

Universidade do Minho
Conselho de Cursos de Engenharia
Licenciatura em Engenharia Informática

Disciplina de Laboratórios de Informática IV

Ano Lectivo de 2007/08



Universidade do Minho



escola de engenharia



departamento de
informática

Query-By-Example

Miguel dos Santos Esteves (43165)
Luís Pedro Martins dos Santos Machado (43178)
Nuno Miguel Mendonça Coutinho Correia (43179)
Tiago Mendonça Coutinho Correia (47108)

Supervisão:

Pedro Rangel Henriques
Flávio Miguel Xavier Ferreira
Alda Lopes Gancarski

Julho, 2008

Data de Recepção	
Responsável	
Avaliação	
Observações	

Query-By-Example

Miguel dos Santos Esteves (43165)
Luís Pedro Martins dos Santos Machado (43178)
Nuno Miguel Mendonça Coutinho Correia (43179)
Tiago Mendonça Coutinho Correia (47108)

Supervisão:

Pedro Rangel Henriques
Flávio Miguel Xavier Ferreira
Alda Lopes Gancarski

Julho, 2008

Resumo

Elaborou-se neste projecto, uma aplicação que permite pesquisar e analisar informação catalogada em formato **XML**. Neste contexto, é possível visualizar ficheiros em formato **XML** e schemas correspondentes. A representação visual do ficheiro **XML** vai permitir ao utilizador elaborar e executar queries de procura de informação usando implicitamente a linguagem **XQuery**. Tal dinamismo é conseguido graças a uma interface interactiva simples e ergonómica que não exige ao utilizador conhecer a própria linguagem **XQuery**. Para criar esta solução foi usado um ambiente web em **JSP**, uma base de dados para armazenar os ficheiros e um servidor web/apache para exportar a página onde se suportará a interface global do programa.

Área de Aplicação: Processamento de elementos estruturados, Integração.

Palavras-Chave: **XML**, **XSD**, **Schema**, **XPath**, **XQuery**, **JSP**, **Java**, **Query-By-Example**

Conteúdo

Conteúdo	i
Lista de Figuras	iii
1 Introdução	1
2 Enunciado do Problema	4
2.1 Problema	4
2.2 Definição de Query-By-Example	6
2.3 Solução pretendida	6
3 Arquitectura do Sistema	8
3.1 Módulos	8
3.1.1 Inicializador	8
3.1.2 Interface Inicial	9
3.1.3 Produtor XQuery	10
3.1.4 Interface Final	10
3.2 Interação	11
4 Implementação	12
4.1 Tecnologias	12
4.1.1 XML	12
4.1.2 XSD	13
4.1.3 XQuery	15
4.1.4 XPath	15
4.1.5 JSP	16
4.1.6 GlassFish	17
4.1.7 Apache Derby	17
4.1.8 JavaScript	17
4.1.9 Applet	18
4.1.10 CSS	18

4.1.11	JavaBeans	19
4.1.12	Prefuse	19
4.1.13	SAXON	19
4.1.14	HTML	20
4.1.15	DOM	20
4.2	Explicação detalhada dos diferentes módulos	20
4.2.1	Colecção de ficheiros XML e XSD	20
4.2.2	Inicializador	21
4.2.3	Interface Inicial	22
4.2.4	Produtor XQuery	23
4.2.5	Interface Final	25
5	Conclusão	33
	Bibliografia	34
A	Glossário	36
B	Listagem de código	39
B.1	Criação da XQuery	39

Lista de Figuras

1.1	Exemplo de um documento XML	2
2.1	Exemplo de uma colecção	5
2.2	Exemplo de XQuery 1	5
2.3	Exemplo de XQuery 2	5
3.1	Diagrama de blocos	9
3.2	Diagrama de sequencia do sistema	11
4.1	Exemplo de um documento XML	13
4.2	Exemplo de um documento XSD [wikb]	14
4.3	Diagrama da leitura da Base de Dados	21
4.4	Diagrama do módulo de Front-end	22
4.5	Pagina inicial	24
4.6	Árvore da estrutura do XML gerada pelo Prefuse	25
4.7	Grafo da estrutura do XML gerada pelo Prefuse	26
4.8	Visualização de um documento XML	27
4.9	Seleccção de um documento XML	28
4.10	XQuery gerada	29
4.11	Resultado da XQuery do tipo misto	30
4.12	Resultado da XQuery do tipo literal	31
4.13	Resultado da XQuery do tipo elementos	32

Capítulo 1

Introdução

Este relatório tem como principal objectivo explicar a resolução de um projecto sobre [Query-By-Example](#) (pergunta “por exemplos”). A estrutura deste capítulo envolve inicialmente uma breve contextualização que faz um reparo à tecnologia usada e respectivo *background*, seguido de uma apresentação do caso de estudo onde ocorrerá uma alusão à condição e condução da proposta, precedendo a sua motivação e objectivos, por fim, o relevo da estruturação das secções seguintes do relatório.

Inicialmente são brevemente explicadas as tecnologias base usadas para a resolução deste projecto, nomeadamente, [XML](#), [XSD](#), [XPath](#), [XQuery](#).

A linguagem [XML](#) é muito utilizada para catalogar informação dos mais diversos tipos. É um subtipo de [SGML](#) capaz de descrever diversos tipos de dados. O seu propósito principal é a facilidade de partilhar informações através da Internet.

A filosofia do [XML](#) contém vários princípios importantes:

- Separação do conteúdo da formatação
- Simplicidade e Legibilidade, tanto para humanos quanto para computadores
- Possibilidade de criação de tags sem limitação
- Criação de arquivos para validação de estrutura (Chamados [DTDs](#))
- Interligação entre bases de dados distintas
- Concentração na estrutura da informação, e não na sua aparência

O [XML](#) é considerado um bom formato para a criação de documentos com dados organizados de forma hierárquica, como se vê frequentemente em documentos de texto formatados, imagens vectoriais ou bases de dados [[wikc](#)].

Podemos observar na figura 1.1 um exemplo de um documento XML.

É portanto instanciável pois qualquer utilizador pode definir os seus elementos. Para regularizar esta linguagem, foram criadas Schemas de forma a normalizar ou contextualizar o XML. XML Schema é uma linguagem baseada no formato XML para definição de regras de validação (“esquemas”) em documentos no formato XML. Foi a primeira linguagem de esquema para XML a obter o status de “recomendação” por parte do W3C. Esta linguagem é uma alternativa ao DTD, cuja sintaxe não é baseada no formato XML.

Um arquivo contendo as definições na linguagem XML Schema é chamado de XSD (XML Schema Definition) [wikd].

Foi aqui introduzido o conceito de família para facilitar a compreensão do que é um Schema.

Na figura 2.1 pode-se visualizar o Schema (ou família) BOOKS.XSD que define uma série de regras que vão ser respeitadas por todos os documentos XML associados.

Para seleccionar nodos num documento XML é usado o XPath. Assim sendo o XPath é uma linguagem para encontrar informação num documento XML. O XPath é usado para percorrer os elementos e atributos do mesmo [w3s].

```
<book>
  <livro>
    <artigo titulo="Aplicacoes XML"> XML </artigo>
  </livro>
</book>
```

Figura 1.1: Exemplo de um documento XML

Segue-se um exemplo muito simples para se ter uma ideia do que é o XPath:

- /artigo[título=“Aplicações XML”]

que acede à lista de artigos de uma colecção XML cujo título é “Aplicações XML”. No entanto, a formulação das perguntas implica o conhecimento da sintaxe do XPath e a certeza, à priori, dos elementos desejados.

Para facilitar a formulação das perguntas, muitos sistemas de processamento de XPath, como por exemplo o Microsoft Query by Example, permitem a “interrogação através de exemplos” (“query by example”), que consiste em:

- mostrar ao utilizador um documento exemplo e
- permitir ao utilizador fazer a escolha interactiva dos elementos.

Para interrogar uma colecção de documentos XML, foi desenhada a linguagem XQuery. Linguagem essa que usa as expressões XPath para construir a pergunta ou query.

Este projecto tem como objectivo o desenvolvimento de um programa que facilite a formulação de perguntas a uma família de documentos XML com base num documento já

conhecido.

Para facilitar a formulação da pergunta optou-se por desenvolver um sistema de selecção no próprio documento [XML](#). O projecto vai ser explicado detalhadamente nos capítulos que se seguem. No capítulo 2 começa-se por expor o problema e a sua motivação. No capítulo 3 é apresentada uma arquitectura do sistema que serve de modelo para a implementação do mesmo. No capítulo 4 é aprofundado o capítulo 3, no sentido de explicar o uso e finalidade das tecnologias. Por último, no capítulo 5 é explicado o que foi feito ao longo do projecto, descrevendo os objectivos. São dadas algumas ideias de possíveis melhoramentos e conclui-se com uma opinião pessoal.

Capítulo 2

Enunciado do Problema

Neste projecto pretende-se desenvolver um sistema de Recuperação de Documentos de arquivos [XML](#) (Information Retrieval system) que permita visualizar, num navegador web comum, um documento [XML](#) retirado de uma colecção ou família de modo a que cada nodo desse documento possa ser seleccionado como um tipo de dados desejado.

Segue-se uma análise e descrição do problema seguida da introdução do conceito principal em estudo [Query-By-Example](#). Por fim mostra-se o que se pretende como solução para este projecto.

