Une extension de XQuery pour la recherche textuelle d'information dans des documents XML

Nicolas Faessel*, Jacques Le Maitre**

*LSIS (UMR CNRS 6168)
Université Paul CézanneDomaine Universitaire de Saint-Jérôme
Avenue Escadrille Normandie-Niemen
13397 Marseille Cedex 20
nicolas.faessel@lsis.org
**LSIS (UMR CNRS 6168)
Université du Sud Toulon-Var
BP 20132, 83957 La Garde
lemaitre@univ-tln.fr

Résumé. Nous présentons dans cet article une extension de XQuery que nous avons développée pour interroger le contenu et la structure de documents XML. Cette extension consiste à intégrer dans XQuery le langage NEXI, un sous-ensemble de XPath, défini dans le cadre de l'initiative INEX. Notre proposition est double : (i) équiper NEXI d'une sémantique floue, (ii) intégrer NEXI dans XQuery au moyen d'une métafonction appelée *nexi*, ayant une requête NEXI comme paramètre, et d'une extension de la clause *for* de l'opérateur FLWOR de XQuery. De plus, nous décrivons le prototype paramétrable que nous avons développé au dessus de deux moteurs XQuery classiques : Galax et Saxon.

1 Introduction

Il y a deux visions d'un document XML : une vision « centrée données » et une vision « centrée document ». Les documents XML « centrés données » sont constitués d'un ensemble d'éléments ayant une structure régulière : un ensemble de fiches bibliographiques, par exemple. Les documents XML « centrés document » décrivent des textes plus ou moins structurés : des livres scientifiques, par exemple. Pour interroger des documents XML « centrés données », le langage de requêtes XQuery (le SQL de XML), défini par le W3C (W3C, 2006b), est tout à fait bien adapté. Par contre, pour interroger des documents XML « centrés document » XQuery n'est pas suffisant lorsque l'interrogation est de nature sémantique, comme par exemple, la recherche des chapitres de livres qui concernent un certain sujet. De telles requêtes sont traitées traditionnellement par les systèmes de recherche d'information (Baeza-Yates et Ribeiro-Neto, 1999). Ce constat a conduit le W3C à proposer une extension de XQuery, XQuery Full-Text (W3C, 2006a), pourvue de fonctionnalités de recherche plein-texte. Le cœur de XQuery Full-Text est une fonction nommée *ftcontains* qui permet de tester si le contenu textuel d'un élément est conforme à une requête exprimée à l'aide d'opérateurs spécifiques : troncatures, connecteurs logiques, calcul de distance entre

mots, etc. La fonction *ftcontains* retourne un degré de similarité entre l'élément et la requête qui peut être strict (appartient à {0, 1}) ou flou (appartient à [0, 1]). Cependant, XQuery Full-Text n'impose pas la façon de calculer ce degré de similarité, qui dépend donc de l'implémentation.

Parallèlement, dans le cadre de l'initiative INEX pour l'évaluation de la recherche d'information XML[1], un langage de recherche d'information spécifique nommé NEXI (Trotman et Sigurbjörnsson, 2005), a été proposé. Ce langage est une version simplifiée de XPath (W3C, 1999), qui intègre une fonction *about* permettant de calculer le degré de similarité d'un élément XML avec une requête textuelle restreinte à un sac de mots éventuellement marqués d'un indicateur de présence ou d'absence. Pour ce langage aussi, la façon de calculer ce degré de similarité n'est pas imposée.

Dans (Le Maitre, 2005), nous avons proposé d'étendre le langage XQuery en y intégrant le langage NEXI muni d'une sémantique floue basée sur le modèle de recherche d'information vectoriel (Baeza-Yates et Ribeiro-Neto, 1999). Cet article améliore cette proposition sur le plan formel et décrit le prototype que nous avons construit au-dessus d'un moteur XQuery classique : Galax (Fernandez et al., 2003) et Saxon (Saxon, 2006), à l'heure actuelle.

