UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

# Uma linguagem visual de consulta a XML baseada em ontologias

por

ADROVANE MARQUES KADE

Dissertação submetida a avaliação,
como requisito parcial para a obtenção do grau de
Mestre em Ciência da Computação

Carlos Alberto Heuser
Orientador

Porto Alegre, junho de 2001

**CIP — CATALOGAÇÃO NA PUBLICAÇÃO**

Kade, Adrovane Marques

Uma linguagem visual de consulta a XML baseada em ontologias / por Adrovane Marques Kade. — Porto Alegre: PPGC da UFRGS, 2001.

85 f.: il.

Dissertação (mestrado) — Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2001. Orientador: Heuser, Carlos Alberto.

1. XML. 2. ontologias. 3. linguagens visuais de consulta. I. Heuser, Carlos Alberto. II. Título.

*Computer programs are built of abstractions at all levels. They are like poems whose language is pure thought, whose form is of science, and whose power, if controlled by any engineering discipline, can be put to extending ourselves and our environment or destroying them.*

Tim Denvir

# Agradecimentos

Agradeço inicialmente, e em especial, à minha esposa, Ana Paula. Sem a tua colaboração e compreensão e apoio e sacrifícios pessoais e tantas outras coisas que não caberiam neste espaço, este trabalho não teria sido possível. Agradeço também aos teus pais, Bianor e Salete, e aos teus irmãos, que me receberam como parte da família.

Agradeço aos meus pais e à minha irmã, pela confiança depositada em mim.

Agradeço à Elceni Gelain e à Graziela Corneli, pela amizade e companheirismo.

Agradeço ao colega Frederico Goldschmidt, companheiro de viagem nesses anos de mestrado.

Agradeço à Eliane, ao Roberto e a todo o pessoal da casa, pelo modo carinhoso com que me receberam e me fizeram sentir em casa.

Agradeço aos colegas de mestrado e doutorado, em especial aos da sala 215, pelo apoio, amizade e companheirismo e idéias e experiências trocadas (principalmente nos bate-papos após o almoço).

Agradeço à Vanessa Braganholo, pela leitura e revisão do texto final.

Agradeço ao meu orientador, Prof. Heuser, não só pelos conhecimentos compartilhados, mas também pela convivência e pelo respeito e cordialidade que demonstrou comigo ao longo da execução deste trabalho.

Agradeço ao Instituto de Informática da Ufrgs, em todos os os seus setores e departamentos com os quais mantive contato.

Agradeço à IBM/Solectron, pelo apoio para execução de parte deste trabalho.

Finalmente, agradeço a todos aqueles que, de forma direta ou indireta, contribuíram para a elaboração deste trabalho.

# Sumário

# Lista de Abreviaturas

| | |
|---|---|
| **DTD** | Document Type Definition |
| **ER** | Entidade-Relacionamento |
| **HTML** | Hypertext Markup Language |
| **IA** | Inteligência Artificial |
| **OQL** | Object Query Language |
| **QBE** | Query By Example |
| **RDF** | Resource Description Framework |
| **SGML** | Standard Generalized Markup Language |
| **SQL** | Structured Query Language |
| **W3C** | World Wide Web Consortium |
| **WWW** | World Wide Web |
| **XML** | eXtensible Markup Language |
| **XSL** | eXtensible Stylesheet Language |
| **UML** | Unified Modeling Language |
| **URL** | Uniform Resource Locator |
| **URI** | Unified Resource Identifier |

