Extensão do XQuery com operações de selecção para a construção interactiva das perguntas

Alda Lopes Gançarski[1] and Pedro Rangel Henriques[2]

[1] Universidade do Minho, Braga, Portugal
email: aldalopes@di.uminho.pt
Membro do LIP6, Université Paris 6, Paris, França
[2] University do Minho, Braga, Portugal
email: prh@di.uminho.pt

Abstract. XQuery é a linguagem de interrogação proposta pelo W3C para XML usando restrições estruturais e sobre o conteúdo. XQuery está a ser complementado com a linguagem Full-Text para realizar operações sobre textos, tratando-os como uma sequência de palavras, sinais de pontuação e espaços. Dada a natureza complexa das perguntas estruturadas do XQuery, propõe-se, neste artigo, uma extensão para permitir a *selecção* do subconjunto interessante de elementos de cada resultado intermédio das perguntas. Os resultados intermédios são, portanto, acessíveis ao utilizador durante a construção das perguntas, o que torna mais fácil obter o resultado desejado. As operações de selecção são formalmente definidas, estendendo a gramática do XQuery e definindo novas funções. Estas definições podem ser usadas para construir um sistema de processamento adequado.

1 Introdução

O XQuery [4] é a proposição do W3C como linguagem de interrogação padrão para XML. Esta linguagem utiliza, entre outras funções, a definição de caminhos (*paths*), através da linguagem XPath [3], para aceder a elementos ou atributos dos documentos. Surgiram, entretanto, trabalhos que estendem estas linguagens com operações de similaridade textual da Recuperação de Informação tradicional [2], propondo métodos de cálculo de relevância, como os apresentados em [5]. Uma operação de similaridade textual consiste em verificar se um texto aborda um assunto expresso em linguagem natural. O resultado dessa verificação é a *relevância* do texto, normalmente um valor no intervalo [0, 1].

O W3C propõe a linguagem *Full-Text* como um complemento ao XQuery e ao XPath que inclui a possibilidade de associar um *score* (ou relevância) a uma expressão que verifica se uma dada frase existe no conteúdo de um elemento ou atributo. O cálculo da relevância fica a cargo da aplicação.

Contudo, a dificuldade na construção das perguntas estruturadas levou à definição da linguagem de interrogação interactiva IXDIRQL [6]. Esta linguagem tem por base o XPath, estendendo-o, não só com operações de similaridade

textual, mas também com um paradigma interactivo/iterativo de construção das perguntas. Com este paradigma, cada operação conduz a um resultado intermédio que o utilizador pode consultar para decidir acerca da próxima operação a efectuar, da alteração de alguma operação já introduzida ou da selecção do subconjunto de elementos interessantes dos resultados intermédios, até especificar a pergunta que conduz ao resultado desejado. Foi construído um protótipo para o processamento da IXDIRQL. Este protótipo foi posto ao dispor de utilizadores reais [7] para se verificar o seu funcionamento correcto e a correcta utilização das operações de selecção por parte dos utilizadores face a determinados pedidos de informação. Em [8] é sugerido estender o XQuery com o paradigma interactivo/iterativo de construção das perguntas, incluindo as operações de selecção e propondo um método para o cálculo das relevâncias. O presente artigo propõe uma definição formal das operações de selecção que pode ser usada para construir um sistema de processamento adequado. Esta definição tem por base as definições do XQuery e do Full-Text apresentadas em [4] e [1], respectivamente, seguindo, portanto, o mesmo formalismo.

O artigo é organizado da seguinte forma. A secção 2 introduz as linguagens XQuery e Full-Text. Depois, as secções 3 e 4 definem as operações de selecção *select* e *judgeRel*, respectivamente. A secção 5 propõe o processamento incremental para o XQuery estendido. Finalmente, é feita uma conclusão e são dadas algumas directivas para trabalho futuro.

