categorie" use="required"
type="xs:string"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="titre" type="xs:string"/>
  <xs:element name="auteur" type="xs:string"/>
  <xs:element name="pages" type="xs:integer"/>
</xs:schema>
```

Figure 3.3 :le fichier XSD

Les trois documents de la collection choisie sont conformes au fichier XSD correspondant . C'est pour cela que lorsuque l'utilisateur appui sue le bouton « Valider » le message Présenté par la figure 3.4 lui sera affiché :

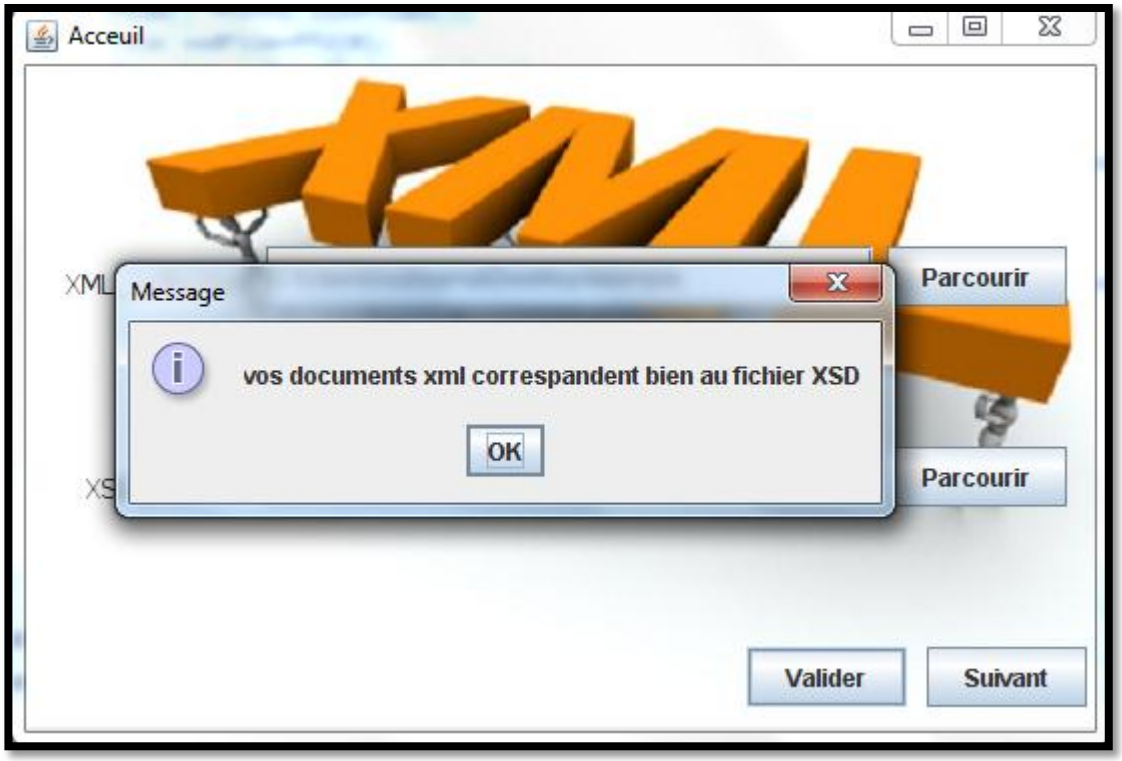

figure 3.4 : validation réussite de la collection XML par le fichier XSD

Maintenat on va traiter le cas échéant c'est-à-dire lorsque un des fichiers XML n'est pas conforme au fichier XSD. Pour cela , nous avons choisi d'ajouter la balise <NbChapitres>dans la description d'un livre à un des trois fichiers XML ce qui va nous donner le fichier de la figure 3.5 :

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

<livre categorie="autobiograhie">
<titre>Sept ans a Trenton</titre>
<auteur>Patrick</auteur>
<pages>135</pages>
<NbChapitre>5</NbChapitre>
</livre>

<livre categorie="roman">
<titre>Histoire de Trenton</titre>
<auteur>Sylvan</auteur>
<pages>125</pages>
<NbChapitre>6</NbChapitre>
</livre>

</Librairie>
```