2.1 Problema

Respeitante à figura [2.1](#), na primeira coluna são mostradas as famílias de documentos [XML](#), na segunda a respectiva árvore de documentos [XML](#) e na terceira o conteúdo de cada documento [XML](#). Considere-se neste exemplo uma videoteca como uma compilação de arquivos culturais nomeadamente livros e filmes. A família de documentos [XML](#) *BOOKS.XSD* é uma espécie de prateleira onde o funcionário arruma livros. Essa prateleira tem uma forma específica onde só encaixam livros que possuam a mesma árvore [XML](#). Essa árvore [XML](#) exige que o elemento na raiz de cada livro seja *BOOK* e contenha pelo menos os atributos *title*, *author* e *publisher* em que os respectivos valores têm que ser do tipo *string*. Além disso o livro só pertence à família de *BOOKS.XSD* se contiver uma lista de páginas ou por outras palavras o elemento *PAGES*.

Se o funcionário quiser pesquisar e listar todos os livros do autor “J. R. R. Tolkien” ele precisa de escrever e executar [XQuery](#) da figura [2.2](#).

No exemplo [2.3](#) o funcionário quer ler a página 70 do livro cujo atributo *title* é “The Fellowship of the Rings”. Portanto tem que escrever a [XQuery](#) da figura [2.3](#).

No exemplo [2.3](#), depois de executar esta query, vai obter o texto da página 70 desse livro. O que se pretende aqui é livrar o funcionário de ter que dominar a linguagem XQuery. Em vez disso ele vai utilizar a linguagem [Query-By-Example](#) cuja *learning curve* (curva de aprendizagem) é muito inferior à anterior.

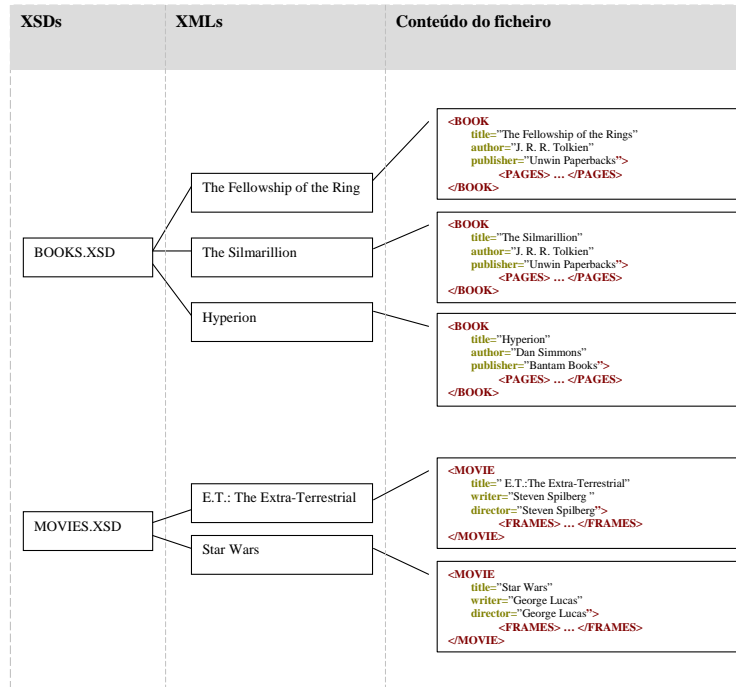


Figura 2.1: Exemplo de uma colecção

```
for $x in doc('The Fellowship of the Ring.xml')
  return $x/book[@author='J. R. R. Tolkien']
```

Figura 2.2: Exemplo de XQuery 1

```
for $x in doc('The Fellowship of the Ring.xml')
  return $x/book[@title='The Fellowship of the Rings']/pages/page[@numero='70']
```

Figura 2.3: Exemplo de XQuery 2

2.2 Definição de Query-By-Example

Segue-se uma definição genérica do [Query-By-Example](#) retirado de [QBE]

Query-By-Example (QBE) is a method of query creation that allows the user to search for documents based on an example in the form of a selected text string or in the form of a document name or a list of documents. Because the QBE system formulates the actual query, QBE is easier to learn than formal query languages, such as the standard Structured Query Language (SQL), while still enabling powerful searches.

To conduct a search for similar documents based on matching text, the user enters or copies selected text into the form search field. This is then passed to the QBE parser for processing. A query is created using the relevant words (common words such as “and,” “is” and “the” are ignored by default) and a search is carried out for documents containing them. Because the meaning of the selected text is less precise than a formal query, results may be more variable than those in a formal query entry.

To conduct a search for similar documents based on full document text, the user submits documents or lists of documents to the QBE results template. The QBE parser performs an analysis of these and formulates a query to submit to the search engine, which in turn conducts a search for similar material.

In terms of database management system, QBE can be thought of as a “fill-in-the blanks” method of query creation. The Microsoft Access Query Design Grid is an example. To conduct a search for field data matching particular conditions, the user enters criteria into the form, creating search conditions for as many fields as desired. A query is automatically generated to search the database for matching data.

Concluindo esta definição, apesar de limitada na forma de elaborar queries, a linguagem [Query-By-Example](#) é interactiva, rápida e fácil de utilizar. Além disso existem casos onde não é necessário grande criatividade nas queries que se querem construir e portanto não há necessidade de investir em linguagens mais baixo nível. O exemplo é a chave que abre mais facilmente as portas para a aprendizagem e está presente nesta linguagem que faz transparecer algum empirismo.

2.3 Solução pretendida

Já foi mostrado na secção 2.1 o problema de ter que dominar uma linguagem de programação como a [XQuery](#). Foi apresentada a linguagem [Query-By-Example](#) na secção 2.2. Pretende agora, no exemplo do capítulo 2.1, que o funcionário formule uma query, interrogando um documento [XML](#) visualizado em modo texto, usando a linguagem [Query-By-Example](#) e internamente o programa reconheça o que lhe está a ser pedido e devolva a resposta que um processador de [XQuery](#) vai resolver. Assim é preciso aqui elaborar todo um mecanismo de selecção de nodos num documento [XML](#) que vai ser explicado com

detalhe no capítulo 4. Como será mais adiante falado no mesmo capítulo, a linguagem XML contém vários tipos de nodos: elementos, atributos, valores destes e texto. É preciso que todos estes nodos sejam seleccionáveis e que cada selecção/combinacção de selecções seja interpretada com diferentes personalidades. Assim, no exemplo dado na secção 2.1, de acordo com a figura 2.1, para o programa interpretar a query 2.2, acordando com a linguagem Query-By-Example, (relembrar que se quer seleccionar todos os livros cujo autor seja J. R. R. Tolkien) o funcionário vai ter que seleccionar o nodo do valor do atributo *author* que é '*J. R. R. Tolkien*'. O resultado desta produção será o próprio elemento *BOOK*, incluindo a árvore de nodos a partir dele, do ficheiro que está a ser interrogado e ainda o elemento *BOOK* do ficheiro *The Silmarillion*. Isto porque ambos estes dois elementos possuem um atributo *author* com o valor que queríamos pesquisar na XQuery. O que no primeiro caso se teve que escrever em código XQuery e especificar para todos os documentos daquela mesma família a mesma interrogação, com Query-By-Example basta seleccionar o nodo correspondente ao valor do atributo *author*. Portanto, o programa deve construir queries de modo a englobar todos os documentos da família do documento que está a ser interrogado.

Capítulo 3

Arquitectura do Sistema

Neste capítulo é descrita a arquitectura do sistema. Por outras palavras os módulos que se pretendem implementar na secção 4 e que constroem o programa serão descritos na secção 3.1. A sequência de acções e interacções com o utilizador que se pretendem implementar será falada na secção 3.2.

3.1 Módulos

Considere-se a figura 3.1 que constitui um diagrama modular geral da maneira de funcionamento do programa. Neste existe uma sequência de acções intra-modular descrita pelas setas rectas e interactiva pelas setas curvas.

De seguida vão ser explicados os módulos e o motivo da sua concepção.

3.1.1 Inicializador

Assumiui-se aqui a necessidade de criar um repositório onde estão guardados os ficheiros XML e XSD. Esse repositório está caracterizado na figura 3.1 como um cilindro (pilha de dados) acessível pelo módulo **Inicializador**. Esse repositório será construído de forma relacional pois vai ser preciso relacionar o conteúdo do ficheiro com o nome do mesmo e ainda os documentos XML com as respectivas famílias. Isto será explicado detalhadamente na secção 4.

O módulo **Inicializador** deve implementar mecanismos de leitura da lista dos nomes dos ficheiros XSD a partir do repositório e enviar para o módulo **Interface Inicial**. Uma vez escolhida a família, este módulo vai responder ao **Interface Inicial** a lista de documentos XML correspondentes. Este módulo deve portanto ser capaz de ler também os nomes dos ficheiros XML e enviar para o **Interface Inicial**. Quando esse pede o conteúdo de um ficheiro XML, o módulo **Inicializador** deve ser capaz de ler o conteúdo do ficheiro XML do repositório e devolvê-lo ao **Interface Inicial**.