2 As linguagens *XQuery* e *Full-Text*

O XQuery é formado por vários tipos de expressões, incluindo o XPath e expressões *for..let..where..order_by..return* (FLWOR) baseadas nas linguagens típicas de interrogação das bases de dados, como o SQL. Para passar informação de um operador para outro são usadas variáveis. Por exemplo, seja um documento sobre artigos que inclua o seu título, o seu autor e a sua editora. A pergunta que se segue recupera artigos do autor *Silva* ordenados pelo título respectivo[3]:

```
for $a in doc("http://...") /artigos/artigo
where $a/autor = "Silva"
order by $a/título
return $a
```

O XQuery opera sobre a estrutura lógica e abstracta dos documentos. O modelo de dados correspondente representa os documentos como árvores onde os nós podem corresponder a documentos, elementos, atributos, textos, espaços de nomes, instruções de processamento ou comentários. Cada nó tem um identificador único.

[3] Para simplificar, ao longo do artigo, algumas expressões e símbolos opcionais são subtituídos por "...".

A linguagem Full-Text estende o XQuery com expressões *ftcontains* e com variáveis de tipo *score* dentro das expressões FLWOR. A função *ftcontains* pode ser usada sempre que uma comparação é possível, como a igualdade. Uma expressão *ftcontains* inclui um caminho, para especificar os nós onde a função será aplicada, e a expressão com as frases (expressões em linguagem natural) a procurar no conteúdo desses nós. O valor lógico verdade é retornado se algum nó inclui a frase procurada. Por exemplo, a pergunta seguinte retorna o(s) autor(es) dos artigos cujo título contém *"linguagem XML"*.

for $a in doc("...") /artigos/artigo
where $a/título ftcontains "linguagem XML"
return $a/autor

Uma variável *score* guarda a relevância associada a uma expressão que consiste numa combinação Booleana de expressões *ftcontains*. Trata-se, portanto, de uma operação de similaridade textual. A variável que guarda a relevância fica associada a um valor de tipo *xs:float* (o espaço de nomes *xs* refere-se ao *XML Schema*) no intervalo [0, 1], um valor maior indicando uma maior relevância. A forma como a relevância é calculada é dependente da implementação. Por exemplo, a pergunta seguinte retorna artigos ordenados pela relevância do seu título em relação à expressão *"XML"*.

for $a score $s in doc("...") /artigos/artigo[título ftcontains "XML"]
order by $s
return $a

3 A função *Select*

O paradigma interactivo de construção de perguntas é baseado nas operações de selecção. Estas consistem na restrição dos resultados intermédios ao subconjunto de elementos interessantes. A selecção é feita nos caminhos através da função *mf:select* (o espaço de nomes *mf* é associado neste artigo às funções criadas).

A selecção do conjunto de elementos interessantes é feita segundo um determinado critério. Enquanto num filtro do XPath o conjunto de elementos é especificado por intenção, na função *mf:select* é-o por extensão, i.e., referindo explicitamente cada elemento. Isto pode ser interessante quando a especificação do critério é demasiado complicada (o utilizador pode, inclusivé, não saber fazê-lo) ou quando é mais rápido/eficiente referir os elementos desejados directamente.

Suponha-se que cada nó é identificado por uma palavra. A entrada de *mf:select* é um nó e uma lista de identificadores do nós; a saída é o nó dado como entrada se ele corresponde a um dos identificadores da lista de entrada. Por exemplo, suponha-se que o utilizador quer as referências bibliográficas dos artigos interessantes do autor *"Silva"*. O termo *interessante* pode-se referir, entre outras coisas, ao título, aos co-autores, à editora, à data de publicação ou ao tamanho. A per-

gunta pode, então, ser:

for $a in doc("...")/artigos/artigo[autor = "Silva"][mf:select(., ("a4", "a8"))]
return $a//referência

Nesta pergunta, a função *mf:select* selecciona os artigos identificados por *"a4"* e *"a8"*. Dada a natureza interactiva da função *mf:select*, a pergunta é escrita em três passos:

1. O utilizador especifica a cláusula *for* com o caminho que retorna os artigos do autor *"Silva"*: *for $a in doc("...")/artigos/artigo[author = "Silva"]*