# Lista de Figuras

# Lista de Tabelas

# Resumo

O volume de informações armazenadas e representadas em XML cresce rapidamente, abrangendo desde a Web até bancos de dados corporativos. Nesse contexto, surge a necessidade de mecanismos de recuperação de dados nesse formato que sejam, ao mesmo tempo, mais eficientes e mais eficazes. Várias propostas de linguagens de consulta têm sido feitas, dentre as quais podem ser citadas XQL, XML-QL e Quilt. Essas linguagens, todas textuais, são mais indicadas para manipulação programática ou para usuários experientes. Visando atingir também os usuários menos experientes, foram propostas linguagens visuais, tais como XML-GL e Xing. Todas essas linguagens, entretanto, apresentam duas características comuns: a) o usuário precisa conhecer, pelo menos em um certo nível, a estrutura interna dos documentos; b) a mesma informação, se armazenada de formas diferentes, exige instruções de consulta diferentes. A solução para esses problemas apresentada neste trabalho envolve a utilização de um modelo conceitual para representar os conceitos e as relações entre conceitos que ocorrem em documentos XML pertencentes a um determinado domínio de problema. O modelo conceitual é representado por uma ontologia do domínio do problema. Essa associação permite que consultas possam ser elaboradas tendo como base os conceitos da ontologia. Para permitir a associação da ontologia a conjuntos de documentos XML, apresentam-se regras de mapeamento que permitem definir se um documento XML é compatível com uma determinada ontologia. A partir dessa definição, propõe-se uma linguagem visual para consultas a documentos XML com base em ontologias, e apresenta-se uma proposta de interface visual para essa linguagem.

**Palavras-chave:** XML, ontologias, linguagens visuais de consulta.

**TITLE:** “AN ONTOLOGY-BASED XML VISUAL QUERY LANGUAGE”

# Abstract

The amount of information stored and represented in XML is growing fast, reaching the Web as well as corporative database systems. In this context,the need for XML data retrieval strategies becomes apparent. These strategies should be, at the same time, efficient and effective. Several query languages has been proposed with this intent, among them XQL, XML-QL and Quilt. These languages, all textual-based, are more indicated to programmatic manipulation or to expert users. Other languages, visual-based, has been proposed, with the intent of achieve also the less experienced users. XML-GL and Xing are examples of such languages. All this languages, including textual-based and visual-based, have the same two weakness: a) the user must know, at least at a certain level, the internal structure of the documents to be queried; b) an information, if stored in two different forms, requires two different query expressions to be formulated. The solution for these problems presented in this work involves the use of a conceptual model to represent the concepts and the relationships between these concepts that occurr in XML documents that belong to a certain problem domain. This association allows the queries to be posed using the concepts of the ontology. To allow the association between the ontology and groups of XML documents, in this work rules are presented that allow the definition of the compatibility of a document with an ontology. Following this definition, a visual language is proposed to query XML documents based on an ontology. Further that, a visual interface is proposed to deal with this language.

**Keywords:** XML, ontologies, visual query languages.

# 1 Introdução

Nos últimos anos, um novo formato para a representação e o armazenamento de informações, chamado *semi-estruturado*, começou a ganhar importância. A origem do nome se deve ao fato de que, embora os dados semi-estruturados possuam alguma estrutura, ela não é rígida, podendo variar de instância para instância. Juntamente com o modelo de dados semi-estruturado, surgiram linguagens de consulta próprias para ele, dentre as quais pode-se citar a Lorel [ABI 97].

O modelo de dados originalmente proposto para a representação de dados semi-estruturados foi o *Object Exchange Model* – OEM [PAP 95]. Com o passar do tempo, entretanto, começou-se a perceber as semelhanças entre o modelo semi-estruturado e as linguagens de marcação para a Web, em especial a *eXtensible Markup Language* – XML. A percepção de que XML é adequada para representar dados semi-estruturados, e o fato de que a linguagem se tornou padrão para o intercâmbio de informações, fizeram com que ela substituísse o OEM como modelo de dados para a representação semi-estruturada.

Linguagens de marcação não são soluções novas para a representação de informações, principalmente considerando-se que a origem de SGML remonta à década de 1970. SGML, entretanto, não chegou a ser amplamente difundida, devido à sua complexidade e dificuldade de utilização. As linguagens de marcação se tornaram mais conhecidas a partir do surgimento de HTML para representação de informações na Web. HTML difundiu-se devido, principalmente, à simplicidade das marcações padronizadas, cujo objetivo é o de apresentar informações em navegadores da Web.

XML, apesar de sua origem comum com HTML, apresenta várias diferenças com relação a essa linguagem no que diz respeito aos recursos para representação de informações. Em função disso, XML, inicialmente elaborada para representar documentos na Web em substituição a HTML, percorreu um caminho muito mais amplo, tendo sido adotada como linguagem de intercâmbio para vários tipos de aplicações em áreas distintas. Isso fez com que o volume de informações disponíveis nesse formato crescesse a um ritmo acelerado, despertando o interesse na pesquisa de estratégias de consulta eficientes. Para suprir essa demanda, diversas linguagens de consulta foram propostas, dentre as quais XML-QL, XQL, YATL e Quilt.

