
Le langage de requêtes XFIRM pour la recherche d'information dans les documents XML

De la recherche par simples mots-clés à l'utilisation de la structure des documents

Karen Sauvagnat, Mohand Boughanem

*IRIT – Equipe SIG / RI
118 route de Narbonne
31 062 Toulouse Cedex 4
{sauvagna,bougha}@irit.fr*

RÉSUMÉ. Un des principaux avantages du format XML est sa capacité à combiner des données structurées et des données non structurées, c'est à dire du texte. De nombreux langages de requêtes, basés sur des approches orientées bases de données ou recherche d'information, ont été proposés pour l'interrogation de corpus XML. Cependant, ils accentuent respectivement l'interrogation sur la structure des documents ou l'interrogation sur le texte, mais ne parviennent pas à combiner avantageusement les deux. De plus, dans la plupart des cas, les utilisateurs doivent avoir une connaissance parfaite de la structure des documents qu'ils interrogent, et la syntaxe du langage est relativement complexe. Dans cet article, nous proposons le langage de requête XFIRM, permettant à l'utilisateur de formuler des requêtes simples à base de mots-clés, ou bien des requêtes plus complexes utilisant la structure des documents.

ABSTRACT. One of the key advantages of XML is its capacity to combine structured and unstructured (text) data. Many languages, based on database-oriented approaches or on information retrieval-oriented approaches, have been proposed in the literature for querying XML corpus. However, they respectively accentuate document structure querying or text querying, but they do not manage to combine it in an attractive way. Moreover, in most cases, users have to perfectly know the document structure, and the syntax of the query language is relatively complex. In this paper, we propose the XFIRM query language, allowing the user to express simple queries based on keywords or more complex queries using document structure

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

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

1. Introduction

XML (eXtensible Markup Language) (W3C, 2000), (Bradley, 2001) est un format de données semi-structuré, qui sépare le contenu des documents des instructions de présentations. Grâce à sa flexibilité, il tend à devenir un format universel pour l'échange des données sur le World Wide Web. 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. Ils devront permettre aux utilisateurs d'interroger simplement les corpus de documents XML et d'obtenir en réponse des parties de documents répondant de façon complète à leur besoin.

Dans cet article, nous présentons le langage de requête associé au système XFIRM (XML Flexible Information Retrieval Model), un système flexible pour la recherche d'information dans des documents XML (Sauvagnat, 2004). Ce langage de requête permet d'exprimer 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 les différents langages de requêtes proposés dans la littérature pour la recherche d'information dans des documents structurés. A partir de cette discussion, nous présentons les spécifications de notre langage (section 4 et 5) et la façon dont les requêtes sont traitées par le système XFIRM (section 6).

2. Etat de l'art et motivations

Les documents XML, par leur structure même, permettent aux Systèmes de Recherche d'Information (SRI) de traiter l'information avec une autre granularité que le document tout entier. En effet, l'utilisateur recherche parfois une information précise, et il ne désire pas que cette information soit "noyée" au milieu d'autres sujets. C'est sur cette constatation que repose "le principe de recherche dans les documents structurés" (Chiaromella *et al.*, 1996) : *un système devrait toujours retrouver la partie la plus spécifique¹ d'un document répondant à une requête*. Cette définition suppose que le système recherche d'abord des documents entiers répondant de manière exhaustive² à une requête, puis extrait de ces documents les unités d'information les plus spécifiques. Une telle procédure étant coûteuse en temps, 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* (Lalmas, 1997).

L'accès aux documents XML a été appréhendé selon deux angles principaux (Fuhr *et al.*, 2001): (i) *l'approche orientée données* utilise des techniques développées par la communauté des bases de données, et une kyrielle de langages de requêtes, proches de SQL, ont été proposés; (ii) *l'approche orientée documents*

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

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

est prise en charge par la communauté de la recherche d'information et considère que les balises servent uniquement à décrire la structure logique des documents.

2.1. 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. 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), qui est un langage de requête graphique s'utilisant sur des graphes XML, et Elixir (Chinenyanga et al., 2001). Le langage QUILT (Chamberlin et al., 2000), a été développé dans le but d'être un langage flexible, combinant des caractéristiques pour interroger des documents et des bases de données. Des fonctions navigationnelles sont disponibles pour parcourir les documents, et il est possible de créer des variables et des fonctions peuvent être définies pour agir sur les objets de données. En ce qui concerne l'interrogation même, elle est basée sur la construction FLRW (for-let-where-return), et elle permet de faire des jointures et d'utiliser des opérateurs d'agrégation. QUILT permet enfin de restructurer l'information résultat d'une requête en document XML.