2. Ao analisar a lista de artigos dada pelo caminho, o utilizador selecciona os artigos que lhe interessam usando *mf:select*:
for $a in doc("...")/artigos/artigo[autor = "Silva"][mf:select(., ("a4", "a8"))]

3. O utilizador completa a pergunta com a cláusula *return*:
for $a in doc("...")/artigos/artigo[autor = "Silva"][mf:select(., ("a4", "a8"))]
return $a//referência

Apesar de *mf:select* receber uma lista de identificadores de nós, o utilizador não é obrigado a conhecê-los se houver uma interface adequada para visualizar dos resultados intermédios. Esta interface deve permitir a selecção dos elementos usando, por exemplo, um botão associado a cada elemento. O sistema deve, então, escrever automaticamente os identificadores dos nós seleccionados na pergunta em construção no editor das perguntas.

3.1 Definição da sintaxe

O argumento de *mf:select* é um nó e uma lista de identificadores que consistem em nomes, i.e., cadeias de caracteres excluindo espaços. Esta definição está de acordo com a produção 91 da gramática do XQuery apresentada em [1], produção essa que permite derivar chamadas a funções:

[91] FunctionCall ::= <QName "("> (ExprSingle ("," ExprSingle))? ")"*

Nesta produção, o nome da função é derivado por *QName*. Este símbolo deriva nomes possivelmente associados a espaços de nomes. No caso da função *mf:select*, *QName* deriva o nome da função *select* associado ao espaço de nomes *mf*.

Cada argumento de uma função é derivado pelo símbolo *ExprSingle*. Este símbolo deriva numa expressão qualquer do XQuery, incluindo expressões que definem nós ou listas de valores do tipo *xs:string*. Então, os argumentos da função *mf:select* são derivados por *ExprSingle*. Cada identificador de nó é do tipo *xs:string*. Por sua vez, um valor de tipo *xs:string* é derivado por *StringLiteral*, a partir de *ExprSingle* após uma série de passos de derivação.

3.2 Definição da semântica

Seja *IdNodeTab* uma tabela mantida pelo sistema que faz corresponder a cada nó o seu identificador. Sendo *node()* o tipo de um qualquer nó, tem-se:

IdNodeTab : xs:string × *node()*

A função *mf:select* pode, então, ser definida por:

```
declare function mf:select($contextNode as node()?, $selectedIds as xs:string*)
as node( )?
{ for $s in $selectedIds
  let $n := IdNodeTab[$s]
  if ($contextNode=$n) then return $contextNode else return ( )
}
```

Nesta definição, *$selectedIds* é uma variável que contém a lista de identificadores do tipo *xs:string* dos nós explicitados como argumentos da função *mf:select*. A variável *$contextNode* guarda o nó que será retornado se o seu identificador (dado pela tabela *IdNodeTab*) se encontra na lista *$selectedIds*.

4 O operador *judgeRel*

O operador *judgeRel* selecciona o subconjunto de elementos julgados relevantes pelo utilizador na lista ordenada resultante duma expressão *ftcontains* associada a uma variável *score*. Seja a seguinte pergunta:

```
for $a score $s in doc("...")/artigos/artigo[título ftcontains "XML"]
order by $s
return $a//referência
```

Esta pergunta retorna a lista de referências ordenadas pela relevância do título do artigo onde elas são citadas. As referências retornadas correspondem ou não a títulos realmente relevantes, uma vez que a lista é baseada em relevâncias estimadas pelo sistema. Com o operador *judgeRel*, o utilizador pode seleccionar os elementos relevantes durante a construção da pergunta, analisando a lista ordenada dada pela função *ftcontains*. Desta forma, a relevância associada a elementos relevantes fica 1 e não relevantes fica 0. Estes novos valores de relevância são tidos em conta no cálculo final do *score*. Na pergunta precedente, suponha-se que o título identificado por *"t4"* é seleccionado quando o utilizador analisa a lista ordenada dada por *ftcontains*. A pergunta passa, então, a ser:

```
for $a score $s in
        doc("...")/articles/article[título ftcontains "XML" judgeRel ("t4")]
order by $s
```

return $a/referência

Aqui, as referências retornadas pela cláusula *return* pertencem a artigos onde o título é seguramente relevante (o utilizador julgou-o relevante e seleccionou-o) e podem ser usadas para processamento posterior. Como para a função *mf:select*, dada a natureza interactiva do operador *judgeRel*, esta pergunta é escrita em três passos:

1. O utilizador especifica a cláusula *for* e a expressão *ftcontains*:
   for $a score $s in doc("...")/artigos/artigo[título ftcontains "XML"]

2. A lista ordenada resultante de *ftcontains* dá ao utilizador um bom ponto de partida para escolher os títulos relevantes. Ao analisá-la, o utilizador insere o operador *judgeRel* com os títulos relevantes:
   for $a score $s in
   doc("...")/artigos/artigo[título ftcontains "XML" judgeRel ("t4")]

3. O utilizador escreve as cláusulas *order by* e *return* para obter a lista final de referências:
   for $a score $s in
   doc("...")/artigos/artigo[título ftcontains "XML" judgeRel ("t4")]
   order by $s
   return $a/referência

Da mesma forma que para a função *mf:select*, a janela que mostra a lista ordenada deve permitir ao utilizador seleccionar os elementos relevantes, não tendo de conhecer os identificadores dos nós.

4.1 Definição da sintaxe

O operador *judgeRel* tem de ser incluído na gramática da linguagem Full-Text apresentada em [1]. Nesta gramática, as produções 35, 37, 38 e 51 derivam a cláusula *for*, uma variável de tipo *score*, a cláusula *let* e a expressão *ftcontains*, respectivamente:

[35] ForClause ::= "for" "$" VarName ... FTScoreVar? "in" ExprSingle ...
[37] FTScoreVar ::= "score" "$" VarName
[38] LetClause ::= (("let" "$" VarName ... FTScoreVar?) |
("let" "score" "$" VarName)) ":=" ExprSingle ...
[51] FTContainsExpr ::= RangeExpr ("ftcontains" FTSelection
FTIgnoreOption?)?

Na produção 51: *RangeExpr* deriva a expressão que conduz à lista de nós onde *ftcontains* será aplicada; *FTSelection* deriva combinações booleanas de frases a procurar e opções de combinação (*match options*), tais como *case sensitive*; *FTIgnoreOption* permite a exclusão de certos nós dentro do conjunto retornado por

RangeExpr, conduzindo ao conjunto final de nós chamado *contexto de procura*. Nas produções 35 e 38, a variável *score* guarda a relevância associada à expressão derivada por *ExprSingle*. *ExprSingle* permite derivar uma expressão qualquer do XQuery. Contudo, as expressões associadas às variáveis de *score* são restringidas a combinações Booleanas de expressões *ftcontains*, envolvendo apenas "and" e "or". Consequentemente, propõe-se a substituição de *ExprSingle* nas produções 35 e 38 pelo símbolo *ScoreExpr*, obtendo-se:

[35] ForClause ::= "for" "$" VarName ... (FTScoreVar "in" ScoreExpr | "in" ExprSingle) ...
[38] LetClause ::= ("let" "$" VarName ... (FTScoreVar ":=" ScoreExpr | "=:" ExprSingle) | "let" "score" "$" VarName ScoreExpr) ...

O símbolo *ScoreExpr* deriva a combinação Booleana de expressões *ftcontains* através das produções seguintes:

ScoreExpr ::= ScoreOrExpr
ScoreOrExpr ::= ScoreAndExpr | ScoreAndExpr "or" ScoreOrExpr
ScoreAndExpr ::= ScoreExprUnit | ScoreExprUnit "and" ScoreAndExpr
ScoreExprUnit ::= "(" ScoreExpr ")" | RangeExpr ("ftcontains" FTSelection FTIgnoreOption? JudgeRelExpr?) ?