Figure 3.5 : Fichier XML modifié

Puisque «<NbChapitres> n'est pas supportée par le schéma XSD , ce fichier posera un problème lors de la validation ce qui est illustré par la figure 3.6 :

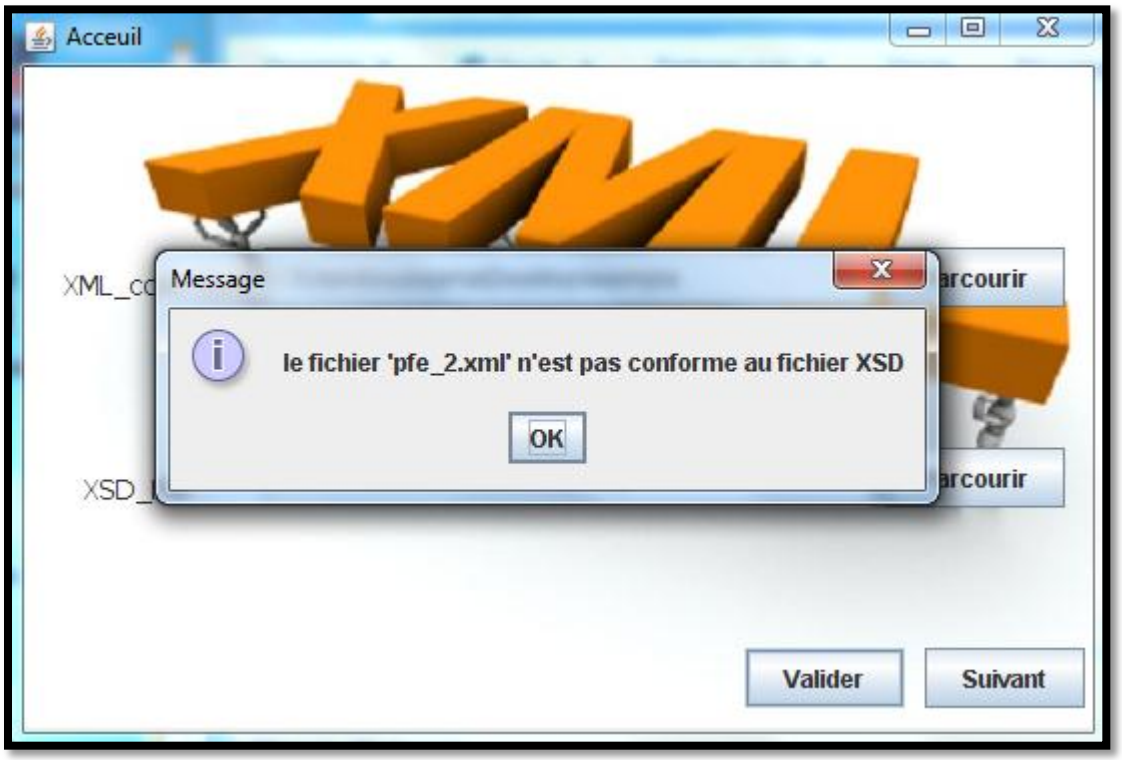

Figure 3.6 : Echec de la validation de la collection XML par le fichier XSD

On voit donc qu'il y a une petite fenêtre qui s'ouvre indiquant le nom du premier fichier XML qui n'est pas conforme au fichier XSD. Par conséquant l'utilisateur sera appelé à corriger le contenu du fichier XML ou bien à éviter de saisir le fichier XSD , s'il n'est pas sure que tous les documents XML sont lui conformes , afin de passer l'étape de la validation .

En validant la collection XML par le fichier XSD ou en saisissant seulement la collection XML ,l'utilisateur pourrait passer à la deuxième Interface qui est intitulé **« XQUERY »** (voir Annexe II : Manuel d'utilisation de l'interface « XQUERY »).

