XFIRM : Un Modèle Flexible de Recherche d'Information pour le stockage et l'interrogation de documents XML

Karen Sauvagnat

IRIT – SIG-RFI
118 route de Narbonne
31 062 Toulouse Cedex 4
sauvagna@irit.fr

RESUME : *Les utilisateurs recherchant une information précise ne souhaitent pas la voir noyée aux milieux d'autres sujets, comme cela peut être le cas dans de grands documents. Les documents XML, par leur structure même, permettent de traiter l'information qu'ils contiennent à un niveau de granularité autre que celui du document tout entier. Deux approches s'affrontent pour la recherche d'information (RI) dans des documents XML. La première est basée sur des méthodes issues de la communauté des bases de données, alors que la seconde étend des techniques utilisées pour RI traditionnelle et permet d'associer des valeurs de pertinences aux unités d'information retournées. Cet article présente le modèle XFIRM, combinant les avantages des deux approches. Le modèle est basé sur un modèle de représentation et de stockage des données complet et sur un langage de requête simple, permettant à l'utilisateur de formuler son besoin à l'aide de simples mots-clés ou de manière plus précise, et ce en intégrant des contraintes sur la structure des documents. Grâce à la flexibilité apportée par la structure d'index proposée, de nombreux modèles de recherche d'information pourront être expérimentés.*

ABSTRACT. *Users looking for precise information do not want it to be drowned by noisy subjects, as it can be the case in long documents. Thanks to their structure, XML documents allow Information Retrieval (IR) systems to retrieve information units, that are not necessarily whole documents. Two main approaches have been proposed in the literature : database-oriented approaches and IR-oriented approaches, which extend traditional methods and allow to associate relevance values to the returned information units. This paper describes the XFIRM model, combining the advantages of both approaches. The model is based on a complete data representation and storing model. The XFIRM query language aims at allowing users to express their need with a set of keywords and/or in a more precise way with structural conditions. Thanks to the flexibility of the index structure, many information retrieval models could be supported.*

MOTS-CLÉS : *XML, recherche d'information, langage de requête, modèle de représentation*

KEYWORDS: *XML, information retrieval, query language, representation model*

1. Introduction

Le développement considérable qu'a connu Internet ces dernières années, notamment avec l'apparition du World Wide Web, a conduit à une croissance quasi-exponentielle du nombre d'utilisateurs de ce type de ressources, mais également du nombre de documents mis à disposition. Afin de faciliter l'échange et la standardisation des données, le W3C a introduit XML (eXtensible Markup Language) (W3C, 2000) (Bradley, 2001), qui est un format de texte simple et flexible dérivé de SGML (Standard Generalized Markup Language). XML est un format de données semi-structuré, qui sépare le contenu des documents des instructions de présentations. Ces caractéristiques laissent présager une explosion du nombre de documents au format XML.

Dans ce contexte, la question de la recherche d'information dans des corpus XML devient cruciale. En effet, pour mieux répondre aux différents besoins des utilisateurs et mieux valoriser l'ensemble des informations disponibles, des systèmes de recherche d'information spécifiques doivent être développés.

Dans cet article, nous présentons XFIRM (XML Flexible Information Retrieval Model), un système flexible pour la recherche d'information dans des documents XML. Ce système, basé sur un modèle d'indexation complet, permet de traiter tout aussi bien des requêtes simples à base de mots clés portant sur le contenu des documents que des requêtes plus précises portant sur le contenu et la structure. Dans la section 2 de cet article, nous décrivons différentes problématiques soulevées par la recherche d'information dans des documents structurés et les solutions qui ont été proposées. A partir de cette discussion, nous présentons notre système XFIRM, à savoir le modèle de représentation des informations proposé, le langage de requête associé et enfin les différentes stratégies de recherche possibles.

2. Problématique et état de l'art

2.1. Terminologie

XML est un langage de balisage similaire à SGML. Le texte est encadré par *des balises de début* et des *balises de fin* et le *nom* des balises permet de donner des informations sur la sémantique de leur *contenu*. Les différentes balises peuvent être imbriquées, comme dans les figures 1 et 2, qui nous serviront par la suite de scénario d'exemple pour la description du système XFIRM. Ces deux documents XML sont issus d'une collection de documents représentant au format XML les livres d'une bibliothèque. Le premier document concerne une pièce de théâtre et le second un roman.

```
<pièce date-publication= « 1600 »>
<titre>Le songe d'une nuit d'été</titre>
<auteur>William Shakespeare</auteur>
<texte>
<acte></acte>
<acte numéro= « 2 »>
<scene numero= « 1 »>
<commentaires> … </commentaires>
<texte>Puck : Et bien esprit, où errez
                vous ainsi ?
La fée : par la colline…</texte>
</scene>
…
</acte>
…
</texte>
</pièce>
```

Figure 1: *Document 1: songe.xml*

```
<roman date-publication= « 1987 »>
<titre>La fée carabine</titre>
<auteur>Daniel Pennac</auteur>
<texte>
<chapitre numéro= « 1 »>
<titre>La ville, une nuit</titre>
<texte>C'était l'hiver…</texte>
</chapitre>
…

</texte>
</roman>
```

Figure 2: *Document 2: fée.xml*

Les documents XML peuvent être représentés sous forme d'arbre, encore appelé arbre DOM (Document Object Model). Chaque *nœud feuille* est de type #PCDATA (plein texte) et les autres *nœuds* sont le point de départ de sous-arbres.

On appelle *élément* tout sous-arbre issu de l'arbre DOM d'un document. Dans la suite de l'article, les termes *nœud* et *élément* seront utilisés indifféremment pour désigner un sous-arbre de l'arbre DOM d'un document.

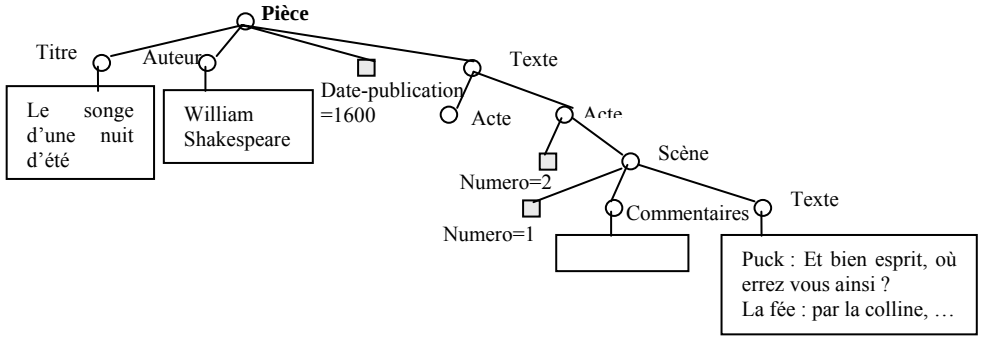

Figure 3: *Arbre DOM du document songe.xml*

Le langage XPath (W3C, 1999) est un langage d'expression qui permet de définir précisément dans l'arbre XML la position d'un nœud :

Syntaxe	Description
Pièce	*Les éléments fils du nœud courant de type pièce*
Acte[1]	*Le premier fils du nœud courant de type acte*
Pièce/texte/acte[2]/scene[1]	*La première scène du deuxième acte de la pièce*
Pièce//acte	*Les éléments actes dans la descendance de pièce*

Tableau 1 : *Quelques exemples de syntaxe XPath*

Des *attributs* peuvent être assignés aux éléments, et sont alors exprimés dans la balise de début : <roman date-publication= « 1987 »>…</roman>. Ici le *nom* de l'attribut est date-publication et la *valeur de l'attribut* est 1987.