Assim como na gramática do XQuery especificada em [1], estas produções reflectem a precedência dos operadores. Os operadores definidos a mais baixo nível são os de maior precedência. Os símbolos *ScoreOrExpr* e *ScoreAndExpr* derivam um "or" e um "and", respectivamente. O símbolo *ScoreExprUnit* deriva a expressão *ScoreExpr* entre parêntesis ou deriva uma expressão *ftcontains* associada a variáveis *score*. Esta última expressão é similar às derivadas pela produção 51 mas agora aumentadas com o operador opcional *judgeRel*. O símbolo *JudgeRelExpr* deriva o operador *judgeRel* na seguinte produção:

JudgeRelExpr ::= "judgeRel" "(" StringLiteral ")"*

Nesta produção, *StringLiteral* deriva o identificador de nó, como referido na secção 3.1. *judgeRel* é, então, associado ao conjunto de identificadores de nós julgados relevantes pelo utilizador[4].

4.2 Definição da semântica

O operador *judgeRel* está incluído nas expressões associadas a variáveis *score*. Assim, a sua definição semântica é dada em conjunto. Contudo, a definição dessas expressões não pode ser feita em termos de XQuery porque requer a presença de funções de segunda ordem, i.e., funções que não avaliam os seus

[4] Pode acontecer que o utilizador não encontre elementos relevantes, sendo a lista de identificadores vazia.

argumentos como expressões regulares do XQuery, mas usam-nos de forma interpretada. Em [1] assume-se que existe a função de segunda ordem *fts:score* (*fts* refere-se a *Full-Text semantics*) que aceita como argumento uma expressão *ScoreExpr* e retorna a relevância (*score*) desta expressão. Dada esta função, a expressão genérica *score $var as ScoreExpr* é avaliada como se fosse substituída por *$var:=fts:score(ScoreExpr)*. Propõe-se, aqui, a seguinte definição da função *fts:score*:

```
declare function fts:score($e as xs:string) as xs:float
{
1. if (mf:operatorScore($e) = "or") then
2.         mf:scoreOr(fts:score(mf:operandLeftScore($e)),
3.                    fts:score(mf:operandRightScore($e)))
4. else if (mf:operatorScore($e) = "and") then
5.         mf:scoreAnd(fts:score(mf:operandLeftScore($e)),
6.                     fts:score(mf:operandRightScore($e)))
7.       else
8.               let $s := mf:nodeList($e)
9.               return
10.                    if mf:includesJudgeRel($e) then
11.                         let $j := mf:judgeRelIds($e)
12.                         let $i := { for $a in $j return IdNodeTab[$a] }
13.                         return mf:scoreJudgeRel($s, $i)
14.                    else
15.                         let $m := mf:matchExpr($e)
16.                          return mf:scoreFTContains($s, $m)
}
```

O argumento de *fts:score* é uma cadeia de caracteres correspondente à expressão derivada pelo símbolo *ScoreExpr* definido na secção 4.1. A função retorna um valor de tipo *xs:float*.

Devido às chamadas recursivas da função *fts:score* (linhas 2, 3, 5, 6), a relevância é calculada, primeiro, para cada expressão *ftcontains*, e, depois, para cada operador Booleano da expressão *ScoreExpr*, respeitando a precedência dos operadores, até se obter o resultado final.

Na linha 1, a função *mf:operatorScore* aceita uma expressão *ScoreExpr* e retorna o primeiro operador a avaliar: "and", "or" ou nenhum. Dependendo do operador, acções diferentes são tomadas. Se o operador é um "or" (linha 1), a relevância é dada pela função *mf:scoreOr* tendo como argumentos a relevância dos operandos do "or" (linhas 2 e 3). Estes operandos são dados pelas funções *mf:operandLeftScore* e *mf:operandRightScore*. Se o operador é um "and" (linha 4), uma acção similar é efectuada, sendo a relevância desta vez dada pela função *mf:scoreAnd* (linhas 5 e 6).