XQuery (W3C, Nov. 2003) est le langage proposé par le W3C. Avec XQuery, une requête est représentée par une expression, comme avec QUILT. En fait, XQuery tire très fortement ses constructions et caractéristiques de QUILT, lui-même dérivé de XPath, XQL, XML-QL, Lorel et Yatl (Cluet et al., 98). Xquery peut être perçu comme un sur-ensemble de SQL. Les fonctionnalités de SQL sur les tables (collections de tuples) sont étendues pour supporter des opérations similaires sur les forêts (collections d'arbres). Ces extensions ont conduit à intégrer les fonctions suivantes : projection d'arbre sur des sous-arbres, sélection d'arbres et de sous arbres en utilisant des prédicats sur les valeurs des feuilles, utilisation de variables dans les requêtes pour mémoriser un arbre ou pour itérer sur des collections d'arbre, combinaison des arbres extraits de collection en utilisant des jointures d'arbres, imbrication de requêtes, calculs d'agrégats et utilisation possible de fonctions utilisateurs. De plus, XML étant conçu pour gérer des documents, Xquery propose un prédicat *contains* pour la recherche par mots-clés. On trouvera ci-dessous un exemple de requête Xquery :

```
FOR $a IN distinct(document("volume.xml")//auteur)
LET $l := document("volume.xml")//article[auteur = $a]
WHERE count ($l) > 2
RETURN $a
```

Figure 1 : exemple de requête XQuery s'effectuant sur un volume d'articles scientifiques et permettant de sortir tous les auteurs qui ont écrit plus de 2 articles.

2.2. Les approches orientées Recherche d'Information

Les documents XML peuvent aussi être considérés comme une collection de documents texte possédant des balises et des relations entre ces balises. Les techniques de la RI traditionnelle doivent donc être étendues pour prendre en compte la structure et la sémantique de ces documents. De nombreux modèles ont ainsi été adaptés.

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). 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.

Dans le *modèle probabiliste* proposé dans (Fuhr et al., 2001), le langage de requête XIRQL 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. 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, mais la syntaxe des requêtes reste encore relativement complexe.

2.3. Discussion

Si les approches orientées SGBD permettent de traiter avec efficacité la structure des documents XML, elles sont cependant limitées pour le traitement de la partie textuelle des documents. Les mots clés sont en effet 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, comme le proposent les approches orientées RI. On notera cependant que le W3C a récemment proposé un Working Draft (W3C, Fev. 2003), qui a pour but d'étendre les caractéristiques de recherche de XQuery à la recherche plein-texte. Le langage TexQuery (Amer-Yahia et al., 2004) en est une application.

Cependant, dans les deux approches, une connaissance parfaite de la structure des documents est souvent nécessaire à l'utilisateur pour pouvoir formuler des requêtes. Il doit de plus spécifier l'élément qu'il désire voir retourné par le SRI, alors qu'il n'a pas forcément d'idée précise de ce qu'il recherche exactement. La tâche de recherche de l'utilisateur est aussi complexifiée par la syntaxe des différents langages (souvent dérivée de SQL), qui sont difficilement accessibles pour les novices.

De plus, dans les modèles proposés, les documents d'une même collection doivent tous respecter la même DTD. On notera cependant que le langage XISS (Li et al., 2001) propose un modèle pour répondre à des requêtes demandant de trouver des éléments ou des attributs communs dans des documents XML ne suivant pas la même DTD. Enfin, les modèles proposés traitent rarement les unités d'information ne répondant pas de façon exacte aux requêtes sur leur structure. Pour répondre à ce problème, le langage de requête du moteur XXL (Theobald et al., 2002) offre des fonctionnalités sur la recherche orientée-pertinence de chemins, c'est à dire que la recherche est effectuée avec des conditions de chemins *vagues*. On peut cependant déplorer que la syntaxe du langage repose elle-aussi sur SQL.