La syntaxe des documents XML peut être décrite dans une DTD (Document Type Definition). Les deux documents *songe.xml* et *fée.xml* suivent ici des DTDs différentes, suivant qu'ils soient structurés comme des pièces de théâtre ou comme des romans.

2.2. Problématique

Le but des systèmes de recherche d'information (SRI) est de répondre au besoin en information de l'utilisateur (Salton, 1970). Ce dernier s'intéresse rarement à une représentation ou à une structuration précise d'un document, il veut du contenu. S'il est capable de fournir des informations supplémentaires dans sa requête, par exemple des informations structurelles, la réponse fournie par le système ne devra être que plus précise.

Les SRI classiques, tant dans leur modèle de représentation des données que dans les résultats qu'ils renvoient, traitent les documents dans leur globalité. Les SRI traitant les documents structurés ont la possibilité de traiter l'information avec une autre granularité. En effet, l'utilisateur recherche parfois une information précise, et il ne désire pas que cette information soit « noyée » au milieu d'autres sujets. Bien avant l'apparition d'XML, des travaux concernant la granularité de l'information à renvoyer à l'utilisateur ont été présentés. Ces travaux ont cherché à découper un document textuel en entités plus petites. L'intérêt de considérer une granularité plus fine est de traiter des documents que l'on va supposer homogènes (Moffat et *al*., 1993) (Hearst 1997).

Les documents XML permettent/imposent ainsi aux SRI de retrouver des unités d'information, qui ne sont pas nécessairement les documents entiers. Dans (Chiaramella et *al*., 1996), on trouve « le principe de recherche dans les documents structurés » : *un système devrait toujours retrouver la partie la plus spécifique*[1] *d'un*

[1] Un élément est spécifique à une requête si tout son contenu concerne la requête.

document répondant à une requête. Cette définition suppose que le système recherche d'abord des documents entiers répondant de manière exhaustive[2] à une requête, puis extrait de ces documents les unités d'information les plus spécifiques. La plupart des SRI traitant les documents structurés permettent une recherche directe des unités d'information, sans passer au niveau de granularité document entier. Le principe de recherche dans les documents structurés pourrait donc être étendu ainsi : *un système devrait toujours retrouver l'unité d'information la plus exhaustive et spécifique répondant à une requête.*

L'accès aux documents XML a été appréhendé selon deux angles principaux (Fuhr *et al*., 2001):

- L'approche orientée données utilise les documents XML pour échanger des données sous une forme structurée (comme par exemple des bases de données entières) ;

- L'approche orientée documents se focalise sur des applications considérant les documents structurés d'une manière traditionnelle, c'est à dire que les balises servent uniquement à décrire la structure logique des documents.

L'approche orientée données nécessite l'utilisation de techniques développées par la communauté des bases de données, alors que l'approche orientée documents est prise en charge par la communauté de la recherche d'information (RI).

2.3. Les approches orientées SGBD

Aujourd'hui, la plupart des travaux concernant le stockage, l'indexation, l'interrogation et la recherche sur des documents XML proviennent du travail de la communauté des bases de données sur les données semi-structurées. Ces approches orientées SGBD réalisent en surcouche l'intégration de XML en base objet-relationnelle. La surcouche transforme les documents XML en tables et vice-versa. Les approches EDGE, BINARY (Florescu et *al.*, 1999), et Xpath Accelerator (Grust, 2002) sont basées sur des modèles de mapping génériques, c'est à dire qu'elles créent un schéma générique de la base de données qui reflète le modèle de données du format XML. L'approche STORED (Deutsch et *al*., 1999) se concentre plus sur la structure. De nombreux langages ont été développés pour interroger ces modèles. Bien que fondés sur des approches différentes, tous les langages mixent des prédicats sur les méta-données (les balises des documents XML) et les données (les valeurs). Parmi ces langages, on peut citer UnQL (Buneman et *al.*, 1996), Lorel (Abiteboul et *al.*, 1997), XML-QL (Levy et *al.,* 98), XQL (Robie et al., 1998), XML-GL (Ceri et *al.,* 1999), QUILT (Chamberlin et *al.,* 2000), et Elixir (Chinenyanga et *al*., 2001). Xquery (W3C, Nov. 2003) est le langage proposé par le W3C. Encore en révision (c'est un Working Draft), quelques caractéristiques doivent être finalisées. Notons cependant qu'il peut être considéré comme un

[2] Un élément est exhaustif à une requête s'il contient toutes les informations requises.

surensemble de SQL, qu'il intègre un prédicat *contains* pour la recherche par mots-clés et qu'il permet d'exprimer des chemins indéterminés ou partiellement connus, tout comme XPath. Un Working Draft (W3C, Fev. 2003) propose en outre d'étendre les caractéristiques de recherche plein-texte, mais celle-ci se fera toujours sur des éléments spécifiés par l'utilisateur.

Si ces approches semblent être capable de traiter avec efficacité la structure des documents XML, elles sont cependant limitées, notamment pour le traitement de la partie textuelle, donc des mots-clés, des documents XML. Ces mots clés sont traités de façon binaire (présent / absent) alors qu'il a été démontré en recherche d'information textuelle (Salton et *al.,* 1984) (Robertson, 1977) que la prise en compte des poids des mots-clés dans un document est primordiale, voire nécessaire. Ceci permet de mesurer un degré de pertinence d'un document (ou d'une partie de document) vis-à-vis d'une requête et donc de renvoyer à l'utilisateur une liste triée de résultats.

2.4. Les approches orientées Recherche d'Information

Les documents XML peuvent aussi être considérés comme une collection de documents textes possédant des balises et des relations entre ces balises. Les modèles de recherche utilisés dans la RI traditionnelle doivent donc être étendus pour prendre en compte la structure et la sémantique de ces documents. Le *modèle booléen* étendu utilise un nouvel opérateur binaire non commutatif appelé « contains ». La première opérande est de type Xpath (W3C, 1999) et la seconde est une expression booléenne. Le modèle permet ainsi aux requêtes d'être complètement spécifiées en terme de contenu et d'information structurelle (Hayashi et *al.*, 2000). Une extension naturelle *du modèle vectoriel* est de séparer l'information structurelle de l'information sur le contenu. Cette proposition est cependant peu utilisée, et à la place, l'idée de proximité doit être étendue pour évaluer les relations entre la structure et le contenu. Dans ce cas, le langage de requête utilisé oblige que chaque terme de l'index du contenu soit encapsulé par un ou plusieurs éléments. Le modèle peut être généralisé en permettant l'agrégation des scores dans la hiérarchie (Fuller et al., 1993). Schlieder et Meuss (Schlieder et *al*., 2002) proposent le modèle ApproXQL, qui intègre la structure des documents dans la mesure de similarité du modèle vectoriel. Leur modèle de requête est basé sur la correspondance d'arbres : cela permet de formuler des requêtes sans connaître la structure exacte des données. Enfin, le *modèle probabiliste* est appliqué aux documents XML dans (Lalmas, 1997), (Wolff et *al*., 2000) et (Fuhr et *al.*, 2001).