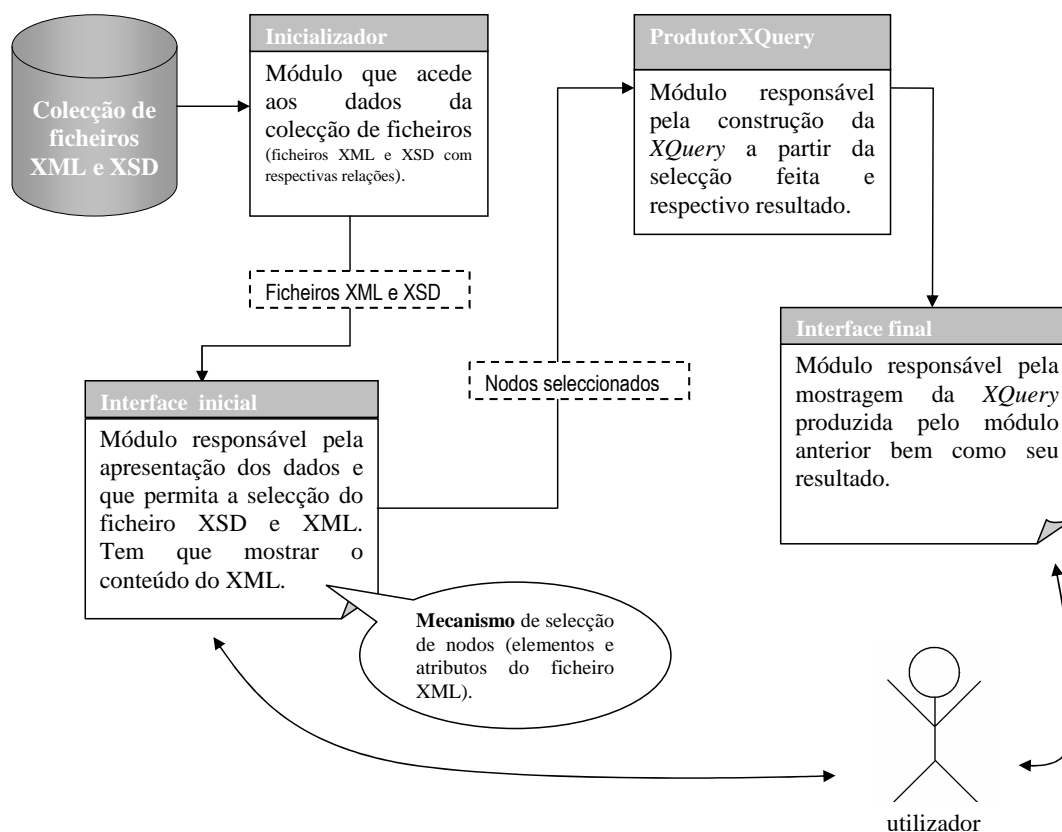


Figura 3.1: Diagrama de blocos

3.1.2 Interface Inicial

Este é o módulo que trata da apresentação da listagem de ficheiros [XML](#), [XSD](#) e do conteúdo do ficheiro XML que deve ser interrogado ainda neste módulo pelo utilizador através de [Query-By-Example](#).

O **Interface Inicial** deve saber pedir informação ao módulo **Inicializador** sempre que o utilizador efectua um pedido. Assim, quando a interface é iniciada, o módulo **Interface Inicial** pede a listagem do nome das famílias ao módulo **Inicializador** e mostra-as ao utilizador. Este escolhe uma família e então é feito um novo pedido ao módulo **Inicializador** que devolve a listagem de todos os documentos [XML](#). O utilizador escolhe o documento que quer interrogar e o **Interface Inicial** pede o conteúdo do ficheiro [XML](#) ao **Inicializador**. Deve ser possível então mostrar a árvore da estrutura do documento [XML](#) para ter uma ideia da query que se quer construir. Depois de ter adquirido o ficheiro, terá de implementar um mecanismo que permita percorrer o conteúdo do [XML](#) como uma árvore de nodo em nodo. A cada nodo é preciso atribuir um carácter seleccionável e um contexto diferente para cada tipo de nodo (possivelmente alguma informação sobre o [XPath](#) de cada nodo deve ficar guardada para posterior acesso no módulo **Produtor XQuery**). Assim se se seleccionar um elemento não é esperado que resulte o mesmo que se se seleccionar um atributo ou um valor de um atributo ou mesmo um nodo de texto.

Cada selecção será submetida para o módulo **Interface Inicial**.

Por fim deverá ser possível escolher o tipo de query que se quer ver no resultado. A opção *mista* englobará todos os nodos como resultado, a opção *literal* deve mostrar apenas os nodos de texto do resultado e a opção *elementos* deve omitir os nodos de texto e mostrar os atributos e elementos do resultado. Esta opção vai ser submetida juntamente com as selecções dos nodos para o módulo **Produtor XQuery** quando o utilizador desejar concluir o processo de construção da query.

3.1.3 Produtor XQuery

É neste módulo que o programa constrói a **XQuery** com base nos nodos seleccionados. Deve receber possivelmente uma série de identidades para todos os nodos do módulo **Interface Inicial** posteriormente seleccionados e submetidos pelo utilizador. Deve ser possível determinar o carácter de cada nodo, i. e. , se se trata de um elemento, atributo, texto ou valor da linguagem **XML**. Isto deve ser resolvido de forma a que o **XPath** de cada nodo esteja relacionado com a sua identidade. Então pretende-se que a query seja relativa a uma série de ficheiros **XML** referentes à mesma família do documento que estava a ser interrogado. Portanto é necessário que também essa referência seja passada do módulo **Interface Inicial** para aqui. A partir de cada nodo seleccionado deve ser construída uma query de pesquisa. Assim se se tratar de um nodo elemento, o resultado deve abranger a procura de todos os elementos de todos os documentos daquela família que sejam iguais ao elemento seleccionado. Esta procura deve encontrar não só o tal elemento mas a árvore de nodos que se situam abaixo deste. De acordo com a opção submetida no módulo **Interface Inicial** vai ser filtrado um tipo diferente de nodos como já foi explicado. Se se tratar de uma selecção do nodo atributo **XML** então vão ser procurados todos os elementos de todos os documentos daquela família que contenham o atributo seleccionado. Se se seleccionar o valor de um atributo a query vai procurar pelos elementos com o atributo cujo valor seja igual ao valor seleccionado. Se se seleccionar um nodo texto, a query deve procurar por todos os elementos que contenham um elemento de modo que o texto dentro desse seja igual ao seleccionado e que o próprio elemento que o contém seja também do mesmo tipo do que contém o texto seleccionado. No capítulo 4 será explicado com pormenor o resultado deste algoritmo com exemplos. Depois de calculada a query, este módulo deve utilizar um processador de **XQuery** para executar a mesma e devolver tanto o resultado como a própria query ao módulo **Interface Final**.

3.1.4 Interface Final

Aqui será apresentado o nome dos ficheiros escolhidos pelo utilizador no módulo **Interface Inicial**, a query gerada no módulo **Produtor XQuery** a partir dos nodos seleccionados no **Interface Inicial** e o respectivo resultado também calculado no módulo **Produtor XQuery** diferenciando o resultado por todos os documentos dessa família.

3.2 Interacção

Perante a arquitectura concebida, chegou-se à conclusão que o sistema deveria respeitar uma lógica igual ou equivalente à que é apresentada na figura 3.2.

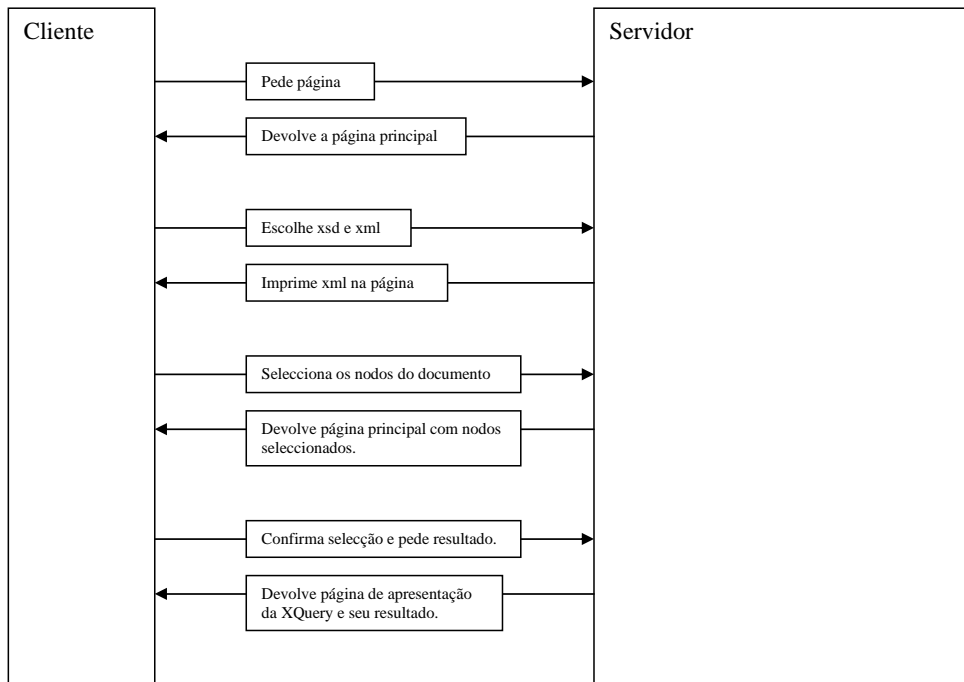


Figura 3.2: Diagrama de sequência do sistema

Na figura 3.2 é mostrada a sequência de interações entre o programa e o utilizador. Utilizou-se o conceito cliente/servidor pois vai-se implementar uma interface web.

Inicialmente o cliente pede a página principal que é concebida no módulo **Interface Inicial** e devolvida pelo mesmo ao cliente. A informação relevante e não estática presente na página é agora apenas a listagem do nome das famílias. O cliente escolhe uma delas e a página devolve a lista de nomes dos ficheiros XML correspondentes àquela família. O cliente escolhe o ficheiro XML e o servidor devolve ao cliente imprimindo na página o conteúdo do documento XML. Assumindo que esse conteúdo contém nodos interactivos, o cliente vai então fazer a selecção dos mesmos. No fim, o cliente vai confirmar a selecção e pedir o resultado da query que tem vindo a construir. O servidor responde com uma apresentação da query gerada e seu resultado.