Se não houver nenhum operador Booleano, a relevância de uma expressão *ftcontains* derivada pelo símbolo *ScoreExprUnit* (definido na secção 4.1) é calcu-

lada pelas acções das linhas 7 a 16. A variável *$s* guarda a lista de nós do contexto de procura (linha 8). Esta lista é retornado pela função *mf:nodeList* que tem em conta, não só a expressão derivada por *RangeExpr* (apresentado na secção 4.1), mas também a expressão derivada por *FTIgnoreOption* (também apresentado na secção 4.1). A existência de um operador *judgeRel* é, então, verificada pela função *mf:includesJudgeRel* analisando a expressão *ScoreExpr*. Se existe o operador (linha 10), as acções seguintes são tomadas. A variável *$j* guarda os identificadores de nós julgados relevantes pelo utilizador (linha 11). Estes são dados pela função *mf:judgeRelIds* que recebe a expressão *ScoreExpr*. Outra variável, *$i*, guarda os nós correspondentes aos identificadores julgados relevantes (linha 12). Estes nós são dados pela tabela *IdNodeTab* (apresentada na secção 3.1). A função *mf:scoreJudgeRel* aceita o contexto de procura (guardado em *$s*) e a lista de nós julgados relevantes (guardada em *$i*) e retorna a relevância da cláusula *score* (linha 13).

Se não houver o operador *judgeRel* na expressão *ScoreExpr* (linha 14), a variável *$m* guarda a combinação Booleana das frases a procurar (e respectivas *match options*) derivadas pelo símbolo *FTSelection* (apresentado na secção 4.1). Esta combinação é dada pela função *mf:matchExpr* (linha 15). Usando a variável *$m*, a função *mf:scoreFTContains* calcula a relevância associada aos nós do contexto de procura (guardado em *$s*) (linha 16).

As novas funções invocadas dentro de *fts:score* não são definidas aqui mais em detalhe. A maior parte (*mf:operatorScore*, *mf:operandLeftScore*, *mf:operandRightScore*, *mf:includesJudgeRel*, *mf:judgeRelIds*, *mf:matchExpr* e *mf:ignoreOption*) baseia-se numa simples análise léxico-sintáctica da expressão *ScoreExpr* para encontrar sub-expressões específicas. A função *mf:nodeList* retorna o contexto de procura. As restantes funções (*mf:scoreOr*, *mf:scoreAnd*, *mf:scoreJudgeRel* e *mf:scoreFTContains*) são dedicadas ao cálculo da relevância e devem ser definidas pela aplicação.

4.3 Exemplo de processamento de *fts:score*

Ao longo das explicações desta e das próximas secções, por simplicidade, as acções da função *fts:score* definida na secção 4.2 são referidas pelo número da linha respectiva. Também, cada nó é referido pelo respectivo identificador.

Seja, por exemplo, a seguinte pergunta:

```
for $a score $s in doc("...")/artigos/artigo[referência ftcontains "XML"
        and secção ftcontains "Aplicação XML" judgeRel ("s1")]
order by $s
return $a/título
```

Esta pergunta retorna títulos de artigos onde as referências abordam *"XML"* e as secções referem *"Aplicação XML"*. Os títulos resultantes são ordenados pela sua relevância. Suponha-se que o artigo *a1* foi encontrado na cláusula *for* e que tem secções *s1* e *s2* e referências *r1* e *r2*. Quando a função *fts:score* é executada, o operador Booleano "and" é detectado pela função *mf:operatorScore* (linha 4). Assim, a função *fts:score* é chamada recursivamente para cada operando do "and"

(linhas 5 e 6). Estes operandos são sub-expressões de *ScoreExpr* dados pelas funções *mf:operandLeftScore* e *mf:operandRightScore*. Os resultados dados por *fts:score* são usados como argumentos da função *mf:scoreAnd* para calcular o resultado final da relevância para o artigo *a1* (linha 5). Se houver mais artigos, uma relevância é calculada para cada um, usando de novo a função *fts:score*.