Le langage de requête XIRQL (Fuhr et *al.,* 2001) étend les opérateurs du langage Xpath avec des opérateurs pour la recherche « orientée pertinence ». D'autres opérateurs permettent d'effectuer des recherches vagues sur le contenu non-textuel. Par exemple, il est possible d'effectuer des recherches probabilistes sur le contenu numérique d'un élément. Le tri des documents est effectué selon la

probabilité décroissante que le contenu est celui spécifié par l'utilisateur dans sa requête. XIRQL utilise les fonctionnalités de correspondance du langage XPath. Cependant, les utilisateurs ne connaissent pas nécessairement la structure des documents de la collection qu'ils interrogent, ou les documents de la collection peuvent suivre différents schémas. Pour répondre à de tels problèmes, le langage de requête du moteur de recherche XLL (Theobald et *al.,* 2002) offre des fonctionnalités pour la recherche orientée-pertinence de chemins, c'est à dire que la recherche est effectuée avec des conditions de chemins *vagues*. XLL repose sur une syntaxe SQL (select-from-where). Le moteur de recherche XLL est basé sur le modèle vectoriel et utilise une fonction de tri basée sur *tf* et *idf*. XIRQL utilise aussi ce type ce pondération, qui ne permet pas de retrouver de manière fiable des parties de documents XML (même en utilisant des techniques de propagation de poids) (Grabs et *al*., 2002). En effet, les occurrences des termes ne suivent plus forcément une loi de Zipf (Zipf, 1949). Le nombre de répétitions des termes peut être (très) réduit dans les documents XML et l'utilisation d'*idf* (Inverse Document Frequency) n'est pas forcément appropriée. Dans (Wolff et *al*., 2000) et (Grabs et *al*., 2002), les auteurs proposent l'utilisation de l'*ief* (Inverse Element Frequency) et ils calculent un nouveau poids pour les termes, basé sur la fréquence du terme dans chaque élément et l'*ief*.

2.5. Discussion

Les approches orientées SGBD permettent de traiter efficacement des requêtes portant sur la structure des documents, mais ne calculent cependant pas de valeur de pertinence sur les résultats. De plus, une connaissance parfaite de la structure des documents est nécessaire à l'utilisateur pour pouvoir formuler des requêtes et les documents d'une même collection doivent tous respecter la même DTD. L'utilisation de fonctions déjà existantes dans les SGBD (pour les jointures, les projections, etc) permet néanmoins une implémentation relativement simple.

Les approches orientées RI assignent des valeurs de pertinence à chaque unité d'information, ce qui permet de renvoyer à l'utilisateur des listes triées de résultats. Cependant, la plupart de ces approches utilisent des mesures de similarité basées sur *tf* et *idf* (Hayashi et *al*., 2000), (Schlieder et *al*., 2002), (Fuhr et *al.*, 2001), (Theobald et *al.,* 2002), alors que l'emploi d'*ief* semble plus recommandé pour traiter des documents XML (Wolff et *al*., 2000), (Grabs et *al*., 2002). De plus, les modèles proposés traitent rarement les unités d'information répondant de manière partielle aux requêtes sur leur structure. Seules les unités d'information répondant de manière exacte aux requêtes sont renvoyées aux utilisateurs. Enfin, le mode d'interrogation des systèmes est souvent complexe, et les utilisateurs ne peuvent pas formuler de requêtes composées de simples mots-clés (Theobald et *al.,* 2002).

L'approche proposée par Grabs (Grabs et *al*., 2002) combine l'approche centrée données et l'approche centrée documents, mais recrée les index contenant les

informations nécessaires aux techniques de RI à chaque requête, ce qui ne permet pas de traiter les requêtes de façon rapide.

Le système XFIRM que nous présentons ici combine aussi l'approche orientée documents et l'approche orientée données. Ces trois principales contributions sont :

- un ***modèle de représentation des données*** basé sur des approches orientées SGBD mais gardant néanmoins en mémoire toutes les informations nécessaires pour appliquer différents modèles de RI, et permettant de traiter des collections de documents ayant des DTDs différentes ;

- un ***langage de requête associé*** permettant à l'utilisateur de formuler des requêtes simples à base de mots-clés, mais aussi des requêtes plus précises portant sur la structure des documents et leur contenu, et ce même s'il ne connaît pas la structure exacte des documents;

- ***l'implémentation possible de nombreux modèles de RI*** pour retrouver les unités d'information les plus pertinentes aux requêtes des utilisateurs, et ce grâce à la flexibilité apportée par le modèle d'index proposé (utilisation d'*idf* et *ief*).

3. Le modèle de représentation des données XFIRM

3.1. Le modèle logique de représentation des données

Le modèle de représentation des données que nous présentons a pour but principal de permettre la navigation dans la structure en arbre des documents XML et de représenter le contenu de cette structure, afin de pouvoir appliquer ensuite différents modèles de RI.

Un document structuré ds_i est un arbre, composé de nœuds « simples » n_j ,de nœuds feuilles nf_j et d'attributs a_j.

Document Structuré *:* $ds_i = (arbre_i)=(\{n_j\}, \{nf_j\}, \{a_j\})$

Notons que cette représentation est une simplification du modèle de données de Xpath et Xquery présenté dans (W3C, Mai 2003), dans lequel un nœud peut être un *document*, un *élément*, un *attribut*, du *texte*, un *espace de noms*, une *instruction* ou alors un *commentaire*.

Afin de pouvoir facilement naviguer dans l'arbre et déterminer rapidement les relations ancêtres-descendants ainsi que permettre l'accès rapide à un nœud, nous proposons la représentation suivante des nœuds et des attributs, basée sur l'approche XPath Accelerator (Grust, 2002).

Nœud : $n_j = (pre, post, parent, attribut)$
Nœud feuille *:* $nf_j = (pre, post, parent, \{t_1, t_2, \ldots t_n\})$
Attribut : $a_j = (pre, val)$

Un nœud est défini grâce à ses valeurs de pré-ordre et post-ordre (*pre* et *post*), la valeur de pré-ordre de son nœud parent (*parent*), et selon que ce soit un nœud simple ou un nœud feuille, par un champ indiquant la présence d'attributs (*attribut*) ou les termes qui le composent ($\{t_1, t_2, \ldots t_n\}$). Un attribut est définit la valeur de pré-ordre du nœud auquel il se rattache (*pre*) et par sa valeur *(val)*.

Les valeurs de pré-ordre et post-ordre sont assignées aux nœuds comme suit : en chargeant un nouveau document, on effectue un parcours séquentiel de la représentation en arbre du document structuré. Un parcours préfixé permet d'assigner à chaque nœud visité une valeur croissante de pré-ordre (*pre*) avant que ses nœuds descendants ne soient aussi récursivement visités de gauche à droite. D'une manière inverse, la valeur de post-ordre (*post*) d'un nœud lui est assignée lors d'un parcours postfixé, c'est à dire une fois que tous ses nœuds descendants ont été visités de gauche à droite.

La figure ci-dessous donne un exemple pour le document *fée.xml* .