Capítulo 4

Implementação

Neste capítulo segue-se uma explicação das diferentes tecnologias utilizadas neste projecto, nomeadamente as fundamentais que fazem parte do problema e as escolhidas na fase da implementação. Depois segue-se uma descrição dos módulos do capítulo 3 no âmbito da implementação.

4.1 Tecnologias

4.1.1 XML

Extensible Markup Language (**XML**) é linguagem de marcação de dados (meta-markup language) que provê um formato para descrever dados estruturados. Isso facilita declarações mais precisas do conteúdo e resultados mais significativos de busca através de múltiplas plataformas. O **XML** também vai permitir o surgimento de uma nova geração de aplicações de manipulação e visualização de dados via Internet.

O **XML** permite a definição de um número infinito de tags, ou seja, estruturas de linguagem de marcação que consistem em breves instruções, tendo uma marca de início e outra de fim. Há uma tendência nos dias actuais para se usar as tags apenas como delimitadores de estilo e/ou conteúdo, tanto em **HTML** quanto em **XML**.

Enquanto que no **HTML**, as tags podem ser usadas para definir a formatação de caracteres e parágrafos, o **XML** provê um sistema para criar tags para dados estruturados.

Um documento **XML** é uma árvore rotulada onde um nó externo consiste de:

- dados de caracteres (uma sequência de texto)
- instruções de processamento (anotações para os processadores), tipicamente no cabeçalho do documento
- um comentário (nunca com semântica acompanhando)
- uma declaração de entidade (simples macros)
- nós DTD (Document Type Declaration)

Um nó interno é um elemento, o qual é rotulado com:

- um nome ou
- um conjunto de atributos, cada qual consistindo de um nome e um valor.

Um elemento XML pode ter dados declarados como sendo preços de venda, taxas de preço, um título de livro, a quantidade de chuva, ou qualquer outro tipo de elemento de dado. Como as tags XML são adoptadas por intranets de organizações, e também via Internet, haverá uma correspondente habilidade em manipular e procurar por dados independentemente das aplicações onde os quais são encontrados. Uma vez que o dado foi encontrado, ele pode ser distribuído pela rede e apresentado em um browser de várias formas possíveis, ou então esse dado pode ser transferido para outras aplicações para processamento futuro e visualização [Jún].

Podemos ver na figura 4.1 um exemplo de um simples ficheiro XML onde é se expressa uma receita de pão.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<receita nome="pão" tempo_de_preparo="5 minutos" tempo_de_cozimento="1 hora">
  <titulo>Pão simples</titulo>
  <ingredientes>
    <ingrediente quantidade="3" unidade="xícaras">Farinha</ingrediente>
    <ingrediente quantidade="7" unidade="gramas">Fermento</ingrediente>
    <ingrediente quantidade="1.5" unidade="xícaras" estado="morna">Água</ingrediente>
    <ingrediente quantidade="1" unidade="colheres de chá">Sal</ingrediente>
  </ingredientes>
  <instrucoes>
    <passo>Misture todos os ingredientes, e dissolva bem.</passo>
    <passo>Cubra com um pano e deixe por uma hora em um local morno.</passo>
    <passo>Misture novamente, coloque numa bandeja e asse num forno.</passo>
  </instrucoes>
</receita>
```

Figura 4.1: Exemplo de um documento XML

Na figura 4.1, é visível a distinção no tipo de nodos que um documento XML pode conter através do syntax highlighting. Assim os elementos estão formatados em texto negrito e preto, os atributos são da cor azul e a vermelho o valor dos mesmos. O restante texto é do tipo nodo de texto XML.

4.1.2 XSD

XML Schema, ou também conhecido por XSD como ja foi referido no capítulo 1, é uma linguagem XML que descreve a estrutura de um documento XML. O propósito de um XML Schema é definir os blocos de construção permitidos num documento XML.

Um XML Schema:

- define elementos que podem aparecer em um documento

- define atributos que podem aparecer em um documento
- define que elementos são elementos filhos
- define a ordem dos elementos filhos
- define o número de elementos filhos
- define se um elemento é vazio ou pode incluir texto
- define tipos de dados para elementos e atributos
- define valores padrão e fixos para elementos e atributos

Uma das grandes vantagens de [XML Schemas](#) é o suporte a tipos de dados:

- É mais fácil descrever conteúdo de documentos permissíveis
- É mais fácil validar os dados
- É mais fácil trabalhar com dados de um banco de dados
- É mais fácil definir restrições aos dados
- É mais fácil definir padrões/formatos de dados
- É mais fácil converter dados entre diferentes tipos

[XML Schema](#) foi originalmente proposto pela Microsoft, mas se tornou uma recomendação oficial do [W3C](#) em Maio de 2001 [[Mai05](#)].

Na figura 4.2 mostra-se um exemplo de um documento [XML Schema](#)

```
<xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="country" type="Country"/>
  <xs:complexType name="Country">
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="population" type="xs:decimal"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Figura 4.2: Exemplo de um documento [XSD](#) [[wikb](#)]

4.1.3 XQuery

O [XQuery](#) 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](#), para aceder a elementos ou atributos dos documentos. 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:

```
for $a in doc("http://...") /artigos/artigo
where $a/autor = "Silva"
order by $a/íttulo
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 [[GH05](#)].

4.1.4 XPath

[XPath](#) é um conjunto de regras de sintaxe para definir partes de um documento [XML](#). O que é o [XPath](#)?

- [XPath](#) é uma sintaxe para definir partes de um documento [XML](#)
- [XPath](#) usa caminhos para definir elementos [XML](#)
- [XPath](#) define uma biblioteca de funções padrão
- [XPath](#) é o elemento principal em [XSLT](#)
- [XPath](#) não é escrito em [XML](#)
- [XPath](#) é um padrão [W3C](#)

[XPath](#) usa expressões de caminho para identificar nós num documento [XML](#). Essas expressões de caminho parecem-se muito com as expressões de um sistema de arquivos de um computador, como por exemplo “w3schools/xpath/default.asp”.

De seguida é mostrado um exemplo de [XPath](#).

Senão vejamos este exemplo de [XML](#):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
  <catalog>
    <cd country="USA">
```

```
<title>Empire Burlesque</title>
<artist>Bob Dylan</artist>
<price>10.90</price>
</cd>
<cd country="UK">
  <title>Hide your heart</title>
  <artist>Bonnie Tyler</artist>
  <price>9.90</price>
</cd>
<cd country="USA">
  <title>Greatest Hits</title>
  <artist>Dolly Parton</artist>
  <price>9.90</price>
</cd>
</catalog>
```

A expressão [XPath](#) abaixo selecciona o elemento *RAIZ catalog*:

- /catalog

A expressão [XPath](#) abaixo selecciona todos os elementos *cd* do elemento *catalog*:

- /catalog/cd

A expressão [XPath](#) abaixo selecciona todos os elementos *price* de todos os elementos *cd* do elemento *catalog*:

- /catalog/cd/price

[XPath](#) define uma biblioteca de funções padrão para trabalhar com strings, números e expressões booleanas.[[xpa](#)]

4.1.5 JSP

JavaServer Pages ([JSP](#)) é uma tecnologia utilizada no desenvolvimento de aplicações para Web, similar às tecnologias Active Server Pages (ASP) da Microsoft ou PHP. Por ser baseada na linguagem de programação Java, tem a vantagem da portabilidade de plataforma, que permite a sua execução em diversos sistemas operacionais, como o Windows da Microsoft e versões do Unix, como o Linux. Esta tecnologia permite ao desenvolver de páginas para Internet produzir aplicações que acessem o banco de dados, manipulem arquivos no formato texto, capturem informações a partir de formulários e capturem informações sobre o visitante e sobre o servidor [[wikb](#)]. Esta tecnologia permite

utilizar código Java embebido e a sua sintaxe funciona à base de tags em semelhança ao XML. Usou-se a tecnologia JSP uma vez que está associada à linguagem Java, à qual estávamos mais familiarizados e, ao mesmo tempo, foi adquirido conhecimento sobre uma linguagem muito usada hoje em dia. JSP é, por exemplo, usada na construção da página do nosso programa, como mostra o exemplo seguinte:

```
<jsp:setProperty name="resultadosbean" property="*" />
<jsp:setProperty name="dadosbean" property="result" value="${resultadosbean}" />
```

4.1.6 GlassFish

GlassFish é um servidor do java muito usado hoje em dia por ser muito recente e mesmo em comparação com o servidor Tomcat. Foi usado no projecto para poder por a página a correr sobre ele.

4.1.7 Apache Derby

Apache Derby é uma base de dados mais especificamente um gestor de base de dados relacional baseado na linguagem orientada a objectos Java da Sun e também na linguagem SQL [wikb]. Esta tecnologia é usada na construção da nossa base de dados em termos específicos no seguinte código:

```
String driver = "org.apache.derby.jdbc.ClientDriver";
boolean sucesso = false;
String url = "jdbc:derby://localhost:1527/LI4";
```

4.1.8 JavaScript

JavaScript é uma linguagem de Scripting que é usada por milhões de páginas na Internet e atende principalmente as seguintes necessidades :

- Oferece tipagem dinâmica - tipos de variáveis não são definidos;
- É interpretada, ao invés de compilada;
- Possui óptimas ferramentas padrão para listagens (como as linguagens de script, de modo geral);
- Oferece bom suporte a expressões regulares (característica também comum a linguagens de script).

No projecto temos o seguinte exemplo de desta linguagem:

```
<script type="text/javascript">
    var h= <%=htmlPage.getXmlTags().split(",").length%>;
    document.myApplet.width = 1000;
    document.myApplet.height = 200 + h*15;
</script>
```

Esta *script* é responsável pelo redimensionamento dos limites de espaço que a [Applet](#) deste projecto ocupa.