Cet article est organisé de la façon suivante. Dans la section 2, nous décrivons le langage NEXI. Dans la section 3, nous proposons une sémantique pour ce langage, basée sur le modèle vectoriel et la logique floue. Dans la section 4, nous montrons comment intégrer une requête NEXI dans XQuery. Dans la section 5, nous présentons le prototype que nous avons réalisé afin de valider notre proposition. Enfin, dans la section 6, nous concluons et dressons quelques perspectives.

2 NEXI

Le langage NEXI (Trotman et Sigurbjörnsson, 2005), a été défini dans le cadre de l'initiative INEX dans le but d'effectuer de la recherche d'information dans des documents XML. Il permet d'exprimer deux types de requêtes : les requêtes « Content Only (CO) » qui ne portent que sur le texte d'un document et les requêtes « Content and Structure (CAS) » qui portent à la fois sur le texte d'un document et sur sa structure.

Une requête CO est une liste de termes, éventuellement affectés d'un indicateur imposant une présence ou une absence de ce terme dans le contenu textuel du fragment de document interrogé. Voici un exemple de requête CO :

```
+XML XQuery -SGML
```

Une requête CO s'applique à un document XML et a pour réponse la conformité du contenu textuel de ce document avec cette requête. La forme de la réponse à une requête CO et la façon de calculer cette réponse ne font pas partie de la spécification du langage NEXI. Cette réponse pourrait être le degré de similarité du document interrogé avec la requête.

Une requête CAS est une expression XPath simplifiée qui a la forme suivante dans sa version la plus générale :

$$//nt_1[p_1]//nt_2[p_2]$$

[1] http://qmir.dcs.qmw.ac.uk/INEX/index.html

Cette requête s'applique à un arbre de document XML et retourne les nœuds de cet arbre de type nt_2 conformes au prédicat p_2 qui sont des descendants des nœuds de type nt_1 conformes au prédicat p_1. Les prédicats p_1 et p_2 sont construits à partir de prédicats textuels ou numériques connectés par des opérateurs de conjonction ou de disjonction. Un prédicat textuel a la forme suivante :

`about(`*chemin de localisation*`,` *requête CO*`)`

et un prédicat numérique a la forme suivante :

chemin de localisation connecteur littéral numérique

Ici encore, la forme de la réponse à une requête CAS et la façon de la calculer ne font pas partie de la spécification du langage NEXI. La réponse pourra être une séquence d'éléments de type nt_2 affectés de leur degré de similitude avec la requête.

Considérons par exemple un document XML, enregistré dans le fichier « actes.xml », conforme à la DTD suivante :

```
<!element actes (article+)>
<!element article (titre, année, sec+)>
<!element titre (#PCDATA)>
<!element auteur (#PCDATA)>
<!element sec (titre, (sec | para)+)>
<!element année (#PCDATA)>
<!element para (#PCDATA)>
```

Adressée à ce document, la requête CAS suivante :

```
//article[about(.//titre, XML) and année < 2000]
//sec[about(.//para, SGML)]
```

retourne les sections contenant un paragraphe qui concerne SGML contenues dans un article dont le titre concerne XML et qui a été publié avant 2000.

3 Sémantique d'une requête NEXI

Nous proposons de munir les requêtes NEXI d'une sémantique floue (Bosc et al., 2004) basée sur le modèle vectoriel de recherche d'information (Baeza-Yates et Ribeiro-Neto, 1999).

Dans le langage NEXI original, une requête NEXI est adressée à un document XML dans son entier. Mais dans le but d'intégrer NEXI à XQuery, nous considérons qu'une requête NEXI est adressée à une séquence de nœuds éléments de l'arbre d'un document XML dont chaque nœud texte est supposé être annoté par un sac de termes obtenu en extrayant les mots du texte contenu dans ce nœud, et en les normalisant par lemmatisation ou par racinisation. Notons que le fait de ne considérer qu'un seul document n'est pas restrictif, car lorsque l'on souhaite interroger plusieurs documents, il est toujours possible de les rassembler dans un document unique.