L’interface « XQUERY » permet de réaliser les fonctionalités sdécrites dans ce qui suit :

3.2.3 Présentation de la grammaire de la collection XML :

Dès que l’interface « XQUERY » s’ouvre elle nous indique la grammaire de la collection XML sous forme d’une arborescence afin que l’utilisateur puisse comprendre la relation entre les différents éléments des documents XML.

Prenons la même collection XML que celle choisie pour l’étape de la validation , l’arborescence est présentée dans l’onglet **« Grammaire_XML »** de l’interface **« XQUERY ».**

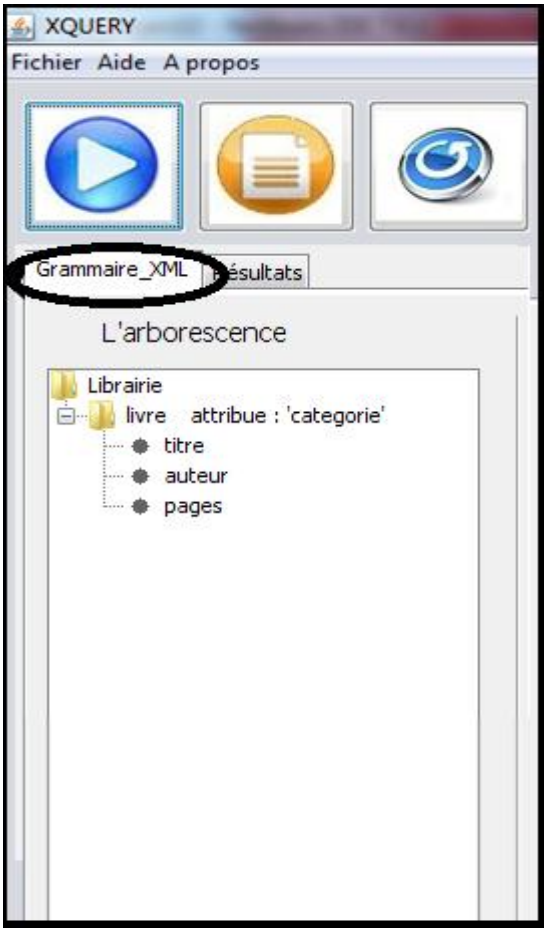

Figure 3.7 : L’arborescence de la collection XML

Pour l’établissement de l’arborescence , on a choisi de parser les documents XML un par un , c’est-à-dire qu’on commence par l’extraction de l’arborescence depuis le premier fichier XML, ensuite à chaque fois qu’on trouve un élement d’enrechissement dans les autres documents on l’ajoute en identifiant son élément père dans l’arborescence initiale.

Prenons toujours la même collection XML mais cette fois ci on va ajouter deux balises (« AgeAuteur » et « NbChapitre ») , une à chaque fichier sans toucher le premier fichier . . Ceci conduit aux documents montrés dans la Figure 3.8:

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

<livre categorie="autobiograhie">
<titre>Sept ans a Trenton</titre>
<auteur>Patrick</auteur>
<pages>135</pages>
<NbChapitre>5</NbChapitre>
</livre>

<livre categorie="roman">
<titre>Histoire de Trenton</titre>
<auteur>Sylvan</auteur>
<pages>125</pages>
<NbChapitre>6</NbChapitre>
</livre>

</Librairie>
```

Fichier 2

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

<livre categorie="autobiograhie">
<titre>L'enfance</titre>
<auteur>Patrick</auteur>
<pages>135</pages>
<AgeAuteur>45</AgeAuteur>
</livre>

<livre categorie="roman">
<titre>Les ennemis</titre>
<auteur>Sylvan</auteur>
<pages>125</pages>
<AgeAuteur>46</AgeAuteur>
</livre>

</Librairie>
```

Fichier 3

Figure 3.8 : Collection XML modifiée

Donc l'arborecsence présentée par l'interface **« XQUERY »** sera celle de la Figure 3.9:

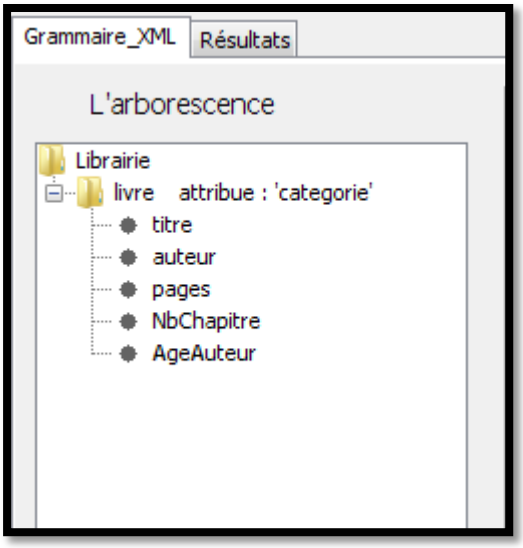

Figure 3.9 : Arborecsence de la collection après modification

Comme nous pouvons voir , Les éléments ont été ajoutés sous l'élément parent **« livre »** ce qui nous donne l'arborescence complète de la collection .

3.2.4. Choix des éléments et affichage de la requête XQUERY

L'interface « **XQUERY** » présente une table dans laquelle l'utilisateur va saisir ses choix ; notamment choisir les éléménts qui l'intéressent, l'ordre de leur affichage (ascendant ou descendant) ainsi que les conditions sur les éléments choisis qui seront tenues en compte dans la requête XQUERY générée à la fin .

Si en plus de la collection XML l'utilisateur choisie le fichier XSD correspondant, l'application identifie les types des éléments et les affiches dans la table afin de faciliter à l'utilisateur la saisie des conditions .

Donc pour les éléménts complexes (qui ont des éléments fils) on a choisi de bloquer leurs cases realtives à l'écriture des conditions afin d'éviter la génération des erreures lors de l'exécution de la requête .

Pour retourner un élément comme résultat il suffit que l'utilisatueur coche la case dans la ligne « show » .

L'utilisateur peut notamment saisir des conditions sur un élément sans cocher la case correspondante dans la ligne « show » », c'est-à-dire que cet élément ne sera pas choisi mais servira à filtrer les éléments choisis par rapport à la condition. En fait la requête XQuery raisonne toujours par rapport aux éléménts complexes et donc mêmes les conditions sur les éléments non demandés comme résultats seront tenus en compte pour filtrer les éléments choisis.

Dans la Figure 3.10, l'interface « XQUERY » présente un exemple de requête spécifiée par l'utilisateur dans la table des choix et spécifiée en XQuery après avoir été générée.