4.1.9 Applet

Applet é um software aplicativo que é executado no contexto de outro programa. O termo foi introduzido pelo AppleScript em 1993.

Os Applets geralmente tem algum tipo de interface de usuário, ou fazem parte de uma parte de uma destas dentro de uma página da web. Isso os distingue de programas escritos em uma linguagem de programação de scripting (como JavaScript) que também roda em um contexto de um programa cliente maior, mas não podem ser considerados applets.

Applets geralmente tem a capacidade de interagir com e/ou influenciar seu programa hospedeiro, através de privilégios de segurança restritos, apesar de geralmente não serem requeridos a fazê-lo.

Diferentemente de um programa, um applet não pode rodar independentemente; um applet geralmente exibe uma parte gráfica e por vezes interagem com o usuário. Entretanto, eles geralmente são stateless e tem privilégios de segurança restritos. o applet deve rodar em um container, que é provido por um programa hospedeiro, através de um plugin, ou uma variedade de outros aplicativos, incluindo aparelhos móveis que suportam o modelo de programação de applet [wikb].

Segue-se um exemplo do código que embebe uma applet numa página [HTML](#).

```
<APPLET name=myApplet code="{param.list4 == "1" ?
    "applet5/Sample.class" : "Applet6/Applet6.class" }"
    archive="Applet6/prefuse.jar" width=350 height=200>
<PARAM NAME="XmlTags" VALUE="{htmlPage.xmlTags}">
</APPLET>
```

Este código embebe a [Applet](#) na página inicial deste projecto que devolve a árvore ou grafo da estrutura [XML](#) em modo gráfico usando o [Prefuse](#). Esta recebe um parâmetro que é uma frase numa linguagem gerada numa função da classe *HTMLBean*.

4.1.10 CSS

Cascading Style Sheets, ou simplesmente [CSS](#), é uma linguagem de estilo utilizada para definir a apresentação de documentos escritos em uma linguagem de marcação, como

[HTML](#) ou [XML](#). O seu principal benefício é prover a separação entre o formato e o conteúdo de um documento. Ao invés de colocar a formatação dentro do documento, o desenvolvedor cria um link (ligação) para uma página que contém os estilos, procedendo de forma idêntica para todas as páginas de um portal. Quando quiser alterar a aparência do portal basta portanto modificar apenas um arquivo [[wikb](#)]. Essa tecnologia é usada no nosso programa por exemplo para marcar a parte do código que foi seleccionado como podemos ver pelo seguinte exemplo:

```
a.attributeValue: hover{color : black;}
a.element: hover{color : black;}
```

4.1.11 JavaBeans

JavaBeans são componentes de software escritos na linguagem de programação [Java](#). Segundo a especificação da Sun Microsystems os JavaBeans são “componentes reutilizáveis de software que podem ser manipulados visualmente com a ajuda de uma ferramenta de desenvolvimento”. [[wikb](#)] Um exemplo da aplicação da JavaBeans é o seguinte :

```
<jsp:useBean scope="session" class="beans.HTMLBean" id="htmlPage"/>
```

Usou-se neste projecto uma [Session Bean](#) para guardar a informação do cliente. Segue-se uma breve ideia deste conceito.

A session bean represents a single client inside the Application Server. To access an application that is deployed on the server, the client invokes the session bean's methods. The session bean performs work for its client, shielding the client from complexity by executing business tasks inside the server.

As its name suggests, a session bean is similar to an interactive session. A session bean is not shared; it can have only one client, in the same way that an interactive session can have only one user. Like an interactive session, a session bean is not persistent. (That is, its data is not saved to a database.) When the client terminates, its session bean appears to terminate and is no longer associated with the client [[Ses](#)].

4.1.12 Prefuse

4.1.13 SAXON

[SAXON](#) é um processador de [XQuery](#) do tipo open-source que existe para versão java e que foi usado para compilar as [XQuery](#) do nosso programa . Um exemplo disso pode ser visto no código seguinte do projecto :

```
XQDataSource ds = new SaxonXQDataSource();
```

4.1.14 HTML

HTML (acrónimo para a expressão inglesa HyperText Markup Language, que significa Linguagem de Marcação de Hipertexto) é uma linguagem de marcação utilizada para produzir páginas na Web. Documentos **HTML** podem ser interpretados por navegadores. A tecnologia é fruto do “casamento” dos padrões HyTime e **SGML**.

Esta tecnologia é muito importante para a apresentação do nosso projecto e o exemplo seguinte mostra isso:

```
<form name="form1" action="index.jsp" method="POST">
<select name="familiaName" onchange="document.form1.submit();">
```

4.1.15 DOM

The Document Object Model is a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents. The document can be further processed and the results of that processing can be incorporated back into the presented page. This is an overview of DOM-related materials here at W3C and around the web [W3C].

O seguinte exemplo presente na classe *HTMLBean* deste projecto lê um documento **XML** e gera uma árvore de nodos, usando a biblioteca **DOM**.

```
public NodeList readXML(String path) throws ParserConfigurationException, SAXException,
    DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
    DocumentBuilder db = dbf.newDocumentBuilder();
    Document doc = db.parse(path);

    return doc.getChildNodes();
}
```

4.2 Explicação detalhada dos diferentes módulos

4.2.1 Colecção de ficheiros XML e XSD

Considere-se a figura 4.3 como uma captura do panorama da implementação do acesso ao repositório dos ficheiros.

Em termos concretos a nossa base de dados é constituída por três tabelas, uma primeira sobre a lista de ficheiros **XML** e os seus respectivos caminhos, a segunda tabela sobre a lista de ficheiros **XSD** e os caminhos correspondentes e finalmente a terceira tabela com a lista de relações entre os ficheiros dos **XSD** e dos **XML**.

Estas tabelas são preenchidas por uma aplicação *stand-alone*, como se pode ver no topo da figura 4.3, que de x em x segundos verifica se o número de ficheiros presentes na directoria

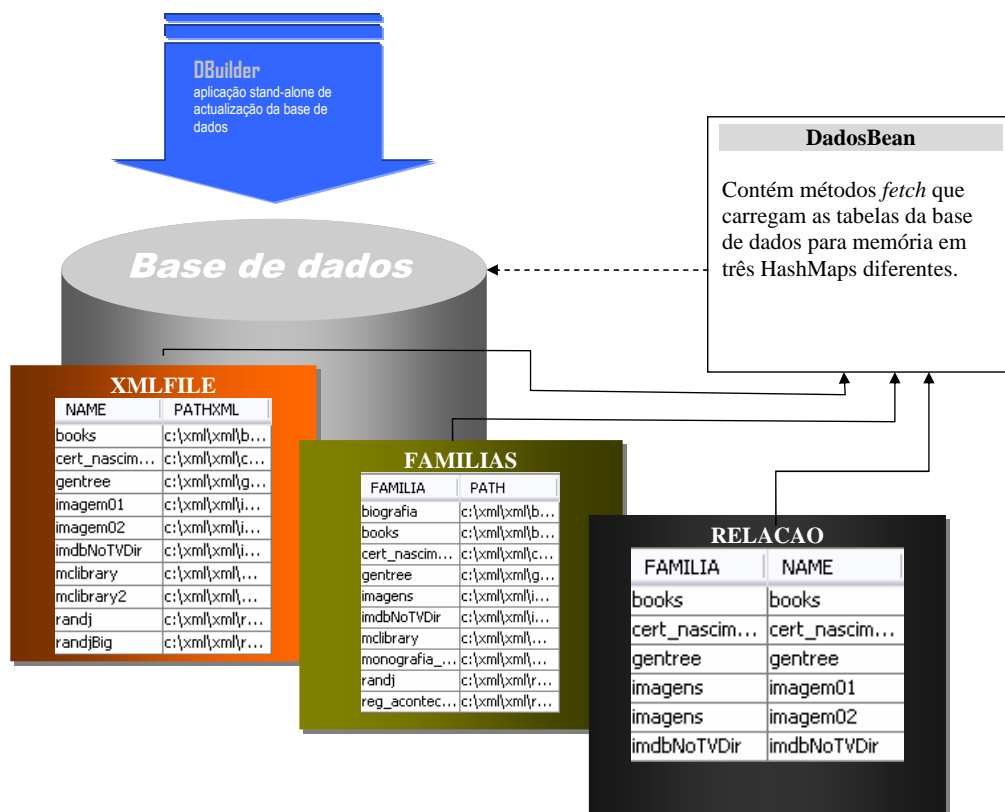


Figura 4.3: Diagrama da leitura da Base de Dados

aqui especificada é igual ao número de registos das tabelas na base de dados. Se não for igual significa que houve alterações, e portanto este programa vai actualizar as tabelas da base de dados. Esta aplicação foi desenvolvida no sentido de não sobrecarregar o servidor web.

4.2.2 Inicializador

Este módulo é principalmente responsável por receber os dados da base de dados e preencher as variáveis instanciadas numa classe *DadosBean* que irá albergar todos os dados base para o funcionamento do programa. Esta classe contém três variáveis de instância do tipo *HashMap*. Estas são preenchidas nesta mesma classe com a informação proveniente dos acessos à base de dados como é mostrado na figura 4.3. Acessos esses que também são efectuados por métodos *fetch* desta mesma classe.

É importante referir que as classes terminam com o nome Bean porque são classes especiais do **Java** que podem depois ser usadas por páginas construídas em **JSP**. A única diferença em relação às classes comuns, é o facto de se ter que declarar os métodos *get* e *set* para poder utilizá-los em conjunto com o **JSP**. O motivo para fazer a leitura da base de dados

usando esta classe é primeiramente pelo facto de o programa ficar mais organizado. Além disso também não é preciso estar sempre a aceder à base de dados sempre que se necessita de informação. A alternativa embora não tão viável seria usar umas tags do **JSP (SQL)** para efectuar a leitura.