Soit d l'arbre du document interrogé et M le nombre de termes indexant les textes contenus dans les nœuds textes de d. Conformément au modèle vectoriel, on associe à chaque nœud texte ou élément n_j de d un vecteur à M dimensions $\mathbf{v_j} = (w_{1j}, \ldots, w_{Mj})$ où w_{ij} est

le poids du terme t_i dans le nœud n_j ($w_{ij} \in [0, 1]$). Nous plaçant dans le cadre de la logique floue, nous interprétons le vecteur associé au n_j comme étant la conjonction floue $p_{1j} \wedge \ldots \wedge p_{Mj}$ où chaque p_{ij} est un prédicat flou dont la valeur de vérité est le poids w_{ij} du terme t_i dans le nœud n_j. Ce poids est calculé de la façon suivante :

- Si n_j est un nœud texte et lt_j est le sac de termes annotant ce nœud, w_{ij} est calculé selon la formule classique du modèle vectoriel : $w_{ij} = tf_{ij} \times ief_i$ où tf_{ij} (« term frequency ») est d'autant plus grand que le terme t_i est fréquent dans lt_j et ief_i (« inverse element frequency ») est d'autant plus petit que le terme t_i est fréquent dans les nœuds texte de d. Remarquons le remplacement du facteur *idf* (« inverse document frequency ») qui n'a pas de sens dans le cas d'un document unique par le facteur *ief* proposé dans (Trotman, 2005). Soit f_{ij} le nombre d'occurrences du terme t_i dans lt_j et $fmax_j$ le nombre maximum d'occurrences d'un même terme dans lt_j, le facteur tf_{ij} est calculé par la formule : $tf_{ij} = f_{ij} / fmax_j$. Soit nbt le nombre de nœuds texte de d et nbt_i le nombre de nœuds texte de d qui sont annotés par le terme t_i, le facteur ief_i est calculé par la formule : $ief_i = log\ (nbt / nbt_i)$.
- Si n_j est un nœud élément, le poids w_{ij} est calculé par fusion des poids $w'_{i1},\ldots, w'_{ik}$ des k nœuds enfants (texte ou élément) du nœud n_j. Cette fusion est interprétée comme la disjonction floue des conjonctions de prédicats associées à chacun de ces nœuds. Par exemple, si l'on choisit la fonction *max* comme opérateur de s-norme, on aura $w_{ij} = max(w'_{i1},\ldots, w'_{ik})$, ce qui correspond à la disjonction des valeurs de vérité des nœuds enfants pour le terme t_i.

Un vecteur $\boldsymbol{v_q} = (w_{1q}, \ldots, w_{Mq})$ est associé à chaque requête CO q où w_{iq} est égal à ief_i si le terme t_i est présent dans la requête q, ou à 0, s'il ne l'est pas. La valeur d'une requête CO q adressée à un nœud n_j auquel est associé un vecteur $\boldsymbol{v_j}$ est égale au cosinus de l'angle entre les vecteurs $\boldsymbol{v_j}$ et $\boldsymbol{v_q}$.

Dans le langage XQuery + NEXI, une requête CAS a la forme suivante :

$$e//nt_1[p_1]//nt_2[p_2]$$

où e est une expression XQuery qui doit avoir pour valeur une séquence de nœuds élément.

La valeur d'une requête CAS est une séquence floue de nœuds de l'arbre du document interrogé. Nous appelons séquence floue de nœuds une séquence S de la forme :

$$(n_1/\mu_1, \ldots, n_k/\mu_k)$$

dans laquelle, pour $i = 1$ à k, n_i est un nœud et $\mu_i = \mu_S(n_i)$ où μ_S est la fonction d'appartenance à S. Nous parlons de séquence floue au lieu d'ensemble flou, car les nœuds sont rangés dans l'ordre du document qui correspond à un parcours préfixe de l'arbre de ce document.