Para ambos os operandos do "and", a função *fts:score* é executada a partir da linha 7 porque não há mais operadores Booleanos. No que diz respeito ao primeiro operando, a função *mf:nodeList* retorna o contexto de procura *("r1", "r2")* que é guardado na variável *$s*. A função *mf:includesJudgeRel* verifica, então, que não existe o operador *judgeRel* (linha 14) e a execução continua na linha 15. Aqui, a função *mf:matchExpr* retorna a frase *"Aplicação XML"* a procurar guardada na variável *$m* (não há *match options*). Esta frase, juntamente com o contexto de procura (guardado em *$m*), é usada pela função *mf:scoreFTContains* para calcular a relevância final do primeiro operando do "and".

Em relação ao segundo operando, a linha 8 também é executada, conduzindo agora ao contexto de procura *("s1", "s2")*. Tendo este operando um operador *judgeRel*, as linhas 11 a 13 são executadas. O utilizador julgou relevante a secção *s1*. Este identificador é dado pela função *mf:judgeRelIds* (linha 11). O nó correspondente é, então, dado pela tabela *IdNodeTab* (linha 12). A relevância resultante é, finalmente, dada pela função *mf:scoreJudgeRel*, a partir do contexto de procura *("s1", "s2")* e da lista *("s1")* de secções julgadas relevantes dentro desse contexto (linha 13).

5 Processamento incremental

O ambiente de edição do XQuery estendido com operações de selecção deve permitir ao utilizador aceder aos resultados intermédios das perguntas. Além disso, o processador deve ser incremental para que, cada vez que uma operação é inserida ou alterada, apenas os resultados dependentes das alterações feitas são calculados. Um caso particular é o cálculo incremental da função *fts:score* (definida na secção 4.2) porque inclui muitas operações. Suponha-se, por exemplo, que o utilizador introduz a pergunta seguinte:

```
for $a score $s in
        doc("...")/artigos/artigo[título ftcontains "XML"]
```

O resultado da cláusula *score* é dado pela função *fts:score*. Como ainda não é incluído o operador *judgeRel*, a condição *else* da linha 14 é executada. Para um correcto acesso aos resultados intermédios, a lista resultante de títulos guardada na variável *$s* (linha 8) deve ser apresentada ao utilizador, juntamente com as relevâncias respectivas calculadas na linha 16 pela função *mf:scoreFTContains*. Face a esta lista, o utilizador julga relevantes os títulos *"t1"* e *"t4"*, ficando a pergunta da seguinte forma:

for $a score $s in
doc("...")/artigos/artigo[título ftcontains "XML" judgeRel ("t1", "t4")]

A função *fts:score* é executada de novo, passando agora pelas linhas 11 a 13 porque existe o operador *judgeRel*. O processamento incremental deve assegurar que todos os cálculos feitos antes destas linhas não serão efectuados de novo.

A criação de um processador incremental para XQuery estendido pode ser feita recorrendo a um gerador como o LRC [9]. O LRC é um gerador de ambientes incrementais baseados numa definição formal atributiva da linguagem a processar. Este gerador foi usado na criação do sistema de processamento da linguagem IXDIRQL [7].

6 Conclusão e perspectivas

Este artigo define uma extensão ao XQuery com operações de selecção para a contrução interactiva/iterativa das perguntas. Isto ajuda o utilizador, não só a escolher as operações que conduzem ao resultado desejado, mas também a restringir cada resultado intermédio ao subconjunto de nós que interessam o utilizador. A definição formal das operações de selecção aqui proposta pode ser usada para a construção de um sistema de edição e processamento incremental para o XQuery assim estendido. Como trabalho futuro, pretende-se criar um protótipo de um tal sistema usando o LRC. Uma vez criado, este protótipo será alargado de forma a incluir a informação das ontologias associadas à colecção de documentos, como abordado de seguida.