O grande desafio em consultar XML consiste no fato de que, apesar de incluir alguma semântica nos documentos, XML ainda é uma linguagem de marcação, com função mais sintática. Em [ERD 00], por exemplo, chega-se a afirmar que "os recursos semânticos de XML são freqüentemente superestimados", uma vez que, do ponto de vista computacional, as *tags* `<Estudante>` e `<H1>`, por exemplo, têm o mesmo poder semântico. De fato, o poder de expressão semântica da linguagem restringe-se aos nomes dos elementos e atributos que compõem a estrutura do documento, o que pode ser insuficiente para lidar, por exemplo, com situações nas quais uma mesma informação pode ser representada por elementos com os mais diversos nomes, dependendo da aplicação e até mesmo do idioma utilizado.

Informações pertencentes a um domínio de problema podem ser representadas em XML de diversas formas. Por exemplo, em um domínio de informações acadêmico, no qual deseja-se representar a relação existente entre professores e disciplinas, pode-se, em uma instância XML, representar cada professor seguido de suas disciplinas, enquanto que, em outra instância XML, cada disciplina é seguida de seus professores. Essa multiplicidade de representações advém da estrutura hierárquica de XML, na qual, dados

dois elementos de informação relacionados entre si, é necessário sempre indicar um deles como sendo hierarquicamente superior ao outro.

A representação dos mesmos dados de maneiras diferentes traz dificuldades para muitos tipos de aplicação, tais como consultas e integração de dados. Por exemplo, caso as informações sobre professores e disciplinas estejam armazenadas em documentos XML com estruturas diferentes, é necessário aplicar consultas com sintaxes diferentes para obter as informações de um ou outro documento.

Uma solução para esses problemas consiste em associar um modelo conceitual a conjuntos de documentos XML. Um modelo conceitual descreve *quais* os conceitos que existem em um domínio de aplicação e *como* esses conceitos se relacionam. Dessa forma, quando associado a um conjunto de documentos XML, um modelo conceitual serve como uma representação abstrata dos elementos de informação presentes naquele conjunto de documentos. Assim, pode-se expressar consultas sobre o conjunto de documentos utilizando o modelo conceitual para extrair informações sem que se conheça com exatidão a estrutura hierárquica dos documentos.

O uso de ontologias como modelos conceituais para documentos XML é, dentre as propostas de modelos conceituais para dados XML, a abordagem que tem ganho mais atenção da comunidade científica. Ontologia é um termo originário da filosofia que, desde o princípio da década de 1990, passou a ser utilizado pela comunidade de IA, com um sentido mais restrito. No campo de IA, a definição mais citada de ontologia é a de Gruber [GRU 93]: “uma ontologia é uma especificação explícita de uma conceitualização”. Já [CRA 99] define ontologias como “modelos formalmente especificados de corpos do conhecimento, que definem os conceitos utilizados e os relacionamentos entre eles”. É sob este último ponto de vista que se aplica o conceito de ontologia neste trabalho: um modelo conceitual, que representa os conceitos comuns a um domínio de problema, assim como a relação existente entre eles.

Para associar ontologias a documentos XML, entretanto, deve-se especificar qual a relação que existe entre uma ontologia e uma (ou mais) classes de documentos XML, representadas pelos seus esquemas.

Uma vez definida a relação entre ontologias e esquemas XML, pode-se elaborar consultas sobre o conjunto de documentos utilizando os conceitos da ontologia. Isso, apesar de enriquecer o resultado das consultas, não evita que o usuário tenha que conhecer a sintaxe da linguagem de consulta, o que pode ser indesejável, principalmente se o usuário não for experiente e a linguagem for textual.

Nesse trabalho, propõe-se um conjunto de regras que permitem a verificação da compatibilidade de um esquema XML com uma determinada ontologia. O objetivo é permitir que ontologias sirvam como modelos conceituais para documentos XML, os quais devem ser *compatíveis* com ela. Além disso, propõe-se uma interface visual de consulta a dados XML baseada em ontologias. O objetivo dessa interface é permitir que expressões de consulta possam ser construídas com base na ontologia, sendo posteriormente mapeadas para o conjunto de documentos de interesse do usuário.