La valeur d'une requête CAS $e//nt_1[p_1]//nt_2[p_2]$ est obtenue de la façon suivante (rappelons que `//` est la notation abrégée du pas `/descendant-or-self::node()/`) :

- Soit $(e_1, \ldots, e_n)$ la séquence de nœuds valeur de l'expression e, le premier pas du chemin de localisation $//nt_1[p_1]//nt_2[p_2]$ est appliqué à la séquence floue de nœuds $(e_1/1, \ldots, e_n/1)$ et les pas suivants sont appliqués à la séquence floue de nœuds produite par le pas précédent.
- La valeur de la requête est celle de son dernier pas.
- La valeur d'un pas de localisation appliqué à une séquence floue de nœuds est la séquence floue de nœuds obtenue en appliquant ce pas à chacun des nœuds de cette séquence floue et en concaténant les séquences floues obtenues.

- Un pas de localisation *a*`::`*nt*`[`*p*`]` appliqué à un nœud *n* d'une séquence floue *S*, a pour valeur la séquence floue *S*' constituée des nœuds *n*' tels que *n*' appartient à l'axe *a*, vérifie le test de nœud *nt* et tel que $\mu_{S'}(n') = t\text{-}norme(\mu_S(n), valeur(p))$ où *p* est évalué dans le contexte où *n*' est le nœud contexte. Ceci, parce que le parcours d'un pas de localisation est traité comme une semi-jointure (et donc comme un produit cartésien) entre les nœuds sources de ce pas et l'ensemble des nœuds de l'arbre du document qui appartiennent à l'axe *a* et vérifient le test de nœud *nt*.
- La conjonction `and` et la disjonction `or` sont remplacées respectivement par les opérateurs de *t-norme* et de *s-norme* choisis puis évalués. Si ce sont les fonctions *min* et *max*, les valeurs de « p_1 `and` p_2 » et de « p_1 `or` p_2 » sont respectivement égales à celles de $min(p_1, p_2)$ et $max(p_1, p_2)$.
- La valeur d'un prédicat `about(`*e*`,` *q*`)` est celle de la requête CO *q* appliquée au nœud valeur de l'expression *e*.
- Un prédicat numérique est évalué de manière stricte : sa valeur appartient à [0, 1].

Considérons, par exemple, le document « actes.xml » et supposons que son arbre ne contienne qu'un seul nœud de type `article` ayant uniquement deux nœuds descendants de type `para`. Soit respectivement *a*, p_1 et p_2 ces trois nœuds. Supposons de plus que la valeur du prédicat `about(.//titre, XML)` soit 0.5 pour le nœud *a* et que la valeur du prédicat `about(.,SGML)` soit 0.2 pour le nœud p_1 et 0.8 pour le nœud p_2. La valeur de la requête CAS :

```
fn:doc("actes.xml")//article[about(.//titre, XML)]
                    //para[about(., SGML)]
```

aura pour valeur la séquence floue de nœuds :

(p_1/0.2, p_2/0.5)

si l'opérateur de *t-norme* est *min*.

4 Intégration de NEXI dans XQuery

L'intégration de NEXI dans XQuery est faite au moyen de la métafonction `nexi` et d'une extension de la clause `for` de l'opérateur FLWOR, similaire à celle de XQuery Full-Text (W3C, 2006a).

L'appel de la métafonction `nexi` a la forme suivante :

`nexi(`*q*`,` *t*`)`

où *q* est une requête CAS ayant la forme définie au paragraphe 3 et *t* est une expression XQuery dont la valeur appartient à [0, 1] et qui spécifie un seuil de pertinence. Cet appel retourne la séquence floue de nœuds résultant de l'évaluation de *q* suivie du filtrage des nœuds dont le degré d'appartenance est supérieur ou égal à *t*.

Pour accéder aux nœuds de cette séquence et à leur degré d'appartenance, il faut utiliser la clause `for` étendue suivante :

`for $`*n* `score $`*s* `in nexi(`*q*`,` *t*`)`

qui lie successivement chaque nœud de la séquence floue de nœuds résultant de l'appel `nexi(`*q*`, `*t*`)` à la variable `$`*n* et le degré d'appartenance de ce nœud à la variable `$`*s*.