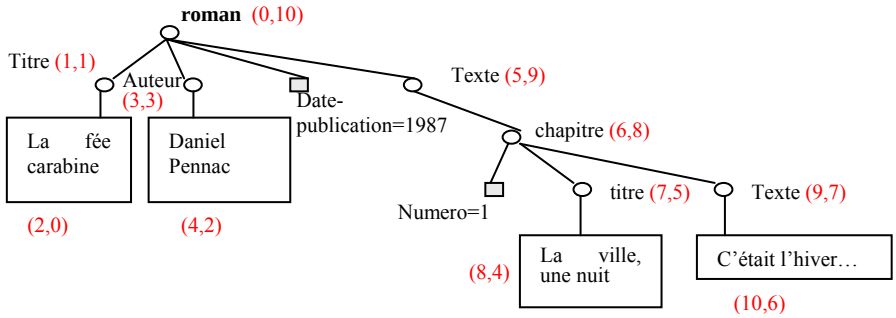

Figure 4: *Valeurs de pré-ordre et post-ordre assignées aux nœuds du document XML fée.xml*

Si l'on transpose tous les nœuds dans un espace à deux dimensions basé sur les coordonnées de pré-ordre et post-ordre, on peut exploiter les propriétés suivantes illustrées par l'exemple de la figure 5. Etant donné un certain nœud *n* (le nœud roman[1]/texte[1]/chapitre[1]/titre dans l'exemple ci-dessous):

- tous les ancêtres de *n* sont au-dessus à gauche de la position de *n* dans le plan

- tous ses descendants sont en dessous à droite

- tous les nœuds le précédant dans la lecture séquentielle du document sont en-dessous à gauche

- la partition du plan au dessus à droite comprend tous les nœuds successeurs dans la lecture séquentielle du document.

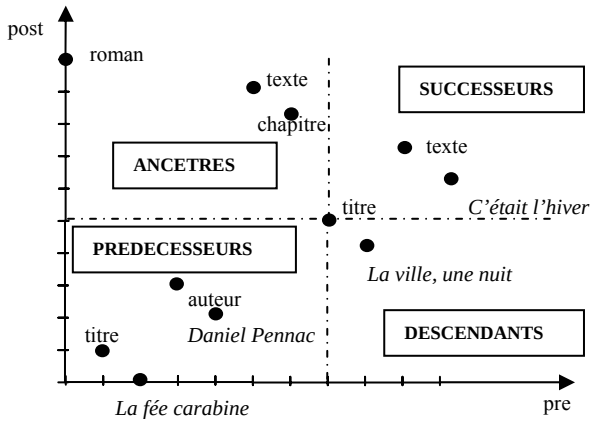

Figure 5: *Représentation du document fée.xml dans un espace à deux dimensions basé sur les coordonnées de pré-ordre et post-ordre.*

Ainsi, les requêtes XPath (W3C, 1999) du type *: Child, Descendant, Parent, Ancestor, following, preceding, following-sibling, preceding-cibling sont rapidement traitées*. Par exemple :

Un nœud n' est ancêtre de n si pre(n') < pre(n) et post (n')> post (n)

Outre le traitement des expressions XPath, cette représentation des nœuds est particulièrement intéressante pour une navigation dans la structure des documents. Contrairement à d'autres approches basées sur des index de structure, elle permet de répondre à des expressions XPath qui n'ont pas pour origine la racine du document, et ce en élaguant l'arbre représentant le document. Elle permet de plus de reconstruire rapidement le XPath correspondant à un nœud.

3.2. Le modèle physique de représentation des données

Comme les SRI traditionnels, XFIRM propose la construction de structures d'index pré-calculées qui sont utilisées pour l'évaluation des différentes conditions de recherche énoncées dans les requêtes. Ces index sont basés sur la modélisation des documents structurés que nous avons présentée ci-dessus. Ils sont stockés sous forme de tables dans une base de données relationnelle. Afin d'obtenir les différents index, les documents à indexer sont parcourus à l'aide d'un parseur de type SAX (SAX) par exemple.

Trois *tables génériques*, utilisées par les index principaux, sont présentes dans la base de données :

Table générique	Description
Documents (doc_id, document, date, nb_termes)	*doc_id* est l'identifiant unique de chaque document, *document* est le nom de fichier du document, *date* est la date d'insertion dans l'index du document, et *nb_termes* est le nombre total de termes du document
Balises (balise_id, balise),	*balise_id* est l'identifiant unique de chaque nom de balise et *balise* est le nom de la balise
Attributs (att_id, attribut)	*att_id* est l'identifiant unique de chaque nom d'attribut et *attribut* est le nom de l'attribut

Tableau 2: *Tables génériques du modèle physique de XFIRM*

Les index principaux, au nombre de cinq, sont les suivants :

- *L'index des chemins (IC)* permet de reconstituer la structure des documents ;

- *L'index des termes (IT)* donne pour chaque terme de la collection les éléments associés et permettra de calculer diverses mesures de pertinence en fonction du modèle de recherche choisi ;

- *L'index des éléments (IE)* décrit le contenu de chaque nœud feuille, et permettra de faire des évaluations de pertinence sur des nœuds précis ;

- *L'index des attributs (IA)* donne pour chaque attribut ses différentes valeurs ;

- et enfin le *dictionnaire (DICT)* permet de regrouper les balises de la collection ayant la même sémantique. En effet, la qualité des recherches sur des données semi-structurées pourrait être améliorée en utilisant la sémantique du nom des éléments (Theobald, 2002).

On trouvera la description plus précise de ces index dans le tableau 3.

Le choix des nœuds à indexer (c'est à dire de l'unité d'information minimale qui pourra être renvoyée à l'utilisateur) est fait au début du processus d'indexation. Plusieurs scénarios sont possibles :

- tous les nœuds sont indexés ;

- le choix des nœuds à indexer est fait manuellement ou grâce à des statistiques selon la ou les DTD des documents et une liste d'éléments non indexables est crée ;

- seuls les nœuds n'ayant pas pour frères des nœuds feuilles #PCDATA sont indexés. Dans cette dernière situation en effet, les nœuds concernés sont souvent des nœuds indiquant la mise en forme du texte (balises de type italique ou gras). Le texte contenu à l'intérieur de ces balises sera indexé comme faisant partie de l'élément #PCDATA le précédant.

Index	Table	Description
IC	***Chemins (chemin_id, doc_id, pre, post, parent, attribut, balise_id)***	*chemin_id* est l'identifiant unique de chaque chemin, *doc_id* est l'identifiant du document concerné, *pre* et *post* sont les valeurs de prédécesseurs et successeurs, *parent* est la valeur de prédécesseur du parent de l'élément, *attribut* est un booléen indiquant la présence d'attribut pour l'élément concerné, et *balise_id* est l'identifiant de la balise de l'élément concerné. Si le champ *balise_id* est nul pour un certain *chemin_id*, l'élément est alors un élément feuille de type #PCDATA et on trouvera sont contenu dans l'index des éléments.
IT	***TermesElements (terme_id, terme, total_fréquence, nb_doc, nb_elt, fréquences)***	*terme_id* est l'identifiant unique de chaque terme, *terme* est le terme lui même, *total_fréquence* est la fréquence totale du terme dans la collection, *nb_doc* est le nombre total de documents dans lesquels le terme apparaît, *nb_elt* est le nombre total d'éléments (c'est à dire de chemins) dans lesquels le terme apparaît et *fréquences* est un champ de type BLOB (Binary Long Object) contenant pour chaque élément où le terme apparaît (élément représenté par *chemin_id*) le nombre d'occurrences du terme, ainsi que les positions auxquelles il apparaît. Par exemple, la chaîne « 2 1 2/ 21 2 4 8 » indique que le terme *t* est présent 1 fois dans l'élément 2 à la position 2 et 2 fois dans l'éléments 21 aux positions 4 et 8.
IE	***ElementsTermes (chemin_id, nb_termes, nb_total_termes)***	*chemin_id* est l'identifiant de chaque chemin, *nb_termes* est le nombre de termes uniques inclus dans l'élément concerné, *nb_total_termes* est le nombre de termes inclus dans l'élément concerné
IA	***ValeursAttributs (chemin_id, attribut_id, valeur)***	*chemin_id* est le nœud auquel se rattache l'attribut, *attribut_id* est l'identifiant de l'attribut (en référence à la Table Attributs) et valeur est une chaîne de caractère contenant la valeur de l'attribut.
DICT	***Dictionnaire (balise_id, ListeBalise_id)***	*balise_id* est un identifiant de balise et *ListeBalise_id* est une liste d'identifiants de balise ayant une sémantique proche de *balise_id.*

Tableau 3 : *Index du modèle physique de XFIRM*

Notons de plus que les termes contenus dans l'IT sont lemmatisés. La lemmatisation peut être effectuée avec l'algorithme de Porter (Porter, 1980) pour les documents de langue anglaise ou bien en effectuant des troncatures pour les autres langues. Une liste de mots vides est utilisée pour supprimer les termes qui n'apportent pas de sens au contenu des éléments, comme par exemple les pronoms ou les déterminants.