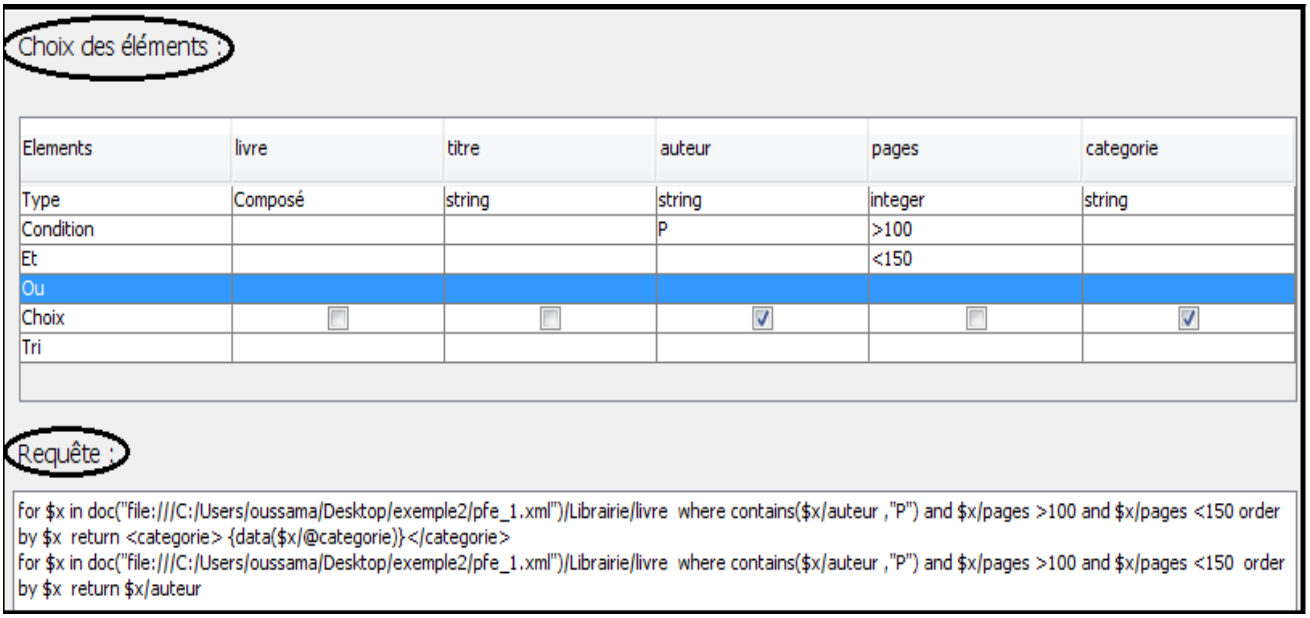

Figure 3.10 : Exemple de requête spécifiée dans l'interface XQUERY

Dans la requête de la Figure 3.10, l'utilisateur choisi des éléments des livres dont le nombre de pages varie entre 100 et 150 ; les éléments choisis sont tous les auteurs qui ont un prénom qui contient la lettre « P » et la catégorie de ces livres.

Il est également possible dans l'interface XQUERY d'afficher la requête générée suite au choix de l'utilisateur afin qu'il puisse entrer quelques modifications avant de l'exécuter. Cette fonctionnalité sert aussi à éditer une requête complète, si l'utilisateur le désire.

3.2.5. Affichage des résultats :

L'interface « XQUERY », dans son deuxième onglet « Résultats », permet à l'utilisateur d'afficher le résultat de la requête XQUERY générée ainsi que de choisir les documents XML depuis lesquels il veut extraire l'information.

Prenons l'exemple de requête précédent. Dans la Figure 3.11, le résultat est affiché pour les deux premiers documents, « pfe_1.xml » et « pfe_2.xml », étant explicitement exclu le document « pfe_3.xml ».

Figure 3.11 : Résultat de la requête XQuery

3.2.6. Enregistrement du résultat dans un fichier XML :

L'application nous permet aussi d'enregistrer le résultat obtenu dans un fichier XML. Donc pour l'exemple précédent on aura comme résultat final celui présenté dans la Figure 3.12.

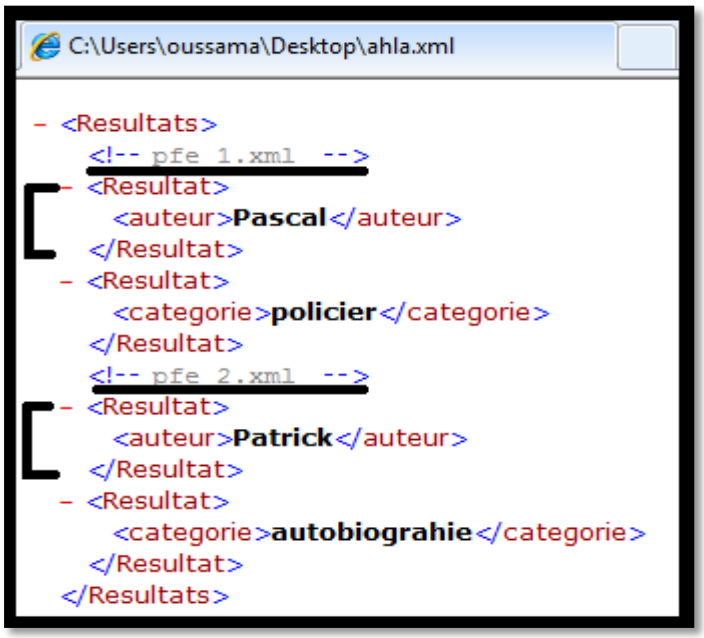

Figure 3.12 : Le résultat sous forme d'un fichier XML

Dans le fichier XML résultat, les différents documents XML, depuis lesquels on a extrait l'information, sont séparés par un commentaire où le nom du document correspondant est indiqué.