Par exemple, la requête: « Titres et année des articles concernant SGML avec un seuil de pertinence de 0,5, triés par pertinence décroissante », peut s'exprimer de la façon suivante :

```
for $a score $s in
   nexi(fn:doc("actes.xml")//article[about(.,SGML)], 0.5)
order by $s descending
return <article>{$a/titre, $a/année}</article>
```

Montrons maintenant que l'on peut traduire en XQuery standard l'appel de la métafonction `nexi` et la clause `for` étendue. Le principe consiste

1. à traduire l'appel `nexi(`*q*`, `*t*`)` en une expression XQuery qui retourne la séquence d'items (n_1, μ_1, …, n_k, μ_k), si l'évaluation de la requête *q* suivie du filtrage des nœuds dont le degré d'appartenance est supérieur ou égal à *t* produit la séquence floue (n_1/μ_1, …, n_k/μ_k),
2. à remarquer que dans cette séquence les nœuds ont un rang impair ($2k - 1$, avec k entier strictement positif) et que le degré d'appartenance d'un nœud de rang $2k - 1$ se trouve au rang $2k$.

Une requête CAS $e//nt_1[p_1]//nt_2[p_2]$ peut être traduite en la requête XQuery suivante :

```
for $e1 in e//nt1
let $score1 := translate[$e1, p1]
for $e2 in $e1//nt2
let $score2 := translate[$e2, p2],
let $score := s-norme($score1, $score2)
where $score > t
order by $score
return ($e2, $score)
```

où ***translate*** est une métafonction définie par les règles suivantes :

- ***translate*** [*e*, *p*] où *p* est un prédicat numérique = *e*/*p*,
- ***translate*** [*e*, `about(`*c*`, `*q*`)`] = `nexi:about(`*e*/*c*`, `*q*`)`,
- ***translate*** [e, p_1 `and` p_2] = *t-norme*(***translate***[e, p_1], ***translate***[*e*, p_2]),
- ***translate*** [e, p_1 or p_2] = *s-norme*(***translate***[*e*, p_1], ***translate***[*e*, p_2]),
- ***translate*** [*e*, `(`*p*`)`] = ***translate***[*e*, *p*].

où `nexi:about` est la fonction qui calcule la valeur d'une requête CO appliquée à un nœud élément ou texte.

Une clause `for $`*n* `score $`*s* `in nexi(`*q*`, `*t*`)` peut être traduite en la suite de clauses `for` et `let` suivante :

```
let $fs := traduction de l'appel nexi(q, t)
for $k in 1 to fn:count($fs) div 2
let $n := $fs[2 * $k - 1]
let $s := $fs[2 * $k]
```

5 Description du prototype

Le prototype que nous avons réalisé est composé :

- d'un préprocesseur qui reçoit en entrée une requête XQuery + NEXI et la traduit en une requête XQuery pure, en appliquant le processus de traduction défini au paragraphe 4 ci-dessus,
- d'un moteur XQuery,
- d'une interface utilisateur.

5.1 Le préprocesseur

Le processus d'évaluation d'une requête XQuery + NEXI est schématisé sur la figure 1. Le préprocesseur a été programmé en Java. L'analyse lexicale et syntaxique de l'appel à la métafonction `nexi` et de l'extension de la clause `for`, est réalisée au moyen des parseurs JFlex et Cup.