2.4. Motivations

Le langage de requête XFIRM se propose de répondre aux problèmes soulevés ci-dessus. Ces caractéristiques sont les suivantes :

- **Recherche orientée-pertinence** : le langage repose sur des modèles de RI qui permettront d'assigner des poids de pertinences aux unités d'information retournées aux utilisateurs, et donc de renvoyer des listes triées de résultats ;
- **Syntaxe simple**, ne reposant pas sur SQL ;
- Formulation de requêtes à base de **simples mots-clés**, sans précision aucune sur la structure : ce type de requête pourra par exemple être utilisé lorsque l'utilisateur n'a pas la moindre idée de l'unité d'information qu'il désire voir retournée;
- Possibilité de formuler des **contraintes sur la structure des documents, sans nécessairement donner le type de l'unité d'information à retourner** ;
- Possibilité de formuler des requêtes plus complexes, en introduisant la notion de **hiérarchie** entre les différentes contraintes de structure, mais sans pour autant devoir donner des chemins absolus : le langage permet l'expression **de chemins vagues**.
- Possibilité d'étendre les requêtes grâce à un **dictionnaire des noms de balises** des différents nœuds rencontrés dans le corpus. Ceci sert particulièrement dans le cas de corpus composés de documents suivant des DTDs différentes ou dans le cas de requêtes pour lesquelles l'utilisateur ne connaît pas exactement le nom des éléments qu'il recherche (Theobald et al., 2002).

3. Terminologie et scénario d'exemple

Le langage de requête que nous proposons ici repose sur le modèle de représentation des données défini dans (Sauvagnat, 2004). Ce modèle 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 : $ds_i = (arbre_i) = (\{n_j\}, \{nf_j\}, \{a_j\})$

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, la représentation des nœuds et des attributs est basée sur l'approche XPath Accelerator (Grust, 2002). A chaque nœud est ainsi attribué une valeur de *post-ordre* et de *pré-ordre*, respectivement assignées grâce à un parcours post-fixé et pré-fixé de l'arbre du document. Les données sont stockées dans cinq index principaux (*l'Index des Chemins*, *l'Index des Termes*, *l'Index des Eléments*, *l'Index des Attributs* et le *Dictionnaires*) sous forme de base de données relationnelle.

Dans la suite de l'article, nous utiliserons le document XML suivant pour illustrer notre langage de requête.

```
<article date-publi= " 2000 ">
<titre> Moteurs de recherche : où comment retrouver une aiguille dans une botte de
foin...</titre>
<auteur num= " 1 " > J. Dupont </auteur>
<corps>
  <section num = " 1 " >
    <sous-titre> Introduction </sous-titre>
    <para> La croissance d'internet...</para>
  </section>
  <section num = " 2 " >
    <sous- titre> Moteurs de recherche </sous-titre>
    <para> Yahoo ! est un annuaire...</para>
    <para> Google est un moteur de recherche plein-texte </para>
  </section>
</corps>
</article>
```

Figure 2 : Exemple de document XML : *article.xml*

La représentation en arbre DOM (Document Object Model) du document est alors:

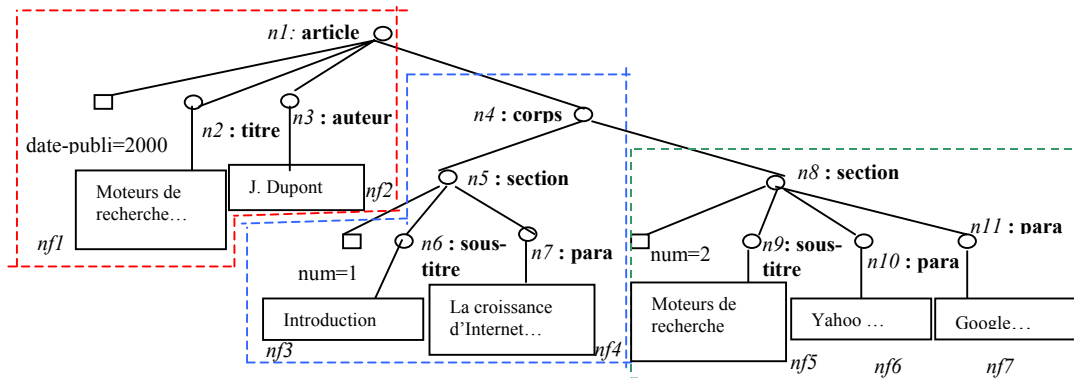


Figure 3 : Représentation en arbre DOM du document *article.xml*

Dans la représentation en arbre du document, chaque *nœud feuille* est de type #PCDATA et les autres *nœuds* (ou *éléments*) sont le point de départ de *sous-arbres*, comme le montre l'exemple ci-dessus. Les nœuds départs de sous-arbres sont notés de *n1* à *n11*, et les nœuds de feuilles de *nf1* à *nf7*, pour plus de facilité dans la suite des exemples.