Aussi, chaque élément retourné est entouré par une balise « Resultat » pour rendre ce fichier traitable encore une fois par les machines, c'est-à-dire que l'on peut l'interroger par des requêtes XQuery pour filtrer encore les résultats obtenus ou même le transformer en un autre format, comme le HTML.

Conclusion :

Dans ce chapitre, nous avons présenté les choix de technologies et les spécifications de notre application, ainsi que ses différents modules, ses deux interfaces et ses fonctionnalités.

Conclusion Générale

Ce document a été rédigé aux termes du projet de fin d'études sur la thématique « Accès à l'information XML par des requêtes XQuery au travers du XSchema ».

La première phase du projet a été consacrée à la compréhension des différentes spécifications des langages XML , XSD et XQuery .

Dans la deuxième phase du projet, nous avons réalisé le premier module du projet : « le Générateur d'Interfaces » qui consiste à générer une interface à l'utilisateur à partir de la collection XML passée en entrée afin de permettre à l'utilisateur d'effectuer ses choix et d'afficher la requête XQuery correspondante pour qu'il puisse la modifier, s'il le désire.

La troisième phase du projet a consisté à réaliser le deuxième module du projet, « le Générateur de Requêtes », qui consiste à exécuter les requêtes XQuery générées moyennant l'API du moteur de recherche Saxon. Celui-ci permet d'interroger les documents XML, d'en extraire les données correspondantes et de les stocker sous forme d'un nouveau document XML résultant.

Ce projet nous a permis de perfectionner nos aptitudes en termes de programmation sur le langage de programmation JAVA en manipulant les différentes API relatives au parcours des fichiers XML et à la génération des requêtes XQuery. En plus, ce projet nous a permis de comprendre les différents spécifications relatives au documents XML et surtout le fait qu'ils peuvent être considérés comme une forme de base de données, et par la suite on peut les interroger par des requêtes XQuery et XPath .

Bibliographie et Netographie

[1] http://fr.wikipedia.org/

[2] http://www.w3schools.com/

[3] http://fr.wikipedia.org/wiki/Query_By_Example

[4] http://saxon.sourceforge.net/saxon7.5/api-guide.html

[5] The invocation and definition of tables and forms. In VLDB ’75: Proceedings of the 1st International Conference on Very Large Data Bases, pages 1–24, New York, NY, USA.

[6] http://www.w3schools.com/xquery/default.asp

ANNEXES

Annexe 1 :

Partie I : Les composants d'un document XML :

Un document XML est structuré en utilisant les composants suivants :

- La déclaration de document XML : <? ...?> .
- Les commentaires.
- Les nœuds éléments : gèrent la structuration des données d'un document XML, un peu à la manière des répertoires qui servent à l'organisation des fichiers.
- Un élément est composé de balise de début, contenu, balise de fin.
- Les attributs : Un attribut est un couple (clé, valeur) associé à la définition d'un élément. Localisé dans la balise ouvrante de l'élément, l'attribut est complémentaire de l'élément de par son rôle au sens où il ajoute une information à l'élément ou bien encore le complète dans sa définition.
- Un document XML est destiné à être compris par un humain aussi bien qu'une machine (facile à traiter par une application). Produit par un humain, le document XML est dit orienté document. Produit par une machine, il est dit orienté données [2].

Voici un exemple de document XML:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Date de création : 30/09/07 -->
<cours titre="XML">
<intervenant nom="alexandre brillant">
</intervenant>
<plan>
Introduction
XML et la composition de documents
</plan>
</cours>
```

Partie II : Modèle du XML Schéma :

Un XML Schema[2]est un document XML avec un ensemble de composants qui permettent de spécifier ce qui suit :

- Déclarer des éléments par la balise <XSD:element/>

  La balise <XSD:element/> accepte deux attributs: name (nom de l'élément) et type (type de l'élément).