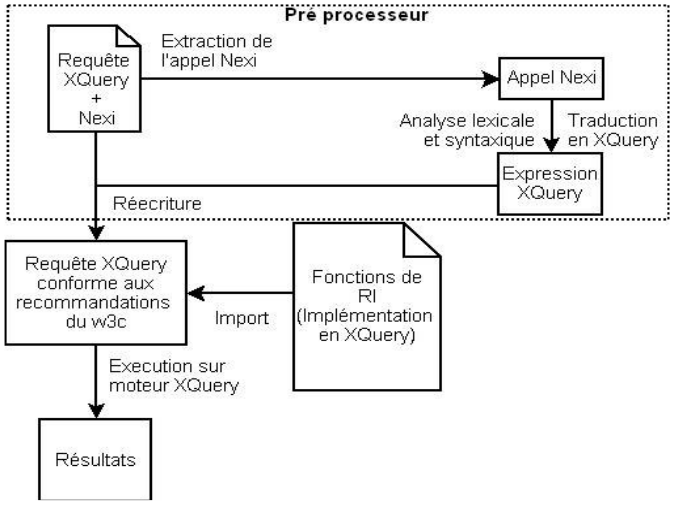

FIG. 1 – *Evaluateur de requête XQuery + NEXI*

L'évaluation d'une requête fait appel à un module XQuery de préfixe nexi qui contient les définitions des fonctions assurant les opérations liées à la recherche d'information textuelle : construction du sac de nœuds annotant un nœud texte, construction des vecteurs associés aux nœuds texte, aux éléments et aux requêtes CO, fonction `about`, etc.

5.2 Moteurs XQuery

Les caractéristiques des deux moteurs XQuery utilisables dans notre prototype, Galax et Saxon, sont :

- **Galax** traite des requêtes XQuery conformes aux recommandations du W3C. Il a été développé en Objective Caml, et fournit des APIs en Objective Caml, C et Java, ainsi que des exécutables binaires pré-compilés pour les différents systèmes d'exploitation. Le prototype fait appel au fichier binaire « galax-run.exe » pour évaluer la requête. L'avantage de ce type d'utilisation est que le traitement de la requête s'effectue sur les ressources système de la machine. L'inconvénient est que la portabilité du logiciel est dépendante des fichiers binaires dont on dispose. Un autre inconvénient réside dans le fait que l'implémentation de NEXI en XQuery nécessite l'utilisation de certaines fonctions mathématiques, comme par exemple le logarithme, qui ne sont pas fournies par Galax.
- **Saxon** est un processeur XQuery/XSLT conforme aux recommandations du W3C. Il a été développé en Java. Saxon est, entre autres, utilisé dans l'éditeur XML Oxygen. Le prototype utilise les APIs Java fournies par Saxon pour l'exécution des requêtes XQuery. L'avantage majeur de ce processeur est qu'il permet de lier des modules Java aux modules XQuery. Ainsi, la déclaration :

  ```
  declare namespace math = "java:java.lang.Math";
  ```

  permet d'utiliser la librairie *java.lang.math* fournie par Java et donc d'avoir accès à la fonction *log* pour calculer le facteur *ief*. Ce processeur est exécuté sur la Machine Virtuelle Java (JVM), et a donc l'avantage d'être portable sur tout système possédant cette machine. L'inconvénient est qu'il n'utilise que les ressources de la JVM, qui sont plus limitées que les ressources système.

5.3 Interface utilisateur

L'interface utilisateur se présente sous la forme de deux fenêtres : la fenêtre « Requête » et la fenêtre « Paramètres ».

La fenêtre « Requête » (Figure 2) est découpée en trois cadres.

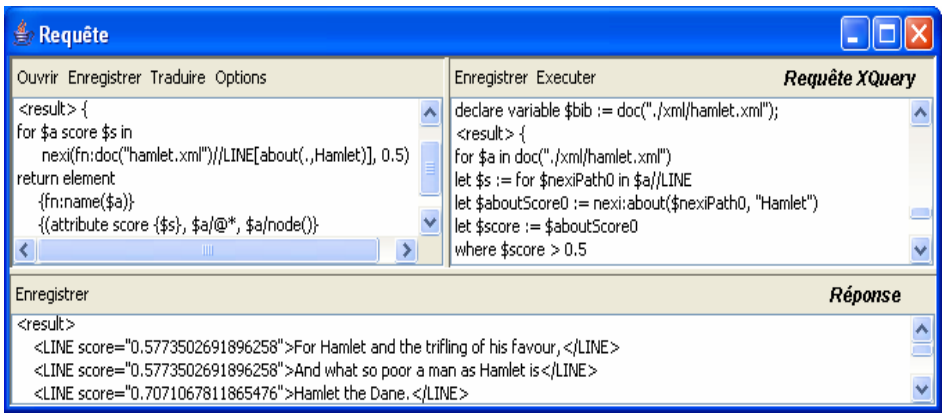

FIG. 2 – *Fenêtre « Requête »*

Le premier cadre sert à l'édition d'une requête XQuery + NEXI. Le deuxième cadre, à droite du premier, permet de visualiser la traduction de cette requête en XQuery pur et de la modifier si nécessaire pour effectuer des tests. Le troisième cadre contient les résultats de la requête, ou les erreurs rencontrées.