Nesse contexto, as principais contribuições desse trabalho são:

1. a definição de regras que estabelecem um mapeamento entre ontologias e esquemas XML;

2. a elaboração de uma linguagem visual de consulta para XML que tem como base uma ontologia do domínio do problema em questão.

A dissertação está estruturada da seguinte forma: o capítulo 2 apresenta um estudo dos requisitos desejáveis para uma linguagem de consulta a XML, bem como as principais linguagens propostas para consultar XML; o capítulo 3 apresenta algumas linguagens visuais, tanto para dados relacionais, quanto para XML; o capítulo 4 define o conceito de ontologia e sua relação com documentos XML; o capítulo 5 apresenta a proposta desse trabalho, definindo a relação existente entre documentos XML e ontologias, bem como uma linguagem visual para construção de consultas XML com base em ontologias e uma interface visual que aplica ambos os conceitos.

## 1.1 Contextualização

O presente trabalho desenvolve-se no contexto do projeto *Intelligent Document* – IDOC, parceria da Universidade Federal do Rio Grande do Sul com CEFET-PR, PUC-PR e a empresa Pólo de Software, de Curitiba-PR. O objetivo do projeto é propor uma arquitetura de armazenamento e recuperação de documentos estruturados. A arquitetura geral do projeto encontra-se na fig. 1.1.

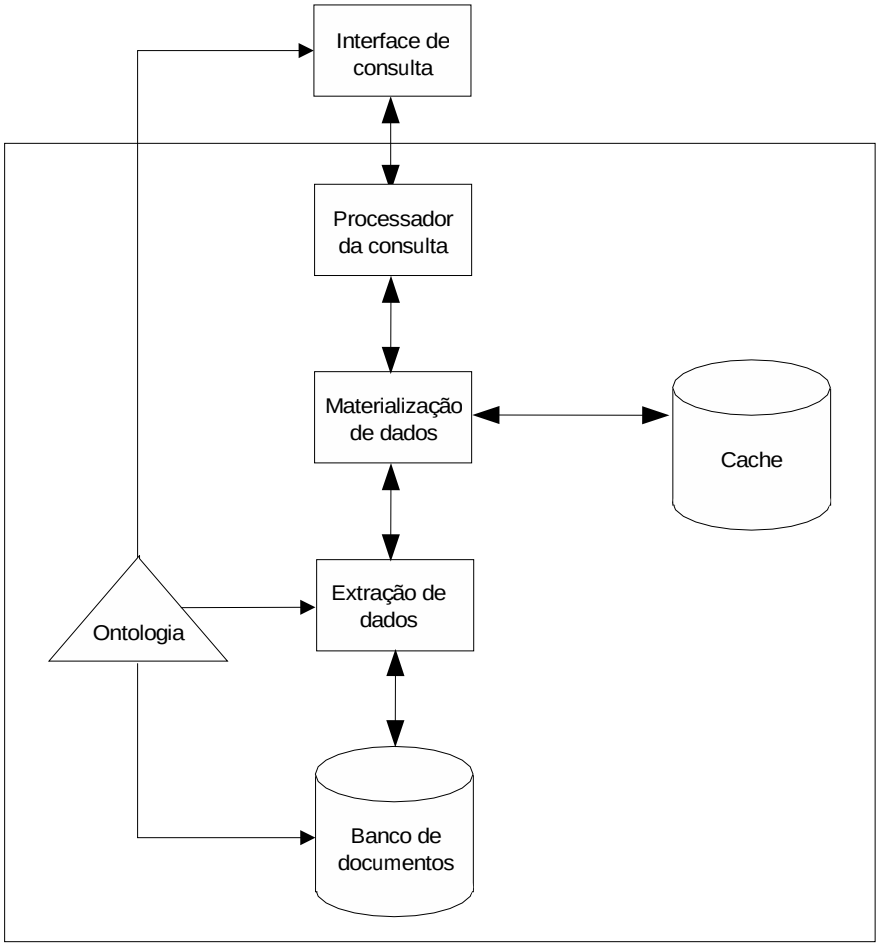


FIGURA 1.1 – Arquitetura do projeto IDOC.