```
<xsd:element name="texte" type="xsd:string" />
```

- Déclarer des attributs par la balise <XSD:attribute/>

  La balise <XSD:attribute/> accepte deux attributs: name (nom de l'attribut) et type (type de l'attribut).

```
<xs:attribute name="lang" type="xs:string"/>
```

- Définir le type d'élément schema. Cet élément est la racine de chaque schéma XML:

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

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

- Définir des types simples (valeurs constituées uniquement à partir d'une chaîne de caractères).

- Définir des types complexes : déclarés grâce à la balise <xs:complexType>/

  On distingue quatre formes de types complexes :

  - éléments vides (*Empty*) ;
  - éléments composés uniquement d'autres éléments ;
  - éléments composés uniquement de texte ;
  - éléments composés à la fois de texte et d'éléments (*Mixte*).

L'exemple suivant illustre un type complexe d'éléments, qui est le type ‘élément composé d’autres éléments’.

XML :

```
<employee>
 <firstname>John</firstname>
 <lastname>Smith</lastname>
</employee>
```

XSchema :

```
<xs:element name="employee">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="firstname" type="xs:string"/>
   <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
</xs:element>
```

- Définir des restrictions sur la valeur d’un type simple, comme par exemple :

Restriction sur le type entier

```
<xs:element name="age">
 <xs:simpleType>
  <xs:restriction base="xs:integer">
   <xs:minInclusive value="0"/>
   <xs:maxInclusive value="120"/>
  </xs:restriction>
 </xs:simpleType>
</xs:element>
```

Restriction sur le type string