Enfin, l'utilisation du dictionnaire permettra d'étendre les requêtes des utilisateurs et d'établir des liens entre des documents suivants des DTDs différentes. Par exemple, les balises *chapitre* et *acte* des documents *songe.xml* et *fée.xml* peuvent être considérée comme équivalentes.

Les structures de stockage que nous venons de présenter contiennent toutes les informations nécessaires pour appliquer différents modèles de RI, tant sur des requêtes portant seulement sur le contenu des documents que des requêtes plus précises portant aussi sur leur structure. Les différents index étant stockés dans une base de données, toutes les fonctions usuelles des bases de données (comme les jointures, les projections ou le tri) ne sont pas à réimplémenter. De plus, la mise à jour des index dans le cas de suppression ou d'insertion de documents restera relativement simple.

4. Le langage de requête XFIRM

Le langage de requête XFIRM propose à l'utilisateur de formuler son besoin selon divers degrés de précision. S'il recherche simplement de l'information et que le type de l'unité d'information renvoyée lui importe peu pourvu qu'elle réponde à son besoin, il pourra formuler sa requête avec de simples mots-clés. Par contre, s'il a une idée plus précise en termes de structure de ce qu'il désire, il pourra ajouter ces informations dans sa requête. Les types de requêtes sont ainsi classés selon quatre niveaux de précision.

Les requêtes peuvent être composées de simple mots clés (degré de précision P1), éventuellement reliés entre eux par des opérateurs. Par exemple, l'utilisateur peut clarifier ses besoins à l'aide des opérateurs + et – indiquant que la présence des mots est indispensable dans les unités d'information résultats ou au contraire qu'elle n'est pas souhaitée. Les opérateurs booléens ET, OU et NON permettent de plus d'exprimer des conditions booléennes dans la requête. Enfin, si l'utilisateur recherche des expressions, il lui suffira de mettre ses mots clés entre ″ ″ .

Les requêtes

P1.1 : esprits fée *P1.2 : +esprits –″la fée carabine ″*

P1.3 : esprits OU (fées ET lutins)

indiquent par exemple que l'utilisateur recherche des unités d'information traitant respectivement d'*esprits* et de *fée* (dans le cas de P1.1), nécessairement d'*esprits*

mais ne contenant pas l'expression *″ la fée carabine ″* (P1.2) , d'*esprits* ou de *fées* et de *lutins* (P1.3).

Si l'utilisateur désire donner des conditions sur la structure des documents, il peut exprimer son besoin en donnant le nom d'une balise. Il peut de plus ajouter des contraintes sur le contenu des éléments correspondants ou sur leurs attributs. Ces requêtes de précision P2 peuvent être combinées entre elles par des opérateurs booléens. Par exemple, les requêtes :

P2.1 : chapitre() *P2.2 : chapitre(ville nuit)*
P2.3 : titre(″la fée carabine ″) ET chapitre(@numero=1)

signifient que l'utilisateur souhaite obtenir un élément de type *chapitre* (dans le cas de P2.1), un élément *chapitre* parlant de *ville* et de *nuit* (dans P2.2), ou une *unité d'information* contenant à la fois un élément *titre* sur *″la fée carabine ″* et un élément *chapitre* ayant un attribut *numéro* de valeur *1* (P2.3).

Si l'utilisateur désire de plus exprimer une notion de hiérarchie entre les différents éléments (précision P3), il peut combiner des requêtes de type P2. Elles sont alors séparées par le signe *″//″* et leur ordre d'apparition indique la hiérarchie souhaitée. Par exemple, les requêtes :

P3.1 : //roman()// titre(″la fée carabine ″) ET chapitre(ville nuit)
P3.2 : //roman()// texte(bonjour)// chapitre(@numero=1)

signifient l'utilisateur souhaite obtenir respectivement un nœud *roman* ayant pour descendant un élément *titre* sur *″la fée carabine ″* et un élément *chapitre* parlant de *ville* et de *nuit* (P3.1), un nœud *roman* ayant pour descendant un élément *texte* contenant le mot *bonjour* et étant lui-même ancêtre d'un nœud *chapitre* ayant un attribut *numéro* de valeur *1* (P3.2).

Dans les requêtes de type P3, les nœuds retournés à l'utilisateur sont par défaut ceux spécifiés dans la première requête de type P2 (*roman* dans les exemples P3.1 et P3.2). Si l'utilisateur a une idée plus précise de ce qu'il recherche, il pourra spécifier l'unité d'information qu'il désire voir retournée. Dans la suite, nous nommerons cette unité d'information *élément cible*. Cet élément cible est spécifié grâce au signe *″*ec :*″* précédant une requête de type P2. Ainsi la requête :

P4.1 : //roman(@date-publication=1987)// ec : [titre(fée)]// chapitre(ville) ET titre(introduction)

signifie que l'utilisateur souhaite obtenir un nœud *titre* contenant le terme *fée* et ayant pour ancêtre un nœud *roman* dont l'attribut *date-publication* vaut *1987* et pour descendant un nœud *chapitre* parlant de *ville* et un nœud titre contenant le terme *introduction*.

La syntaxe de ces requêtes permet à l'utilisateur de formuler des *expressions de chemin vagues* dans l'expression de ses conditions. Il peut par exemple exprimer la requête *roman//chapitre (*il sait alors qu'un nœud *pièce* a pour descendant un nœud *acte)*, sans indiquer nécessairement le chemin d'accès précis (*roman/texte/chapitre)*.

La syntaxe exacte du langage de requête XFIRM est décrite dans la grammaire suivante :

requête : := <P1> | <P2> | <P3> |<P4>

P1 : := <expressionRéduite> <suiteExpressions>
expressionRéduite : := <termes> <suiteExpressionRéduite> | ″ (″ <termes> ″) ″ <suiteExpressionRéduite>
suiteExpressionRéduite : := <expressionRéduite> | vide
suiteExpressions : := <opérateurBooléen> <P1> | vide
termes : :=<opérateurAdditif> <motsClés>
motsClés : := terme<suiteTermes> | ″″ ″ terme<suiteTermes>″ ″ ″<suiteTermes>
suiteTermes : :=vide | <termes>
opérateurAdditif : := ″ + ″ | ″ - ″ | vide
opérateurBooléen : := ″OU ″ | ″ ET ″ | ″ NON ″| vide

P2 : := <expressionStructure><suiteExpressionStructure>
expressionStructure : :=nomElement ″ (″ <condition> ″) ″
condition : := ″ @ ″ nomAttribut ″ = ″ terme | P1 |vide
suiteExpressionStructure : := <operateurBooléen> <expressionStructure> |vide

P3 : := ″ // ″ <P2><suiteP3>
suiteP3 : :=<P3> | vide

P4 : := <suiteP3><elementCible><suiteP3>
ElementCible : := ″ // ec : [″ <P2> ″] ″