4. Le langage de requête XFIRM par l'exemple

Le langage de requête XFIRM propose à l'utilisateur de formuler son besoin selon quatre 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 (degré de précision *P1*). Ces mots-clés pourront éventuellement être reliés par des opérateurs (opérateurs booléens ET, OU, NON et opérateurs d'importance "+ "signifiant que le terme est impératif et "-" signifiant que le terme n'est au contraire pas souhaité). La recherche sur des expressions est aussi possible, en encadrant les expressions de " ". Ce type de requête constitue une forme de recherche habituelle dans les moteurs de recherche "traditionnels". On trouvera ci-dessous quelques exemples de requêtes de type *P1*.

P1.1 : `internet google` *P1.2* : `+internet -"moteur de recherche "`
P1.3 : `internet OU (toile ET réseau)`

Si l'utilisateur désire donner des conditions sur la structure des documents, il peut exprimer son besoin en donnant le nom d'un élément, et éventuellement préciser son besoin sur cet élément en ajoutant des conditions sur son contenu ou la valeur de ses attributs. Ces requêtes de précision *P2* peuvent être combinées entre elles par des opérateurs booléens. Par exemple, les requêtes :

P2.1 : `section[]` *P2.2* : `section[internet recherche]`
P2.3 : `titre["moteurs de recherche "] ET section[@num=1]`

signifient que l'utilisateur souhaite obtenir un élément de type *section* (dans le cas de *P2.1*), un élément *section* parlant de *internet* et de *recherche* (dans *P2.2*), ou une *unité d'information* contenant à la fois un élément *titre* sur "*moteurs de recherche*"

et un élément *section* ayant un attribut *num* de valeur 1 (P2.3).

L'élément retourné à l'utilisateur est donc l'élément spécifié dans la requête si la requête est composée d'une seule opérande (P2.1 ou P2.2) ou alors une unité d'information répondant à toutes les conditions s'il s'agit d'une requête contenant des opérateurs booléens (P2.3).

Les requêtes de précision P3 permettent d'ajouter la notion de *hiérarchie* entre les différentes conditions de structures (requêtes de type P2), qui sont alors séparées par le signe *"/*". Par exemple, les requêtes :

P3.1 : //article[] // titre["moteurs de recherche"] ET section[internet google]

P3.2 : //article[] // corps[internet]// section[@num=1]

signifient que l'utilisateur souhaite obtenir respectivement un nœud *article* ayant pour descendant un élément *titre* contenant les termes *"moteurs de recherche"* et un élément *section* parlant de *internet* et de *google* (P3.1), un nœud *article* ayant pour descendant un élément *corps* contenant le mot *internet* et étant lui-même ancêtre d'un nœud *section* ayant un attribut *num* 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 (*article* 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 : //article[@date-publi=2000]// ec : corps[internet] // para[google] ET sous-titre["moteurs de recherche"]

signifie que l'utilisateur souhaite obtenir un nœud *corps* contenant le terme *internet* et ayant pour ancêtre un nœud *article* dont l'attribut *date-publi* vaut 2000 et pour descendant un nœud *para* parlant de *google* et un nœud *sous-titre* contenant l'expression *"moteurs de recherche"* .

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 *article//section* (il sait alors qu'un nœud *article* a pour descendant un nœud *section*), sans indiquer nécessairement le chemin d'accès précis (*article/corps/section*).

Un *dictionnaire* des balises est utilisé par défaut dans le traitement des requêtes. Il est utile dans le cas où l'utilisateur fait des recherches dans un corpus contenant des documents suivant des DTD différentes ou des documents ayant des balises pouvant être considérées comme équivalentes. Par exemple, dans la requête P4.1, la balise *titre* pourra être remplacée par la balise *sous-titre*, car elles sont considérées comme équivalentes dans le dictionnaire.