Existe ainda neste módulo uma função na classe *HTMLBean* que vai buscar o conteúdo do documento **XML** ao repositório de ficheiros.

4.2.3 Interface Inicial

Considere-se a figura onde *index.jsp* é a página principal.

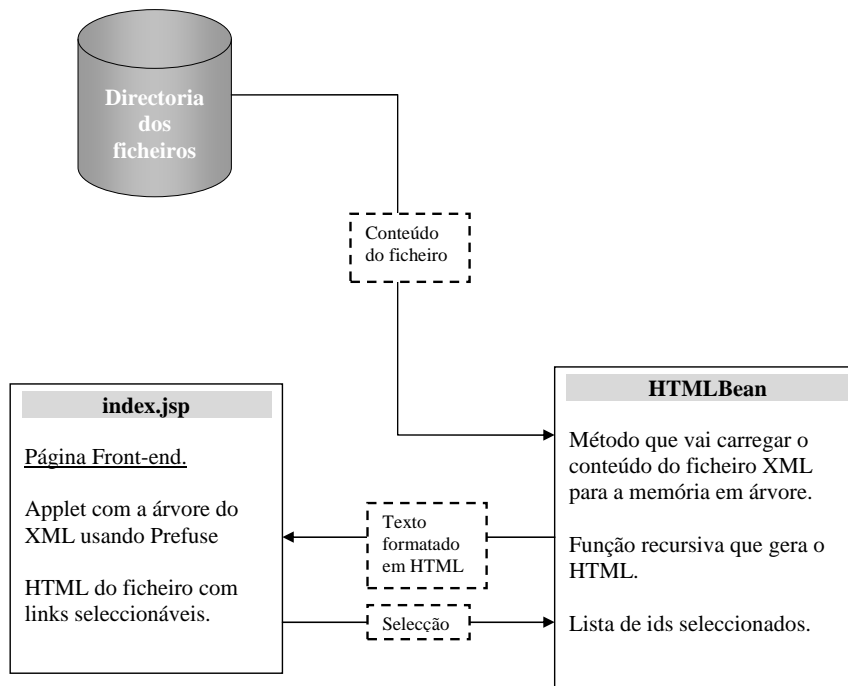


Figura 4.4: Diagrama do módulo de Front-end

Nesta secção temos como objecto de input três “drop-down list”. A primeira irá conter a lista de ficheiros **XSD**, a segunda irá conter a lista de ficheiros **XML**, e a terceira permite a apresentação da estrutura do documento **XML** usando a biblioteca **Prefuse**. A página inicial pode ser observada na figura 4.5.

Depois de escolhidos os documentos da família e correspondente **XML**, serão guardados na classe *ResultadosBean*. Consequentemente o programa irá carregar o ficheiro **XML** seleccionado usando uma função do módulo **Inicializador** presente na classe *HTMLBean*.

Esta classe foi criada com o propósito de permitir apresentar o documento [XML](#), ao utilizador, em modo texto com carácter seleccionável. A classe *HTMLBean* transforma um documento [XML](#) numa árvore de nodos usando a biblioteca [DOM](#).

De seguida, existe um função recursiva que percorre essa árvore e para cada nodo gera um link [HTML](#) com uma referência para a própria página submetendo o identificador desse nodo, como pode ser visto na figura 4.8. Esta função vai preenchendo uma variável de instância do tipo *HashMap<Integer,String>* onde a chave é o identificador de cada nodo da árvore e o valor é o [XPath](#) correspondente. Optou-se, nesta mesma função, por usar um [CSS](#) para distinguir os vários tipos de nodos seleccionáveis (syntax highlighting), e sempre que um link for seleccionado, este muda a sua cor. Além disso é gerada uma indentação na mesma função.

Sempre que é seleccionado um nodo [XML](#) o programa vai adicionando numa *HashMap* da classe *HTMLBean* os identificadores dos links seleccionados, para depois poder gerar a [XQuery](#).

Finalmente, usamos [JavaScript](#) para submeter a posição da página actual dentro dos links dos nodos. Quando a página for recarregada, o [JavaScript](#) vai fazer *scroll* para a posição anterior.

No terceiro “drop-down list” da página é dada a possibilidade de escolher entre visualizar a estrutura do documento [XML](#) no formato árvore 4.6 ou grafo 4.7. Na função recursiva atrás descrita, é ainda gerada uma linguagem que descreve a árvore genérica do documento [XML](#) para que possa ser passada como parâmetro à [Applet](#). Por exemplo *booklist book,book author,book title,book year,author name,author age*. Cada um destes nomes é um elemento ou atributo (se começar por '@'). As relações de pai para filho são expressas com um espaço a separar o pai do filho. Cada declaração é separada por ','. A [Applet](#) vai interpretar esta linguagem e gerar uma árvore ou um grafo conforme a opção seleccionada.

Todo este módulo pode ser acompanhado na figura 4.4.

4.2.4 Produtor XQuery

Nesta aplicação, pode-se seleccionar todos os nodos de um documento [XML](#), tais como, os elementos, os atributos, os valores dos atributos, e o texto dentro dos elementos.

Depois de a selecção estar concluída, o módulo responsável pela construção das queries, cria a respectiva [XQuery](#). Para construir as queries utilizou-se a sintaxe presente no site [W3C](#). A [XQuery](#) usa o seguinte algoritmo:


```
for $x in doc('document.xml') return $x/xpath
```

$$\sum_{i=1}^{t-1} union \$x/xpath$$

sendo t = comprimento da conjunção das [XPath](#)s

Cada [XPath](#) corresponde à path construída a partir da selecção dos ids. É feita a conjunção das [XPath](#)s para minimizar o tamanho da Query.

XSD

XML


  

Figura 4.5: Pagina inicial

No caso de seleccionar um elemento, o [XPath](#) corresponderá ao caminho desde a raiz do documento [XML](#) até ao respectivo elemento seleccionado, como por exemplo, “/livro/autor/elemSeleccionado”. Se um atributo for seleccionado, o [XPath](#) será do tipo “//*[@ATRIB]”, o que corresponde a seleccionar todos os nodos do documento que tenham um atributo “ATRIB”. Por outro lado, a selecção do valor do atributo cria uma [XPath](#) do tipo “/livro/autor/elem3[@ATRIB='val’]”, em que elem3 corresponde ao elemento em que foi seleccionado o atributo “ATRIB”. Esta [XPath](#) resulta na selecção de todos os nodos que tenham o caminho “/livro/autor/elem3” e tenham um atributo “ATRIB” com o valor “val”. Finalmente, a selecção do texto implica a construção de um [XPath](#) que tenha o caminho desde a raiz do documento até ao elemento anterior usando um filtro associado ao texto seleccionado. Por exemplo, “/livro/autor[titulo='Código da Vinci’]”, resulta na selecção de todos os nodos que tenham o caminho “/livro/autor” em que o nodo seguinte seja “titulo” e o seu texto seja igual a “Código da Vinci”. Estas [XPaths](#) serão usadas numa função que constrói a [XQuery](#) correspondente para cada [XML](#) do documento [XSD](#) seleccionado e retorna-as num `ArrayList` de strings. Além disso, gera uma [XQuery](#) que será usada na página de apresentação das queries e dos seus resultados, e que a única diferença é que o nome do documento é o nome geral “document.xml”. Por razões óbvias, esta [XQuery](#) não será executada, uma vez que não usa um documento [XML](#) existente.

É ainda dada a possibilidade ao utilizador de optar por visualizar os resultados das queries

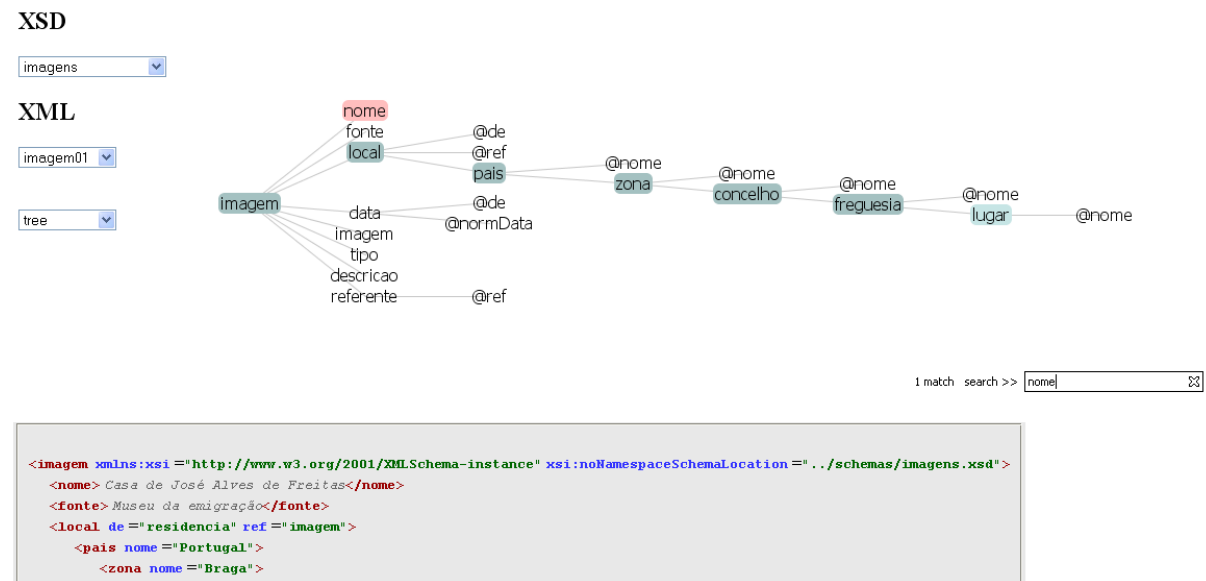


Figura 4.6: Árvore da estrutura do XML gerada pelo Prefuse

apenas composta por elementos, texto ou de forma mista, que engloba os elementos e o texto.