Légende : vide : expression terminale représentant l'ensemble vide
terme : expression terminale représentant un mot clé
nomElément : expression terminale représentant un nom de balise
nomAttribut : expression terminale représentant un nom d'attribut
ec : : expression terminale indiquant la présence d'un élément cible

Figure 6: ***Grammaire BNF du langage de requête XFIRM***

5. Traitement des requêtes XFIRM

Le traitement des requêtes par le modèle XFIRM a pour but de renvoyer les unités d'informations les plus spécifiques et exhaustives à l'utilisateur. Les quatre types de requêtes définis sont étroitement liés. Comme nous l'avons vu dans la section 4, les requêtes les plus précises (de type P4 ou P3) sont construites à partir des requêtes de type P2. Ainsi, une requête $P3_i$ de type P3 et une requête $P4_i$ de type P4 se décomposent de la façon suivante :

$$P3_i = // P2_1 // P2_2 //\ldots//P2_n \quad [1]$$

$$P4_i = // P2_1 // P2_2 //\ldots//ec :[P2_i]//\ldots//P2_n \quad [2]$$

Une requête $P2_i$ de type *P2* peut ensuite être décomposée en ***sous-requêtes élémentaires*** $\boldsymbol{SRE_{i,j}}$ reliées entre elles par des opérateurs booléens et de la forme :

$$SRE_{i,j} = \begin{cases} b\_n\ (q) \\ b\_n\ (n\_a = v) \end{cases} \quad [3]$$

où : - *b_n* est le nom de balise du nœud *n*
- $q = \{t_1, \ldots t_n\}$ est un ensemble de mots-clés, c'est à dire une requête de type *P1*
- *n_a* est le nom d'attribut de l'attribut *a* avec *a estAttribut de n*
- *v* est la valeur désirée de *a*

Illustrons cette décomposition avec la requête P4.1 : *//roman(@date-publication=1987) // ec : [titre(fée)]// chapitre(ville) ET titre(introduction).* Cette requête se décompose en requêtes de type P2 de la façon suivante : P4.1=// $P2_1$= *roman(@date-publication=1987//ec :[*$P2_2$ *=titre(fée)] //* $P2_3$ *= chapitre(ville) ET titre(introduction).*

Nous avons ensuite par exemple *:* $P2_3$ = ($SRE_{3,1}$=*chapitre(ville))* *ET* ($SRE_{3,2}$=*titre(introduction))*, où *ville* et *introduction* sont des requêtes de type *P1*.

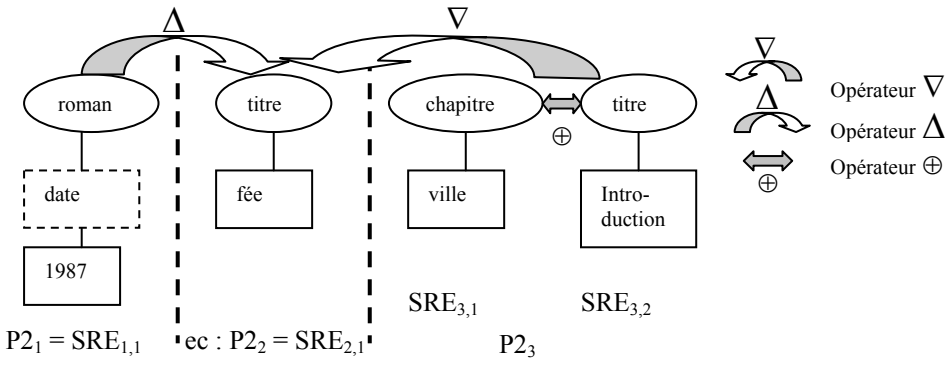

Figure 7 : *Décomposition en sous-requêtes élémentaires de la requête P4.1*

Dans ce qui suit, nous allons donc commencer par décrire le traitement de requête composées de mots-clés (de type P1), pour ensuite décrire le traitement de la structure, à savoir les requêtes de type P2 et enfin les requêtes de type P3 et P4.

5.1. Appariement requête composée de mots clés - Unité d'information

Comme nous l'avons vu ci-dessus, quelque soit la précision avec laquelle une requête Q est exprimée, on peut en extraire une ou plusieurs sous-requêtes de type liste de termes ou expressions et de la forme $q=\{t_1,t_2, \ldots t_n\}$. En effet, soit la requête

est simplement composée de mots-clés (requêtes *P1.1, P1.2, P1.3* de la section 4), soit elle contient des conditions sur la structure, qui peuvent elles-mêmes contenir des conditions de contenu (requêtes *P2.2, P2.3, P3.1, P3.2, P4.1*). Une première étape dans l'évaluation d'une requête Q sera donc de calculer les pertinences des sous-requêtes *q* par rapport à des nœuds feuilles *nf*. La structure de représentation des informations du système XFIRM autorise l'implémentation de nombreux modèles de RI, qui seront utilisés pour assigner un poids de pertinence nœuds feuilles. Ces pertinences sont évaluées à partir d'une fonction de similarité notée $RSV_m(q,nf)$ (Retrieval Status Value), où *m* est le modèle de recherche d'information concerné.

On a alors :

$$RSV_m(q,nf) = f_{j=1,..,n}\ (q_j, w_j^{(ui)}) \qquad [4]$$

où - q_j est le poids du terme t_j dans la requête *q*
- $w_j^{(ui)}$ est le poids du terme t_j dans le nœud feuille *nf*
- la fonction *f* permet d'agréger les poids des termes t_j pour le nœud feuille *nf.*

$w_j^{(ui)}$ est calculé de la façon suivante :

$$w_j^{(ui)} = g(tf_j, nb\_doc_j, nb\_elt_j, nb\_termes, nb\_termes\_tot, nb\_termes\_doc) \qquad [5]$$

où - tf_j est la fréquence du terme t_j dans le nœud feuille *nf*,
- nb_doc_j est le nombre de documents contenant t_j,
- nb_elt_j est le nombre de noeuds contenant t_j ,
- et *nb_termes, nb_termes_tot, nb_termes_doc* sont respectivement le nombre de termes uniques dans *nf,* le nombre de total termes dans *nf,* et le nombre de termes contenus dans le document auquel appartient *nf.*

Pour obtenir les unités d'informations les plus pertinentes aux requêtes Q, les pertinences des nœuds feuilles *nf* à ces sous-requêtes *q* sont ensuite :

- combinées grâce à des opérateurs ensemblistes dans le cas de requêtes de type P1 composées de simples mots-clés

- combinées grâce à plusieurs opérateurs que nous présentons dans le paragraphe suivant dans le cas de requêtes Q exprimant des conditions sur la structure des éléments.

Dans le cas de requêtes à bases de simples mots clés, le modèle utilisé pourra par exemple utiliser des formules d'agrégations des poids (Kazai et *al*., 2001) (Fuhr et *al.*, 2001), ou des statistiques sur la taille des éléments et la structure des documents (Hatano et *al.,* 2002) pour renvoyer un ensemble résultat composé de paires *(unité d'information, pertinence)*. Le but est d'obtenir des unités d'information n'étant pas nécessairement des nœuds feuilles et ayant une granularité appropriée (ni trop petite ni trop grande).