Uso do XQuery na Web semântica

Enquanto as linguagens de interrogação são desenvolvidas, o acesso à informação na Web continua um problema dada a quantidade enorme de recursos disponíveis (XML, HTML e outros). Numa tentativa de atenuar esse problema, os recursos da Web estão a ser associados a descritores semânticos (meta informação) constituindo a chamada Web semântica [10]. A semântica é expressa por ontologias descritas por grafos. Uma ontologia é a representação formal do conhecimento sobre um certo domínio, em que os conceitos são associados entre si por relações binárias (hierárquicas, de sinonímia e outras). Os recursos da Web são ligados às ontologias, constituindo ocorrências de certos conceitos. As ontologias são expressas através de sistemas formais como RDF [11] ou Topic Maps [12].

Pretende-se utilizar o XQuery estendido com a selecção para a interrogação de recursos XML da Web. Para isso, o sistema de processamento deve ser enriquecido de forma a tirar partido da informação dada pelas ontologias. Assim, aquando de uma consulta, a resposta será dada em função dos documentos e das ontologias associadas. Uma possibilidade para o efeito é usar as ontologias no cálculo das relevâncias das operações de similaridade textual: a relevância de um elemento (ou atributo) é calculada em função do conteúdo desse elemento e da descrição semântica fornecida pela ontologia associada.

As ontologias associadas aos elementos podem também ser mostradas juntamente com os resultados intermédios das perguntas. Assim, o utilizador conhecerá melhor o assunto de cada elemento.

Agradecimento: Os autores agradecem à Fundação para a Ciência e a Tecnologia (FCT) o apoio financeiro concedido através da bolsa de pós-doutoramento que suporta o trabalho da Alda Lopes Gançarski.

References

1. S. Amer-Yahia, C. Botev, S. Buxton, P. Case, J. Doerre, D. McBeath, M. Rys, and J. Shanmugasundaram. XQuery 1.0 and XPath 2.0 Full-Text Working Draft. http://www.w3.org/TR/2004/WD-xquery-full-text-20040709/, September 2005.
2. R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley, 1999.
3. A. Berglund, S. Boag, D. Chamberlin, M. Fernandez, M. Kay, J. Robie, and J. Siméon. XML Path Language (XPath) 2.0 W3C Working Draft. http//www.w3c.org/xpath20/, September 2005.
4. S. Boag, D. Chamberlin, M. Fernandez, D. Florescu, J. Robie, and J. Siméon. XQuery 1.0: An XML Query Language. W3C Working Draft. http://www.w3.org/TR/xquery/, September 2005.
5. N. Fuhr, M. Lalmas, S. Malik, and Z. Szlávik, editors. *INEX: Initiative for the Evaluation of XML Retrieval Workshop Proceedings*. DELOS Network of Excellence in Digital Libraries, Schloss Dagstuhl, Germany, December 2004.
6. A. Gançarski and P. Henriques. IXDIRQL: an Interactive XML Data and Information Retrieval Query Language. In *Proceedings of the 7th ICCC/IFIP International Conference on Electronic Publishing*, Guimarães, Portugal, June 2003.
7. A. Gançarski and P. Henriques. Contrução e utilização de um protótipo para o processamento da linguagem de interrogação IXDIRQL. 2005.
8. A. Gançarski and P. Henriques. Extending XQuery with selection operations to allow for interactive construction of queries. In *Proceedings of the 9th ICCC International Conference on Electronic Publishing*, Leuven, Belgium, June 2005.
9. M. Kuiper and J. Saraiva. LRC: A Generator for Incremental Language-Oriented Tools. In K. Koskimies, editor, *7th International Conference on Compiler Construction*, volume 1383, pages 298—301. Lecture Notes in Computer Science, April 1998.
10. E. Miller and S. R. Semantic Web Activity: Advanced Developement. http://www.w3.org/2000/01/sw/, 2003.
11. E. Miller, S. R., and D. Brickley. Resource Description Framework (RDF). http://www.w3.org/RDF, 2005.
12. S. Pepper and G. Moore. XML Topic Maps (XTM) 1.0. http://www.topicmaps.org/xtm/index.html, 2001.