5. Grammaire du langage de requête

La syntaxe du langage 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 ::= <opérateurBooléen> <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
           nomElement : 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 4: Grammaire BNF du langage de requête XFIRM

6. Traitement des requêtes XFIRM

6.1. Décomposition en sous-requêtes

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 :

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

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

Illustrons cette décomposition avec la requête P4.1 : *//article[@date-publi=2000]// ec : corps[internet]// section[google] ET titre["moteurs de recherche"]*. Cette requête se décompose en requêtes de type P2 de la façon suivante : *P4.1= P2₁= article[@date-publi=2000] //ec :P2₂ = corps[internet] // P2₃ = para[google] ET titre["moteurs de recherche"]*.

Les requêtes de type P3 et P4 peuvent être assimilées à des arbres (puisqu'elles contiennent la notion de hiérarchie). On parlera alors d'*arbre de la requête*.

Une requête *P2_i* de type *P2* peut ensuite être décomposée en *sous-requêtes élémentaires 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 = {t₁, ...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*

Nous avons alors par exemple : *P2₃ = (SRE_{3,1}=para[google]) ET (SRE_{3,2}= titre["moteurs de recherche"])*, où *google* et *"moteurs de recherche"* sont des requêtes de type *P1*.

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*.

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

Comme nous l'avons vu ci-dessus, quelle que 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={t₁,t₂, ...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*, *P4.2*). 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 aux 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.

$$\text{On a alors : } RSV_m(q, nf) = f_{j=1, \dots, n} (q_j, w_j^{(ui)}) \quad [4]$$

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 :

$$w_j^{(ui)} = g(tf_j, nb_doc_j, nb_elt_j, nb_termes, nb_termes_tot, nb_termes_doc) \quad [5]$$

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 nœuds 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 booléennes de type P1. Le modèle m utilisé pourra ensuite 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). Une solution simple serait par exemple de renvoyer le nœud étant le plus proche ancêtre commun de nœuds feuilles ayant un score de pertinence non nul pour une requête q .

6.3. Traitement de la structure dans les requêtes

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

Comme nous l'avons vu précédemment, le traitement de la structure dans les requêtes passe d'abord par le traitement des sous-requêtes élémentaires $SRE_{i,j}$ formant les requêtes de type P2. 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)

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

où : - p_n est le score 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 $F_k(RSV_m(q, nf_k), dist(n, nf_k))$ permet de propager et d'agréger les scores de pertinence des nœuds feuilles nf_k descendants de n pour former le score de pertinence du nœud n . Les scores sont calculés d'après [4], et la propagation des scores se fait en fonction des distances $dist(n, nf_k)$ qui séparent le nœud n des nœuds feuilles nf_k dans l'arbre du document (c'est à dire le nombre d'arcs dans l'arbre du document nécessaires pour joindre n et nf_k).

Où $agreg_{ET}(p_n, p_m, dist(l,n), dist(l,m)) = p_l$ et $agreg_{OU}(p_n, p_m) = p_l$ définissent la façon dont les pertinences p_n et p_m des nœuds n et m sont agrégées pour former une nouvelle pertinence p_l .

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

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

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

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

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.

Considérons la requête $P2_3$ issue de notre exemple $P4.1$: $P2_3 = (SRE_{3,1} = \text{para}[\text{google}]) \text{ ET } (SRE_{3,2} = \text{titre}[\text{"moteurs de recherche"}])$. L'ensemble résultat de la requête $SRE_{3,1}$ est $R_{3,1} = \{(n11, p_{n11})\}$, et l'ensemble résultat de la requête $SRE_{3,2}$ est composé de deux nœuds $R_{3,2} = \{(n2, p_{n2}), (n9, p_{n9})\}$, et ce grâce à l'utilisation de l'index Dictionnaire sur l'élément *titre* (la recherche s'est alors effectuée sur des éléments *titre* et des éléments *sous-titre*). L'ensemble R_3 résultat de $P2_3$ sera alors composé de deux nœuds et des pertinences associées, comme le montre la figure 6 ci-dessous. $R_3 = R_{3,1} \oplus_{ET} R_{3,2} = \{(n1, p_{n1}), (n8, p_{n8})\}$, où $p_{n1} = \text{agreg}_{ET}(p_{n2}, p_{n11}, dist(n1, n2), dist(n1, n11)) = \text{agreg}_{ET}(p_{n2}, p_{n11}, 1, 3)$ et $p_{n8} = \text{agreg}_{ET}(p_{n9}, p_{n11}, dist(n8, n9), dist(n8, n10)) = \text{agreg}_{ET}(p_{n9}, p_{n11}, 1, 1)$.