```
<xs:element name="car">
 <xs:simpleType>
  <xs:restriction base="xs:string">
   <xs:enumeration value="Audi"/>
   <xs:enumeration value="Golf"/>
   <xs:enumeration value="BMW"/>
  </xs:restriction>
 </xs:simpleType>
</xs:element>
```

Annexe II : Manuel d'utilisation de l'interface « XQUERY»

- **Lancement de la requête et affichage de résultat :**

Pour lancer l'application il suffit d'appuyer sur le bouton « exécuter » ou bien d'accéder à « Fichier » dans la barre de menu et cliquer sur « Démarrer ». Ces boutons sont visualisés dans la Figure A2.1.

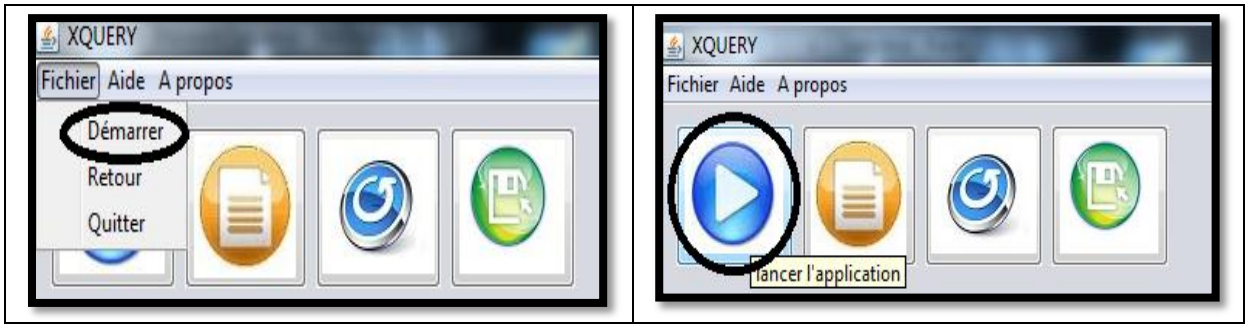

Figure A2.1 : lancer l'application :

- **Affichage de la requête :**

Pour afficher la requête dans l'anglet « Grammaire_XML », on appuie sur le bouton « orangé » comme le montre la Figure A2.2 :

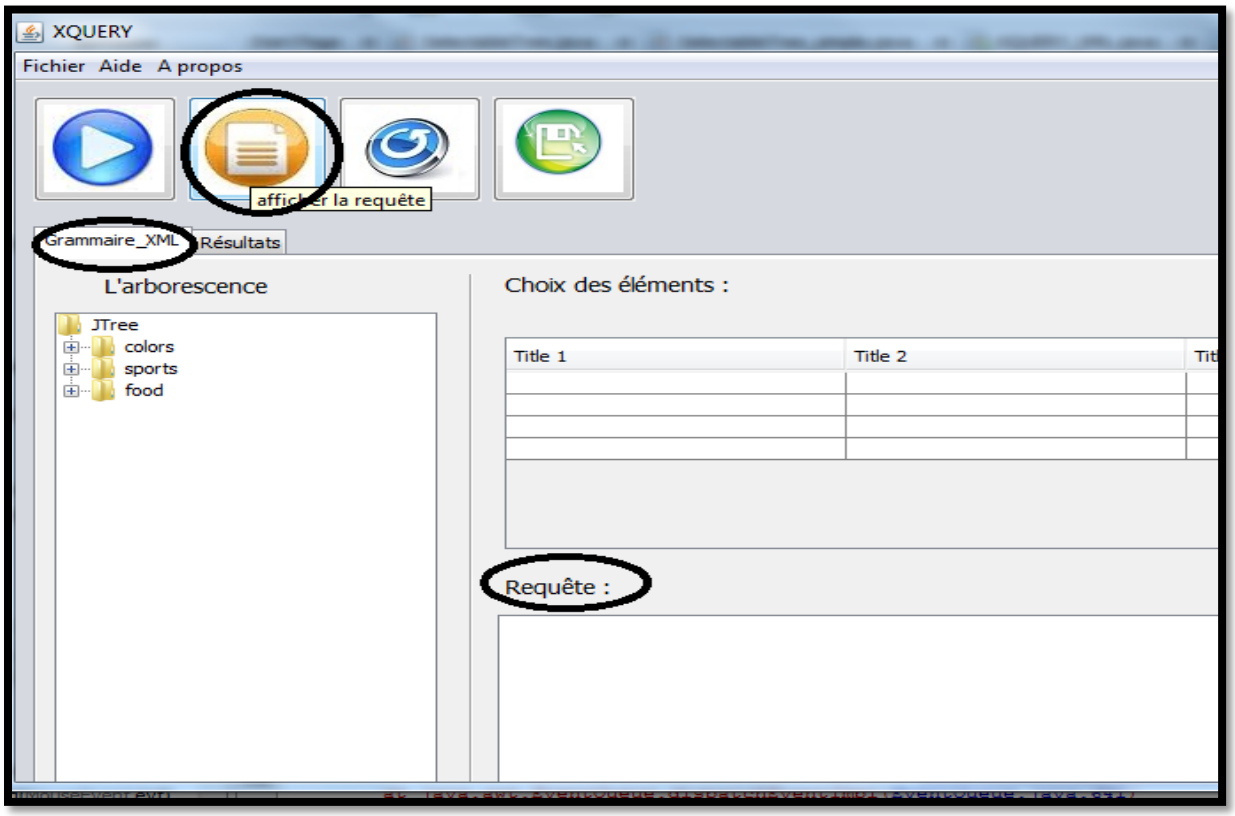

Figure A2.2 : Affichage de la requête

- **Réinitialisation des choix de l'utilisateur :**

Pour la réinitialisation du choix de l'utilisateur on appuie sur le bouton « réinitialiser » (Figure A2.3).

Figure A2.3 : Réinitialisation des choix de l'utilisateur

- **Enregistrement des résultats obtenus :**

  Pour enregistrer les résultats dans un fichier XML on appuie sur le bouton « enregistrer le résultat » (Figure A2.4).

Figure A2.4 : Enregistrements des résultats

- **Autres fonctionnalités :**

En accédant à « Fichier » dans la barre de menu on peut quitter l'application en appuyant sur « Quitter ». L'utilisateur peut également revenir à l'interface « Accueil » en appuyant sur « Retour ».

Figure A2.5 : Menu « Fichier » de l'interface « XQUERY ».