Este módulo está presente na classe *ResultadosQuery*. Recebe uma lista das XQueries previamente construídas na classe *HTMLBean*. De seguida executa-as usando a biblioteca SAXON (processador de XQuery) que devolve um árvore de resultados (XQSequence). Percorrendo esta árvore devolve-se os valores de acordo com a opção pretendida.

Na figura 4.9 é apresentado um exemplo de selecção dos nodos de um documento XML. A XQuery construída está presente na figura 4.10, além de mostrar o resultado da sua execução para todos os tipos de resultados pretendidos (misto 4.11, literal 4.12 e elementos 4.13) e todos os XMLs do respectivo XSD.

Na secção B.1 é possível visualizar o código.

4.2.5 Interface Final

Este módulo é composto pela página *ResultPage.jsp* que recebe a query e o seu resultado e apresenta ao utilizador. Usou-se CSS para *syntax highlighting*, indentação, e tabelas HTML para diferenciar o resultado da query para os diferentes documentos XML da mesma família.

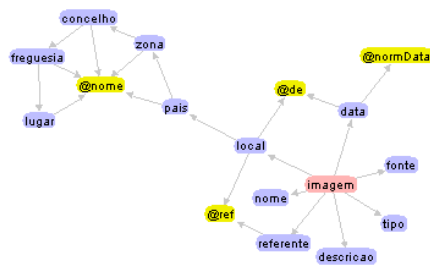
XSD

imagens

XML

imagem01

graph



```
<imagem xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="../schemas/imagens.xsd">
  <nome> Casa de José Alves de Freitas</nome>
  <fonte> Museu da emigração</fonte>
  <local de="residencia" ref="imagem">
    <pais nome="Portugal">
      <zona nome="Braga">
```

Figura 4.7: Grafo da estrutura do XML gerada pelo Prefuse

XSD

imagens ▾

XML

imagem01 ▾

----- ▾

```
<imagem xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation=" ../schemas/imagens.xsd" >
  <nome> Casa de José Alves de Freitas</nome>
  <fonte> Museu da emigração</fonte>
  <local de="residencia" ref="imagem" >
    <pais nome="Portugal" >
      <zona nome="Braga" >
        <concelho nome="Fafe" >
          <freguesia nome="Fafe" >
            <lugar nome="Castro" ></lugar></freguesia></concelho></zona></pais></local>
          <data de="documento" normData="1885-01-01" ></data>
          <imagem> http://www.museu-emigrantes.org/imagens/casas-fafe/jose_alves_freitas.jpg</imagem>
          <tipo> fotografia</tipo>
          <descricao> Casa de José Alves de Freitas</descricao>
          <referente ref="individuo" > José Alves de Freitas</referente>
        </imagem>
```

Misto:
Literal:
Elementos:

Concluir

Figura 4.8: Visualização de um documento XML

XSD

imagens ▾

XML

imagem01 ▾

----- ▾

```
<imagem xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation=" ../schemas/imagens.xsd" >
  <nome> Casa de José Alves de Freitas</nome>
  <fonte> Museu da emigração</fonte>
  <local de="residencia" ref="imagem">
    <pais nome="Portugal">
      <zona nome="Braga">
        <concelho nome="Fafe">
          <freguesia nome="Fafe">
            <lugar nome="Castro"></lugar></freguesia></concelho></zona></pais></local>
          <data de="documento" normData="1885-01-01"></data>
          <imagem> http://www.museu-emigrantes.org/imagens/casas-fafe/jose_alves_freitas.jpg</imagem>
          <tipo> fotografia</tipo>
          <descricao> Casa de José Alves de Freitas</descricao>
          <referente ref="individuo"> José Alves de Freitas</referente>
        </imagem>
```

Misto:
 Literal:
 Elementos:
 Concluir

Figura 4.9: Selecção de um documento XML

XQUERY:

```
for $x in doc('document.xml') return  
    $x//*[@de]  
union  
    $x/imagem/imagem  
union  
    $x/imagem/referente[@ref='indivíduo']  
union  
    $x/imagem[fonte='Museu da emigração']
```

Figura 4.10: XQuery gerada

RESULTADOS

imagem01	<pre> /imagem •<imagem xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="../schemas/imagens.xsd"> <nome>Casa de José Alves de Freitas</nome> <fonte>Museu da emigração</fonte> <local ref="imagem" de="residencia"> <pais nome="Portugal"> <zona nome="Braga"> <concelho nome="Fafe"> <freguesia nome="Fafe"> <lugar nome="Castro"></lugar></freguesia></concelho></zona></pais></local> <data normData="1885-01-01" de="documento"></data> <imagem>http://www.museu-emigrantes.org/imagens/casas-fafe/jose_alves_freitas.jpg</imagem> <tipo>fotografia</tipo> <descricao>Casa de José Alves de Freitas</descricao> <referente ref="individuo">José Alves de Freitas</referente> </imagem> </freguesia> </concelho> </zona> </pais> </local> /imagem/local •<local ref="imagem" de="residencia"> <pais nome="Portugal"> <zona nome="Braga"> <concelho nome="Fafe"> <freguesia nome="Fafe"> <lugar nome="Castro"></lugar></freguesia></concelho></zona></pais> </freguesia> </concelho> </zona> </pais> </local> /imagem/data •<data normData="1885-01-01" de="documento"> </data> /imagem/imagem •<imagem>http://www.museu-emigrantes.org/imagens/casas-fafe/jose_alves_freitas.jpg </imagem> /imagem/referente •<referente ref="individuo">José Alves de Freitas </referente> </pre>
imagem02	<pre> /imagem/imagem •<imagem>http://www.museu-emigrantes.org/imagens/fafe-postais-antigos/av_estação_postal.jpg </imagem> </pre>

Figura 4.11: Resultado da XQuery do tipo misto

RESULTADOS

imagem01	<p><i>/imagem</i></p> <ul style="list-style-type: none"> • Casa de José Alves de Freitas Museu da emigração <p>http://www.museu-emigrantes.org/imagens/casas-fafe/jose_alves_freitas.jpg fotografia Casa de José Alves de Freitas José Alves de Freitas</p> <p><i>/imagem/imagem</i></p> <ul style="list-style-type: none"> •http://www.museu-emigrantes.org/imagens/casas-fafe/jose_alves_freitas.jpg <p><i>/imagem/referente</i></p> <ul style="list-style-type: none"> •José Alves de Freitas
imagem02	<p><i>/imagem/imagem</i></p> <ul style="list-style-type: none"> •http://www.museu-emigrantes.org/imagens/fafe-postais-antigos/av_estação_postal.jpg

Reiniciar

Figura 4.12: Resultado da XQuery do tipo literal

RESULTADOS

imagem01	<pre> /imagem •<imagem xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation=" ../schemas/imagens.xsd"> <nome></nome> <fonte></fonte> <local ref="imagem" de="residencia"> <pais nome="Portugal"> <zona nome="Braga"> <concelho nome="Fafe"> <freguesia nome="Fafe"> <lugar nome="Castro"></lugar></freguesia></concelho></zona></pais></local> <data normData="1885-01-01" de="documento"></data> <imagem></imagem> <tipo></tipo> <descricao></descricao> <referente ref="individuo"></referente> </imagem> /imagem/local •<local ref="imagem" de="residencia"> <pais nome="Portugal"> <zona nome="Braga"> <concelho nome="Fafe"> <freguesia nome="Fafe"> <lugar nome="Castro"></lugar></freguesia></concelho></zona></pais> </local> /imagem/data •<data normData="1885-01-01" de="documento"> </data> /imagem/imagem •<imagem> </imagem> /imagem/referente •<referente ref="individuo"> </referente> </pre>
imagem02	<pre> /imagem/imagem •<imagem> </imagem> </pre>

Reiniciar

Figura 4.13: Resultado da XQuery do tipo elementos

Capítulo 5

Conclusão

Neste trabalho foi feita uma aplicação web que usa a linguagem [Query-By-Example](#). Esta permite ao utilizador construir e executar XQueries de uma forma facilitada e rápida, sem que seja necessário ao utilizador conhecer a linguagem [XQuery](#).

Concluiu-se com sucesso o desenvolvimento de um sistema de Recuperação de Documentos de arquivos [XML](#) (Information Retrieval system) que permite visualizar, num navegador web, um documento [XML](#) exemplo retirado de uma colecção de modo a que cada elemento desse documento possa ser seleccionado. É então construída a pergunta, ou seja, a [XQuery](#) que especifica os documentos onde se pretende obter os resultados da execução dessa [XQuery](#). Para enriquecer a aplicação desenvolveram-se [Applets](#) para visualizar a estrutura do documento [XML](#) seleccionado.

Após a realização deste trabalho, pode-se agora evoluir esta aplicação e torná-la mais interessante a nível de design, além de evoluir o mecanismo de selecção dos nodos no [XML](#) para obter novos tipos de queries mais sofisticadas. Podia-se fazer ainda uma página web (que seria apresentada antes da página inicial deste trabalho) onde permitisse ao utilizador fazer os uploads dos [XMLs](#) e [XSDs](#) de modo a poder fazer as XQueries sobre esses documentos.