5.2. Traitement de la structure dans les requêtes

5.2.1. Traitement des sous-requêtes élémentaires $SRE_{i,j}$

L'ensemble de paires (nœud, pertinence) $R_{i,j}$ résultat d'une $SRE_{i,j,}$ (définie dans [3]) est calculé de la façon suivante :

(1) Si $SRE_{i,j} = b\_n\ (q)$, (c'est le cas par exemple de $SRE_{3,1}$ dans notre exemple)

$$R_{i,j} = \{\ (n, p_n) / n \in \{construct(b\_n)\} \text{ et } p_n = F_k\ (RSV_m(q, nf_k))\ \} \quad [6]$$

où : - p_n est le poids de pertinence du nœud n

- la fonction $construct(b\_n)$ permet de créer l'ensemble de tous les nœuds ayant pour nom de balise b_n,

- la fonction F_k $(RSV_m(q, nf_k))$ permet d'agréger les poids de pertinence des nœuds feuilles nf_k descendants de n. Les poids sont calculés d'après [4] .

(2) Si $SRE_{i,j} = b\_n\ (n\_a = v)$, (c'est le cas par exemple de $SRE_{1,1}$)

$$R_{i,j} = \{\ (n, 1) / \forall\ n \in \{construct(b\_n)\},\ a \in \{construct(n\_a)\}\ estAttribut\ de\ n\ et\ valeur(a) = v)\ \} \quad [7]$$

5.2.2. Traitement d'une requête de type P2

Une fois que les requêtes $SRE_{i,j}$ ont été traitées, les requêtes $P2_i$ de type P2 sont reconstituées grâce aux opérateurs $\oplus_{ET}$ et $\oplus_{OU}$ définis ci-dessous.

Définition 1 : Soient deux ensembles de paires (nœud, pertinence) $N = \{ (n, p_n) \}$ et $M = \{ \{m, p_m) \}$

$N \oplus_{ET} M = \{ (l, p_l) /$ *l est le plus proche ancêtre commun de m et n ou l=m (respectivement n) si m (resp .n) est ancêtre de n (resp. m) ,* $\forall m$, *n appartenant au même document et* $p_l = agreg_{ET}(p_n , p_m) \}$ [8]

$N \oplus_{OU} M = \{ (l, p_l) / l=n \in N$ *ou* $l=m \in M$ *et* $p_l = agreg_{OU}(p_n , p_m) \}$ [9]

Où $agreg_{ET}(p_n , p_m) \text{ -> } p_l$ et $agreg_{OU}(p_n , p_m) \text{ -> } p_l$ définissent la façon dont les pertinences p_n *et* p_m des nœuds n et m sont agrégées pour former une nouvelle pertinence p.

Soit l'ensemble résultat R_i d'une requête $P2_i$. Alors :

Si $P2_i = SRE_{i,j}$, alors $R_i = R_{i,j}$ [10]

Si $P2_i = SRE_{i,j}$ ET $SRE_{i,k}$,, alors $R_i = R_{i,j} \oplus_{ET} R_{i,k}$ [11]

Si $P2_i = SRE_{i,j}$ OU $SRE_{i,k}$,, alors $R_i = R_{i,j} \oplus_{OU} R_{i,k}$ [12]

Le résultat d'une requête $P2_i$ est donc un ensemble R_i composé de paires formées de nœuds l et du poids de pertinence p_l qui leur est associé.

5.2.3. Traitement d'une requête de type P3

Le résultat d'une telle requête est obtenu grâce à l'opérateur non-commutatif ∇ défini ci-dessous :

Définition 2 : Soient deux ensembles de paires (nœud, pertinence) $R_i = \{ (n, p_n) \}$ et $R_{i+1} = \{ \{m, p_m) \}$

$R_i \nabla R_{i+1} = \{ (n, p) / n \in R_i$ *estAncêtre de* $m \in R_{i+1}$ *et* $p=prop\_agg(distance(m, n), p_n , p_m) \}$ [13]

Où $prop\_agg(distance(n,m), p_n, p_m) \rightarrow p$ permet d'agréger les pertinences p_m du nœud *m* et p_n du nœud *n* en fonction de la distance qui sépare les deux nœuds, pour obtenir la nouvelle pertinence *p* du nœud *n.*

L'ensemble résultat R d'une requête de type P3 est alors défini de la façon suivante :

$$R = R_1 \nabla (R_2 \nabla (R_3 \nabla \ldots)) \quad [14]$$

5.2.4. Traitement d'une requête de type P4

Le résultat d'une telle requête est obtenu grâce aux opérateurs non-commutatifs ∇ (défini ci-dessus) et Δ (défini ci-dessous):

Définition 3 : Soient deux ensembles de paires (nœud, pertinence) $R_i = \{ (n, p_n) \}$ et $R_{i+1} = \{ \{m, p_m) \}$

$R_i \Delta R_{i+1} = \{ (m, p) / m \in R_{i+1}$ *estDescendant de* $n \in R_i$ *et* $p=prop\_agg(distance(n,m), p_m , p_n) \}$ [15]

L'ensemble résultat R d'une requête de type P4 est alors défini en trois étapes :

(1) $SR_1 = R_i \nabla (R_{i+1} \nabla (R_{i+2} \nabla \ldots))$

(2) $SR_2 = (((R_1 \Delta R_2) \Delta R_3) \Delta \ldots) \Delta R_i$

(3) $R = SR_1 \cup SR_2$ [16]

A tout moment dans le traitement de la structure des documents, l'index *DICT* peut être utilisé pour étendre la requête et donc augmenter le nombre de résultats.

5.2.5. Remarques sur les fonctions d'agrégation et de propagation des poids

La fonction $prop\_agg(distance(n,m), p_n, p_m) \rightarrow p$ utilise la distance qui sépare les nœuds dans l'arbre du document pour propager les poids des nœuds et calculer de nouvelles pertinences (Carmel et *al*., 2003).

Les fonctions $agreg_{ET}(p_n , p_m) \rightarrow p$ et $agreg_{OU}(p_n , p_m) \rightarrow p$ calculent simplement un nouveau poids de pertinence à partir de deux poids déjà calculés. Il peut par exemple s'agir de simples fonctions produit ou somme.
Ces fonctions permettront d'ajuster la façon dont on répond à la requête :

(i) soit de manière ***stricte***, et alors toutes les conditions sur la structure doivent être respectées,

(ii) soit de manière ***vague*** et dans ce cas certaines conditions pourront ne pas être respectées.

6. Conclusion et perspectives

Le système XFIRM permet de stocker et d'indexer des documents XML et, grâce à la flexibilité apportée par le modèle d'index, d'interroger selon différents modèles de recherche d'information les collections de documents XML. Le modèle de stockage est basé sur une approche base de données, ce qui permet une implémentation relativement simple et l'utilisation des fonctions déjà existantes (tri, jointure, etc). Il repose sur cinq index principaux (Index des Chemins, Index des Elements, index des Termes, index des Attributs et Dictionnaire), qui permettent de naviguer efficacement au sein de la structure des documents et de stocker toutes les informations nécessaires aux différents modèles de RI (entre autres la fréquence et les positions des termes). De plus la mise à jour des index est possible pour la suppression ou l'insertion d'un nouveau document. Le modèle de représentation des données proposé permet en outre de traiter des documents XML ne suivant pas la même DTD au sein d'une même collection.

Le langage de requête associé possède une syntaxe simple et permet à l'utilisateur de formuler des requêtes à base de simples mots-clés ou alors d'exprimer des contraintes sur la structure des documents et sur le contenu de cette structure, et ce selon le degré de précision associé à son besoin. La formulation des requêtes sur la structure des document peut être vague, c'est à dire que l'utilisateur n'a pas besoin de connaître précisément les DTD des documents de la collection. De plus, l'utilisation d'un dictionnaire des balises de la collection permet d'étendre les recherches à d'autres éléments que ceux spécifiés dans la requête.