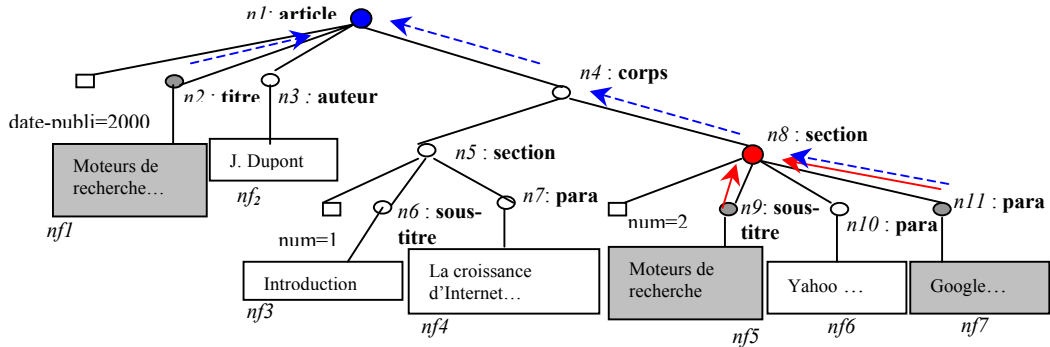


Figure 6 : création de l'ensemble résultat R_3 de la requête $P2_3$

La fonction $agreg_{ET}(p_n, p_m, dist(l,n), dist(l,m)) = p_l$ calcule un nouveau score de pertinence pour le nœud l à partir de deux pertinences p_n et p_m et de la distance qui sépare l de n et m . Il pourra par exemple s'agir d'une fonction produit de la forme :

$$\frac{p_n}{dist(l,n)} * \frac{p_m}{dist(l,m)}, \text{ ce qui dans notre exemple donnerait: } p_{n1} = (p_{n2} * p_{n11}) / (1 * 3) \text{ et } p_{n8} = (p_{n9} * p_{n11}) / (1 * 1)$$

La fonction $agreg_{OU}(p_n, p_m) = p_l$ pourra quant à elle être une simple fonction somme.

6.3.3. Traitement d'une requête de type P3

Les ensembles résultats des requêtes de type P2 sont ensuite utilisés pour le traitement des requêtes de type P3. Le résultat de telles requêtes est ainsi 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 $R_{i+1} = \{ (m, p_m) \}$
 $R_i \nabla R_{i+1} = \{ (n, p_n) / n \in R_i \text{ estAncêtre de } m \in R_{i+1} \text{ et } p_n = \text{prop_agg}(\text{dist}(m,n), p_n, p_m) \}$ [13]
Où $\text{prop_agg}(\text{dist}(m,n), p_n, p_m) \rightarrow p_n$ 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_n du nœud n .

L'ensemble résultat R d'une requête de type P3 est alors défini ainsi :

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

ce qui revient en fait à *propager de bas en haut dans l'arbre du document* les poids des nœuds résultats des sous-requêtes $P2_2$ à $P2_n$ vers les nœuds résultats de $P2_1$, qui constitueront l'ensemble renvoyé à l'utilisateur.

6.3.4. Traitement d'une requête de type P4

Alors que pour les requêtes de type P3, les scores des nœuds sont propagés de *bas en haut* dans l'arbre du document, dans le cas de requêtes de type P4, ces scores peuvent être propagés de *haut en bas*, et ce à cause de la présence d'un élément cible, qui indique le type de nœud à renvoyer à l'utilisateur. Ceci nécessite la définition de l'opérateur non-commutatif Δ défini ci-dessous:

Définition 3 : Soient deux ensembles de paires (nœud, pertinence) $R_i = \{ (n, p_n) \}$ et $R_{i+1} = \{ (m, p_m) \}$
 $R_i \Delta R_{i+1} = \{ (m, p_m) / m \in R_{i+1} \text{ estDescendant de } n \in R_i \text{ et } p_m = \text{prop_agg}(\text{dist}(n,m), p_n, p_m) \}$ [15]

Ainsi, l'opérateur Δ est utilisé pour *propager de haut en bas dans l'arbre du document* les poids des nœuds résultats de sous requêtes $P2_1$ à $P2_{i-1}$ vers les nœuds résultats de $P2_i$, qui constituent les éléments cibles demandés par l'utilisateur.