Concluindo, com uma breve opinião pessoal, a evolução é sempre uma motivação, ainda mais quando é despertada tanto a nível de desenvolvimento como pós-conclusão do trabalho. O nosso objectivo é aprender a desenvolver aplicações web e interiorizar os conceitos que temos vindo a falar. A nossa motivação no fundo é saber que estamos a desenvolver uma espécie de cluster de um projecto maior que tem sérias hipóteses de ver a luz do dia no mercado de trabalho.

Bibliografia

- [GH05] Alda Lopes Gançarski and Pedro Rangel Henriques.
Extensão do XQuery com operações de selecção para a construção interactiva das perguntas.
Technical report, Departamento de Informática, Universidade do Minho, 2005.
- [Jún] Miguel Benedito Furtado Júnior.
XML.
http://www.gta.ufrj.br/grad/00_1/miguel/link5.htm.
- [Mai05] Maurício M. Maia.
XSD, 2005.
<http://www.dicas-l.com.br/dicas-l/20050326.php>.
- [Pre] prefuse.
<http://prefuse.org/>.
- [QBE] Query-by-Example.
http://searchoracle.techtarget.com/sDefinition/0,,sid41_gci214554,00.html.
- [SAX] saxon.
<http://www.saxonica.com/>.
- [Ses] SessionBean.
<http://java.sun.com/j2ee/1.4/docs/tutorial/doc/EJBConcepts3.html>.
- [W3C] w3c.
<http://www.w3.org/>.
- [w3s] w3schools.
XPath.
<http://www.w3schools.com/XPath/default.asp>.
- [wika] WikipediaEN.
<http://en.wikipedia.org>.
- [wikb] WikipediaPT.
<http://pt.wikipedia.org>.
- [wikc] wikipedia.
XML.
<http://pt.wikipedia.org/wiki/XML>.

- [wikd] wikipedia.
XSD.
http://pt.wikipedia.org/wiki/XML_Schema.
- [xpa] XPath.
http://www.macoratti.net/vb_xpath.htm.

Apêndice A

Glossário

Applet An applet is a software component that runs in the context of another program, for example a web browser. An applet usually performs a very narrow function that has no independent use. Hence, it is an application -let [[wika](#)].

CSS Cascading Style Sheets, ou simplesmente CSS, é uma linguagem de estilo utilizada para definir a apresentação de documentos escritos em uma linguagem de marcação, como HTML ou XML. Seu principal benefício é prover a separação entre o formato e o conteúdo de um documento [[wikb](#)].

DI Departamento de Informática.

DOM Document Object Model (Modelo de Objectos de Documentos) é uma especificação da W3C, independente de plataforma e linguagem, onde pode-se alterar e editar a estrutura de um documento. A API DOM oferece uma maneira padrão de se acessar os elementos de um documento, além de se poder trabalhar com cada um desses elementos separadamente, e por esses motivos criar páginas altamente dinâmicas [[wikb](#)].

DTD Document Type Definition, ou simplesmente DTD, contém as regras que definem quais as tags que podem ser usadas em um documento XML e quais os valores válidos [[wikb](#)].

gEPL grupo de Especificação e Processamento de Linguagens.

GUI *Graphical User Interface* - a designação usada para as interfaces gráficas encontradas na maior parte dos sistemas actuais.

HTML *Hypertext Markup Language* - um conjunto de etiquetas e regras de anotação de texto (de acordo com SGML) usado na criação de documentos hipertexto para serem divulgados na Internet e implementados pelos browsers WWW. É um standard mantido pela W3C.

HTTP *HyperText Transfer Protocol*.

IDE *Integrated Development Environment*.

Java Java é uma linguagem de programação orientada a objeto [wikb].

JavaBeans JavaBeans são componentes de software escritos na linguagem de programação Java. Segundo a especificação da Sun Microsystems os JavaBeans são "componentes reutilizáveis de software que podem se manipulados visualmente com a ajuda de uma ferramenta de desenvolvimento"[wikb].

JavaScript JavaScript is a scripting language most often used for client-side web development. Although best known for its use in websites (as client-side JavaScript), JavaScript is also used to enable scripting access to objects embedded in other applications (see below) [wika].

JSP JavaServer Pages é uma tecnologia utilizada no desenvolvimento de aplicações para Web, similar às tecnologias Active Server Pages (ASP) da Microsoft ou PHP [wikb].

learning curve The term learning curve refers to a relationship between the duration of learning or experience and the resulting progress [wika].

Microsoft Query by Example is a method of creating database queries using examples based on a text string, the name of a document or a list of documents[wika].

Prefuse Prefuse is a set of software tools for creating rich interactive data visualizations. The original prefuse toolkit provides a visualization framework for the Java programming language [Pre].

Query-By-Example Query-By-Example (QBE) é uma linguagem de consulta e também um antigo SGDB que incluía essa linguagem [wikb].

SAXON The Saxon package is a collection of tools for processing XML documents [SAX].

Schema An XML schema is a description of a type of XML document, typically expressed in terms of constraints on the structure and content of documents of that type, above and beyond the basic syntax constraints imposed by XML itself [wika].

Session Bean In the Java Platform, Enterprise Edition specifications, a Session Bean is a type of Enterprise Beans. In the J2EE architecture, the other two types are Entity Beans and Message-driven beans. However, in Java EE 5 entity beans have been replaced by Java Persistence API entities[wikb].

SGML Acrônimo de Standard Generalized Markup Language, ou Linguagem Padronizada de Marcação Genérica[wikb].

SQL Structured Query Language, ou Linguagem de Consulta Estruturada ou SQL, é uma linguagem de pesquisa declarativa para banco de dados relacional (base de dados relacional)[wikb].

XHTML O XHTML, ou eXtensible Hypertext Markup Language, é uma reformulação da linguagem de marcação HTML baseada em XML [wikb].

XML eXtensible Markup Language é uma recomendação da W3C para gerar linguagens de marcação para necessidades especiais [wikb].

XPath XML Path Language is a language for selecting nodes from an XML document [wika].

XQuery é uma linguagem de consulta, com alguns recursos de programação, que é projectada para fazer consultas em colecções de dados em XML. Ela é semanticamente similar ao SQL [wikb].

XSD XML Schema é uma linguagem baseada no formato XML para definição de regras de validação ("esquemas") em documentos no formato XML [wikb].

XSLT é uma linguagem de marcação XML usada para transformar documentos XML. A especificação XSLT - eXtensible Stylesheet Language for Transformation (linguagem de folhas de estilo extensível para transformação) - possibilita transformações mais potentes do que as folhas de estilo CSS.[wikb].

W3C The World Wide Web Consortium (W3C) develops interoperable technologies (specifications, guidelines, software, and tools) to lead the Web to its full potential. W3C is a forum for information, commerce, communication, and collective understanding [W3C] .

Apêndice B

Listagem de código

B.1 Criação da XQuery

Segue-se aqui a função `Java` da construção das XQueries que está presente na classe `HTMLBean`.

```
//Funcao que retorna uma xquery para cada documento XML da familia de XMLs (XSD),  
usando a lista dos ids selecionados  
  
public ArrayList<String> getXQuery(String xsd, ArrayList<Integer> l){  
    try {  
        String str = new String(), s = new String();  
        ArrayList<String> aux = new ArrayList<String>();  
        ArrayList<String> resultado = new ArrayList<String>();  
        ArrayList<String> listaXmIs = new ArrayList<String>();  
  
        //Cria uma conexao com a base de dados e selecciona todos os XMLs do XSD  
        indicado  
  
        Connection conn = DriverManager.getConnection(url, user, pass);  
        Statement st = conn.createStatement();  
        ResultSet r = st.executeQuery("select name from relacao where familia = '"+  
        xsd+"'");  
  
        //Adiciona à lista "listaXmIs" todos os XMLs do XSD  
  
        while(r.next())  
            listaXmIs.add(r.getString(1));  
  
        //Adiciona um XML de nome "document.xml" que vai ser apresentado na XQuery  
        resultante, mas nao sera executada com este documento  
  
        listaXmIs.add("document.xml");  
  
        //Adiciona à lista "resultado" as XQueries para cada documento XML  
  
        for(int k=0; k<listaXmIs.size()-1;k++){  
  
            //Selecciona os caminhos dos documentos XMLs a partir da base de dados  
  
            r = st.executeQuery("select pathxml from xmlfile where name = '"+  
            listaXmIs.get(k)+"'");  
            r.next();  
            if(l.size()!=0){  
  
                //Adiciona à string "s" a sintaxe definida no site W3C
```

```

s="for $x in doc('"+r.getString(1)+"') return ";
for(int i=0; i<l.size();i++){
    str = this.tab.get(l.get(i));

    //Adiciona à lista "aux" a string "str" que contém todos os
xpaths e adiciona à string "s" a sintaxe '$x/xpath' seguido de unions se necessario
    if(!aux.contains(str)){
        if (i>0) s+=" union ";
        s+="$x"+str;
        aux.add(str);
    }
}
}
resultado.add(s);
aux = new ArrayList<String>();
}

//Trata da XQuery de apresentação

s="<query1 class='palavras'>for</query1> $x <query1 class='palavras'>in</
query1> doc(<query1 class='str'>'"+listaXmls.get(listaXmls.size()-1)+"'</query1> <
query1 class='palavras'>return</query1> \n";
for(int j=0; j<l.size();j++){
    str = this.tab.get(l.get(j));
    if(!aux.contains(str)){
        if (j>0) s+="\n<query1 class='uniao'>union</query1>\n";
        s+="&nbsp; &nbsp; &nbsp;";
        s+="$x"+str;
        aux.add(str);
    }
}
resultado.add((new HtmlGenerator()).escapeHTMLResult(s));

conn.close();
return resultado;

} catch (SQLException ex) {
    ex.getMessage();
}
return null;
}

```