La fenêtre « Paramètres » (Figure 3) permet de choisir un moteur XQuery, et de fixer différentes stratégies d'évaluation d'une requête XQuery + NEXI.

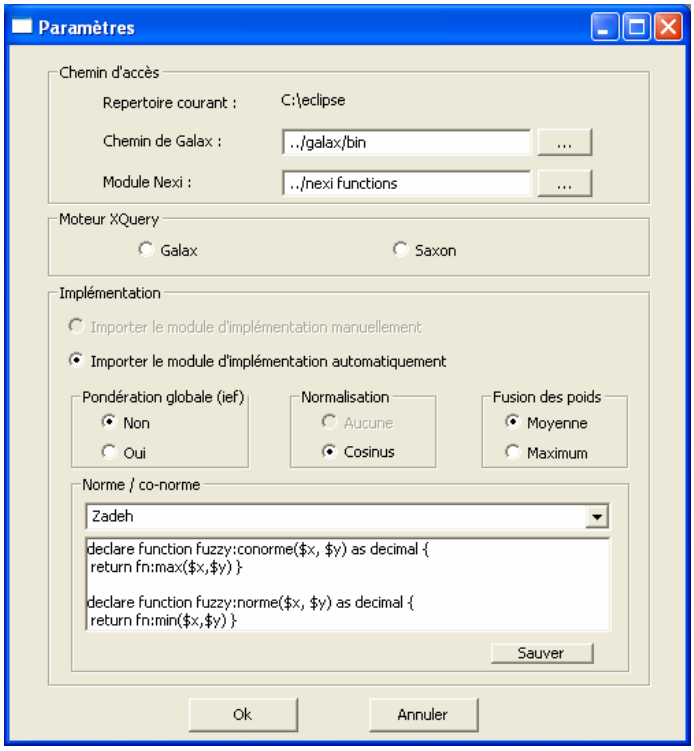

FIG. 3 – *La fenêtre « Paramètres »*

Les paramètres proposés permettent de choisir :

- Les chemins d'accès aux fichiers nécessaires à l'exécution de la requête.
- Le moteur XQuery sur lequel sera exécuté la requête : Galax ou Saxon, dans la version actuelle.
- La pondération globale d'un terme dans le vecteur associé à un nœud texte. Si la réponse est Non : le facteur *ief* a la valeur 1 et donc le poids d'un terme dans un nœud texte est égal à la fréquence de ce terme dans le sac de termes annotant ce nœud. Si la réponse est Oui : le facteur *ief* est calculé selon la formule indiquée au paragraphe 2 ci-dessus.

- Les fonctions utilisées pour calculer la *t-norme* et la *s-norme*. Trois choix sont offerts dans la version actuelle :
  - Zadeh : $t\text{-}norme(x, y) = min(x, y)$, $s\text{-}norme(x, y) = max(x, y)$,
  - Probabiliste : $t\text{-}norme(x, y) = xy$, $s\text{-}norme(x, y) = x + y - xy$,
  - Lukasiewicz : $t\text{-}norme(x, y) = max(x + y - 1, 0)$, $s\text{-}norme(x, y) = min(x + y, 1)$,
  - fonctions fournies par l'utilisateur.

  Ces fonctions sont utilisées dans l'indexation d'un document XML pour calculer la fusion des poids lors du calcul du vecteur associé à un nœud élément ainsi que pour évaluer les conjonctions, disjonctions et la semi-jointures intervenant dans une requête NEXI.