Les informations stockées dans les index serviront à tester de nombreux modèles de recherche d'information, dans le but de renvoyer les unités d'information les plus exhaustives et spécifiques aux requêtes.

Le prototype de XFIRM est actuellement en cours d'implémentation, et la campagne d'évaluation INEX (INitiative for the Evaluation of XML Retrieval) (INEX 2003) nous permettra de valider différents modèles de recherche.

Références :

Abiteboul, S. ,Quass, D., Mc Hugh, J., Widom, J. , Wiener, J-L. , "The Lorel query language for semi-structured data". *International Journal on Digital Libraries*, 1(1), 68-88, 1997.

Bradley, N, *The XML Companion.* Addison Wesley Professional Publisher, 864 p, 2001.

Buneman, P., Davidson, S. ,Hillebrad, G., Suciu, D., "A query language and optimisation techniques for unstructured data". *ACM-SIGMOD record*, pp. 505-516, Montréal, 1996.

Carmel, D., Maarek, Y.S., Mandelbrod, M., Mass, Y. , Soffer, A. "Searching XML documents via XML Fragments". *In Proc. Of SIGIR'03*, 2003, Canada, p. 151-158.

Ceri, S. , Comai, S. , Damiani, E. , Fraternalli, P. , Paraboschi, S. et Tanca, L. : "XML-GL : A graphical language for querying and restructuring WWW Data". . *In Proc. Of the 8th Int. WWW Conference, WWW8*, Toronto, Canada, May 1999.

Chamberlin, D. , Robie, J., Florescu, D. , " Quilt : An XML query language for heterogeneous data sources". *In Proc. 3rd International Workshop on World Wide Web and databases.* LNCS (pp.1-25). Dallas, 2000.

Chinenyanga, T. T., Kushmerick, N. , "Expressive Retrieval from XML Documents". *In Proc. Of ACM SIGIR 2001*, pp. 163-171, New-Orlean, USA, 2001.

Chiramella, Y. , Mulhem, P. , Fourel, F. A model for multimedia search information retrieval. Technical report, Basic Research Action FERMI 8134, 1996.

Deutsch, A. , Fernandez, M. F., and Suciu, D.," Storing Semistructured Data with STORED". *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data*, June 1-3, 1999, Philadelphia, Pennsylvania, USA, pages 431-442.

Florescu, D. and Kossmann, D., "Storing and Querying XML Data using an RDMBS". *IEEE Data Engineering Bulletin*, 22(3):27-34, 1999.

Fuhr, N., Grossjohann, K. "XIRQL: A query Language for Information Retrieval in XML Documents". *In Proc. of the 24th annual ACM SIGIR conference on research and development in Information Retrieval*, New Orléans, USA, p. 172-180. ACM Press, 2001.

Fuller, M. , Mackie, E. , Sacks-Davis, R. , Wilkinson, R. "Structural answers for a large structured document collection". *In Proc. ACM SIGIR*, pp. 204-213. Pittsburgh, 1993.

Grabs, T., Sheck, H.J. : "ETH Zürich at INEX : Flexible Information Retrieval from XML with PowerDB-XML". *In INEX 2002 Workshop Proceedings*, p. 35-40, Germany, 2002.

Grust, T, "Accelerating XPath Location Steps". In M. J. Franklin, B. Moon, and A. Ailamaki, editors, *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, Madison, Wisconsin, USA, pages 109-120. ACM, 2002.

Hatano, K., Kinutani, H., Watanabe, M,:"An Appropriate Unit of Retrieval Results for XML Document Retrieval". *In INEX 2002 Workshop Proceedings*, p. 66-71, Germany, 2002.

Hayashi, Y. , Tomita , J., Kikoi, G , "Searching text-rich XML documents with relevance ranking". *In Proc ACM SIGIR 2000 Workshop on XML and IR* (pp. 27-35). Athens 2000.

Hearst, M.A., "TextTiling : A Quantitative Approach to Discourse Segmentation." *Computational Linguistics*, 23(1) :33-64, Mar. 1997.

INEX - Initiative for the Evaluation of XML Retrieval. http://inex.is.informatik.uni-duisburg.de:2003/

Kazai, G. , Lalmas, M., Roelleke, T. :" A model for the representation and focused retrieval of structured documents based on fuzzy aggregation". *SPIRE'2001*, p. 123-135, Chile, 2001.

Lalmas, M. , " Dempster-Shafer's theory of evidence applied to structured documents : Modeling uncertainty". *In Proc. ACM-SIGIR*, pp. 110-118. Philadelphia, 1997.

Levy, A., Fernandez, M. ,. Suciu, D, Florescu, D. , Deutsch, A. XML-*QL* : A query language for XML. World Wide Web Consortium technical report, Number NOTE- xml-ql-19980819, 1998.

Moffat, A., Sacks-Davis, R. , Wilkinson, R. , and Zobel, J.. "Retrieval of Partial Documents." *In TREC-2*, 1993.

Porter, M. F. " An algorithm for suffix stripping". *Program 14*, pages 130-137, 1980.

Robertson, S.E. , "The probability ranking principle in IR". *Journal of Documentation* 33(4), pages 294-304, 1977.

Robie , J.(Texcel), Lapp, J.(webMethods Inc.), Schach, D.(Microsoft), "XML Query Language (XQL)". *Proc. of W3C QL'98* (Query Languages 98). Massachusetts, 1998.

Salton, G. , *The SMART retrieval system : Experiments in automatic document processing*. PRENTICE HALL, 1970.

Salton, G., McGill, M.J., *Introduction to modern information retrieval*. McGraw-Hill Int. Book Co, 1984

SAX (Simple API for XML). http://sax.sourceforge.net

Schlieder, T., Meuss, H. , "Querying and ranking XML documents". *Journal of the American Society for Information Science and Technology*, 53(6) : 489-503, 2002.

Theobald, A , Weikum, G. "The Index-Based XXL Search Engine for Querying XML Data with Relevance Ranking". *EDBT 2002, 8th International Conference on Extending Database Technology*, Prague, Czech Republic, pages 477-495. Springer, 2002.

World Wide Web Consortium (W3C). James CLARK, Steve DEROSE : XML Path Language (Xpath) ,. W3C Recommendation, Novembre 1999. http://www.w3.org/TR/xpath

World Wide Web Consortium (W3C). Extensible Markup Language (XML) 1.0. http://www.w3.org/TR/REC-xml , Oct. 2000.

World Wide Web Consortium (W3C). Xquery and Xpath Full-Text Use Cases. W3C Working draft, février 2003. Disponible sur: http://www.w3.org/TR/2003/WD-xmlquery-full-text-use-cases-20030214/.

World Wide Web Consortium (W3C). M Fernandez et al. : XQuery 1.0 and XPath 2.0 Data Model. W3C Working Draft, Mai 2003. Disponible sur : http://www.w3c.org/TR/xpath-datamodel/

World Wide Web Consortium (W3C). XQuery 1.0 : an XML query language. W3C Working Draft, Novembre 2003. Disponible sur: http://www.w3.org/TR/xquery/

Wolff, J.E., Flörke, H., Cremers, A.B., "Searching and browsing collections of structural information". *In Proc of IEEE advances in digital libraries*, p. 141-150. Washington, 2000.

Zipf, G. , Human Behaviour and the Principle of Least Effort, Addison-Wesley, 1949.