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

- (1) Propagation des poids des nœuds des ensembles R_{i+1}, \dots, R_n de *bas en haut* vers les nœuds de l'ensemble constitué des éléments cibles R_i : $SR_1 = R_i \nabla (R_{i+1} \nabla (R_{i+2} \nabla \dots))$
- (2) Propagation des poids des nœuds des ensembles R_1, \dots, R_{i-1} de *haut en bas* vers les nœuds de l'ensemble constitué des éléments cibles R_i : $SR_2 = (((R_1 \Delta R_2) \Delta R_3) \Delta \dots) \Delta R_i$
- (3) Union des deux ensembles créés précédemment: $R = SR_1 \cup SR_2$ [16]

L'ensemble résultat R de la requête P4.1 est ainsi obtenu de la façon suivante. Nous avons, à l'issue des étapes précédentes, $R_1 = \{(n1, l)\}$, $R_2 = \{(n4, p_{n4})\}$, $R_3 = \{(n1, p_{n1}), (n8, p_{n8})\}$. On a alors $SR_1 = R_2 \nabla R_3 = \{(n4, p_{n4})\}$, avec $p_{n4} = prop_agg(dist(n4, n8), p_{n4}, p_{n8}) = prop_agg(l, p_{n4}, p_{n8})$. Notons que la paire $(n1, p_{n1})$ faisant partie de l'ensemble R_3 est ignorée, car $n1$ n'est pas un nœud descendant de $n4$. On a ensuite : $SR_2 = R_1 \Delta R_2 = \{(n4, p'_{n4})\}$, avec $p'_{n4} = prop_agg(dist(n4, n1), p_{n4}, p_{n1}) = prop_agg(l, p_{n4}, l)$. Finalement, $R = \{(d, p_{n4} + p'_{n4})\}$, comme le montre la figure 7 ci-dessous.

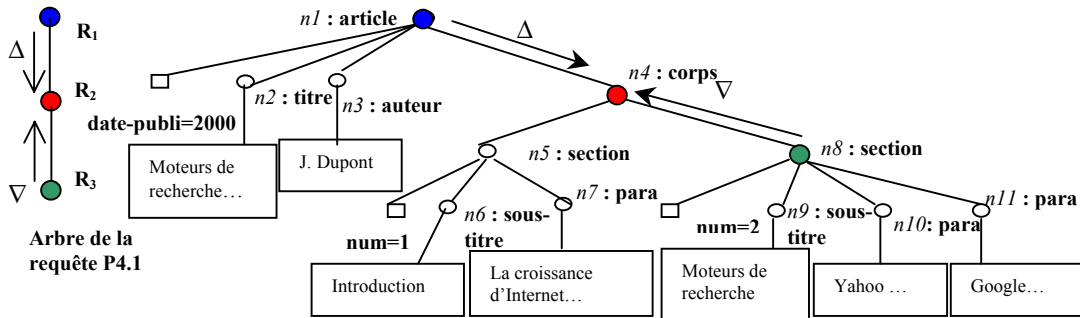


Figure 7 : construction de l'ensemble résultat de la requête P4.1 : comparaison de l'arbre du document et de l'arbre de la requête

La fonction $prop_agg(dist(n, m), p_n, p_m)$ utilise elle-aussi 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). De façon simple, on peut prendre : $p = (p_n + p_m) / dist(n, m)$. On aura alors : $p_{n4} = prop_agg(dist(n4, n8), p_{n4}, p_{n8}) = (p_{n4} + p_{n8}) / l$ et $p'_{n4} = prop_agg(dist(n4, n1), p_{n4}, p_{n1}) = (p_{n4} + p_{n1}) / l$.

Cette fonction permet 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.