6 Conclusion et perspectives

Le prototype décrit dans cet article n'est qu'une ébauche de ce que pourrait être un véritable moteur de recherche d'information XML. Nous poursuivrons son amélioration sur les points suivants :

- **Utilisation de XQuery Full-Text**. XQuery Full-Text possède toutes les fonctions nécessaires à la recherche d'information plein texte. Cette extension de XQuery permet entre autres de lemmatiser ou raciniser des mots, de calculer les distances entre les mots, d'utiliser des thésaurus, etc. Il serait donc plus intéressant de s'appuyer sur un moteur XQuery Full-Text (Galatex, par exemple) que sur un moteur XQuery simple. De plus les moteurs XQuery Full-Text permettent la construction d'index, indispensables pour obtenir des temps de réponses raisonnables dans le cas de documents volumineux.
- **Prise en compte flexible de la structure des documents**. Actuellement les fragments de documents interrogés sont déterminés par le découpage en éléments de ce document et donc par sa structure logique. Or la structure sémantique d'un document n'est pas toujours analogue à sa structure logique. Certains éléments doivent plus leur existence à des raisons de forme qu'à des raisons de sens. Notre objectif est de permettre à l'utilisateur de plaquer sur la structure logique d'un document, une structure « sémantique » qui soit mieux adaptée à la recherche d'information dans ce document. La définition de cette structure pourrait par ailleurs permettre d'uniformiser les noms donnés aux types d'éléments qui peuvent être très variés pour désigner des éléments jouant la même fonction dans le découpage logique d'un document.
- **Evaluation**. Des campagnes d'évaluation sont régulièrement mises en place telles que TREC ou INEX. Nous envisageons d'y participer.

Références

Baeza-Yates, R., and B. Ribeiro-Neto (1999). *Modern Information Retrieval*. New York: Addison-Wesley.

Bosc, P., L. Liétard, O. Pivert et D. Rocacher (2004). *Gradualité et imprécision dans les bases de données. Ensembles flous, requêtes flexibles et interrogation de données mal connues*. Paris: Ellipses.

Fernandez, M., J. Simeon, B. Choi, A. Marian, and G. Sur (2003). Implementing XQuery 1.0: The Galax Experience. *Proceedings of the 29th International Conference on Very Large Data Bases* (Berlin, Germany, September 2003). 1077-1080.

Le Maitre, J. (2005). Indexing and Querying Content and Structure of XML Documents According to the Vector Space Model. *Proceedings of the IADIS International Conference WWW/Internet 2005* (Lisbon, Portugal, October 2005). II: 353-358.

Saxon (2006). The Saxon XSLT and XQuery Processor. http://www.saxonica.com/.

Trotman A. (2005). Choosing document structure weights. *Information Processing and Management*, 41(2): 243-264.

Trotman, A., and B. Sigurbjörnsson (2005). Narrowed Extended XPath I (NEXI), *Advances in XML Information Retrieval, 3rd International Workshop of the Initiative for the Evaluation of XML Retrieval (INEX 2004)*, *Lecture Notes on Computer Science 3493*, 16-40.

W3C (1999). *XML Path Language (XPath) Version 1.0*. W3C Recommendation 16 November 1999. http://www.w3.org/TR/1999/REC-xpath-19991116/.

W3C (2006a). *XQuery 1.0 and XPath 2.0 Full-Text*. W3C Working Draft 1 May 2006. http://www.w3.org/TR/2006/WD-xquery-full-text-20060501/.

W3C (2006b). *XQuery 1.0: An XML Query Language*. W3C Proposed Recommendation 21 November 2006. http://www.w3.org/TR/2006/PR-xquery-20061121/.

Summary

In this paper, we present the extension of XQuery that we have developed in order to query both content and structure of XML documents. This extension consists in integrating into XQuery the NEXI language, a subset of XPath, developed in the framework of the INEX initiative. Our proposition is twofold: (i) providing NEXI with a fuzzy semantics, (ii) integrating NEXI into XQuery by means of a predefined metafunction called *nexi* having an NEXI CAS query as parameter, and an extension of the *for* clause of the FLWOR operator. Moreover, we describe the parametrizable prototype that we developed in front of two classical XQuery engines: Galax and Saxon.