7. Conclusion

Dans cet article, nous avons présenté le langage de requête XFIRM pour l'interrogation de corpus XML. Il permet de calculer des poids de pertinences aux parties de documents XML (ou unités d'information) renvoyées à l'utilisateur, et donc d'obtenir des listes triées de résultats. La syntaxe du langage repose sur quatre types de requêtes: des requêtes composées de simples *mots-clés* (requêtes de type P1), des requêtes contenant des *conditions sur la structure et le contenu de cette structure* (requêtes de type P2), des requêtes contenant la *notion de hiérarchie* dans l'expression des besoins sur la structure des documents (requêtes de type P3) et enfin des requêtes permettant de *spécifier l'élément que l'on souhaite voir retourné*

(requêtes de type P4). Dans les requêtes de type P1, P2 et P3, le type de l'élément retourné est calculé par le système. Dans tous les cas, le but du système est de renvoyer les unités d'information répondant de la façon la plus spécifique et exhaustive possible à la requête. La syntaxe du langage reste simple pour un utilisateur n'ayant pas de besoins structurels précis (requêtes de type P1 ou P2), contrairement aux autres langages proposés dans la littérature, qui reposent tous sur SQL. Dans le cas des requêtes de type P3 et P4, l'utilisateur n'a pas besoin d'exprimer ses conditions sur la hiérarchie du document de façon stricte, le système interprétant les conditions de chemins *vagues*. L'utilisateur n'a ainsi pas besoin de connaître parfaitement la DTD associée aux documents qu'il interroge. L'utilisation d'un Dictionnaire simplifié aussi sa tâche lorsque les corpus interrogés contiennent des documents ne suivant pas la même DTD ou alors lorsque des balises peuvent être considérés comme équivalentes dans un même document.

Le prototype de XFIRM est actuellement en cours d'implémentation. Les informations stockées dans les index nous permettront de tester de nombreux modèles de RI, qui pourront être évalués grâce à la campagne d'évaluation INEX (INitiative for the Evaluation of XML Retrieval).

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.
- S. Amer-Yahia, C. Botev, J. Shanmugasundaram. " TeXQuery: A Full-Text Search Extension to Xquery ". WWW 2004 (to appear). <http://www.cs.cornell.edu/database/TeXQuery/>
- Bradley, N, *The XML Companion*. Addison Wesley Professional Publisher, 864 p, 2001.
- Buneman, P., Davidson, S., Hillebrad, G., Suci, D., "A query language and optimisation techniques for unstructured data". *ACM-SIGMOD record*, pp. 505-516, Montréal, 1996.
- Ceri, S., Comai, S., Damiani, E., Fraternali, 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 Int. Workshop on WWW and databases*. p.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.
- Cluet, S., Delobel, C., Simeon, J., Smaga, K., "Your mediators need data conversion". *In Proc. ACM SIGMOD*, pp. 171-188, Seattle, 1998.
- Fuhr, N., Grossjohann, K. "XIRQL: A query Language for Information Retrieval in XML Documents". *In Proc. of the 24th annual ACM SIGIR*, New Orléans, p.172-180, 2001.

- 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.
- 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 theory of evidence applied to structured documents: modelling uncertainty". In *Proc. ACM-SIGIR*, pp. 110-118, Philadelphia, 1997.
- Levy, A., Fernandez, M., Suci, D., Florescu, D., Deutsch, A. XML-QL : A query language for XML. W3C technical report, Number NOTE-xml-ql-19980819, 1998.
- Li, Q., Moon, B., "Indexing and querying XML data for regular path expressions". In *Proc. Of the 27th International Conference on Very Large Data Bases*, 2001, Italy, p. 361-370.
- 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., McGill, M.J., *Introduction to modern information retrieval*. McGraw-Hill Int. Book Co, 1984
- Sauvagnat, K., "XFIRM, un modèle flexible de Recherche d'Information pour le stockage et l'interrogation de documents XML", *CORIA'04*, Toulouse, France, à paraître.
- 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). XML Path Language (XPath), Version 1.0. W3C Recommendation, Novembre 1999. Disponible sur : <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, fév 2003. <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. <http://www.w3c.org/TR/xpath-datamodel/>
- World Wide Web Consortium (W3C). XQuery 1.0 : an XML query language. W3C Working Draft, Novembre 2003. <http://www.w3.org/TR/xquery/>