Pode-se observar na figura que a base de toda a arquitetura é uma ontologia de domínio. Documentos relacionados ao domínio que a ontologia representa são armaze-

nados em um banco de dados. Esses documentos podem ser recuperados por meio da interface de consulta proposta nesse trabalho. As requisições de consulta do usuário são expressas com base na ontologia do domínio do problema, e são repassadas ao materializador de dados. Este, por sua vez, verifica se os dados solicitados já estão armazenados na *cache* de dados. Se os dados já estão na *cache*, o materializador devolve à interface os dados solicitados; caso contrário, ele faz uma requisição ao extrator de dados, para que os busque no banco de documentos. Se os dados solicitados estiverem no banco, o extrator os repassará ao materializador, e esse para a interface de consulta.

Este trabalho trata exclusivamente da interface de consulta, vista bem ao alto na figura. Informações adicionais sobre o projeto IDOC podem se encontrados em [DOR 00] e em [SIL 00].

## 1.2 Trabalhos relacionados

Ontologias têm sido discutidas no contexto de modelos conceituais para XML em diversos trabalhos, dentre os quais [ERD 00], no qual se apresenta uma metodologia para a criação automática de um esquema XML a partir de uma ontologia, com vistas à recuperação de dados de diversas fontes, entre as quais a Web.

Em [DOR 00] apresenta-se um algoritmo para criação de DTDs, tendo como ponto de partida uma ontologia. O objetivo, à semelhança do trabalho anteriormente citado, é o de recuperar dados de um repositório de documentos, utilizando a ontologia como esquema para a extração.

Outros trabalhos que envolvem linguagens visuais de consulta a XML são [CER 98], que apresenta a XML-GL, e [ERW 00], no qual se apresenta a Xing. Deve-se ressaltar, entretanto, que essas linguagens foram projetadas para considerar documentos XML não acompanhados de modelos conceituais.

# 2 Linguagens de consulta para XML

Neste capítulo, apresentam-se os conceitos e as propostas de linguagens para consulta a XML, com o objetivo de compreender os recursos que elas possuem. A avaliação desses recursos é necessária para a compreensão das funcionalidades que elas oferecem aos usuários, assim como das facilidades e das dificuldades na sua utilização.

Várias linguagens foram propostas para consultar XML, desde extensões de padrões já aceitos pela indústria (SQL e OQL) até linguagens apresentadas como sendo inteiramente *novas*. Dentro desse contexto, de acordo com [BAR 98], percebem-se dois paradigmas:

a) o paradigma de bancos de dados, que apresenta linguagens *SQL-like* ou *OQL-like*;

b) o paradima de programação funcional, baseado em XSL e XQL.

Na avaliação do autor, as primeiras são convenientes para seleção e integração de objetos, mas não suportam a reestruturação de estruturas profundas; enquanto que as últimas são particularmente adequadas para transformações profundas de documentos XML.

Há também propostas de linguagens que buscam unificar características de ambos os paradigmas, agregando em uma linguagem recursos apropriados para consultar dados e documentos XML. Dentre essas, destaca-se a linguagem Quilt.

O W3C, organização que define padrões para a Web, criou, a partir do *Workshop on Query Languages*, em 1998, um comitê para estudar e definir as bases para uma linguagem de consulta a XML. Com resultado, foi proposta a XQuery – *XML Query Language* [CHA 01]. A linguagem do W3C se baseia em duas outras propostas: o *XML Query Data Model* [FER 00], que define um modelo de dados para consultas a XML, e o *XML Query Requirements* [CHA 00] , que especifica requisitos para uma linguagem de consulta a XML, considerando-se o ambiente Web.

A diversidade de linguagens propostas para consulta XML advém do fato de que informações pertencentes aos mais variados contextos são representadas em XML, desde documentos semi-estruturados, tais como artigos ou relatórios, até dados extraídos de bancos relacionais, fortemente estruturados. Nesse sentido, espera-se que uma linguagem de consulta consiga lidar com esses diversos formatos e extrair informações com a mesma facilidade de documentos escritos em qualquer um deles.

Nesse contexto, características desejáveis em uma linguagem de consulta para XML podem ser classificadas em:

a) *Genéricas*, tais como seleção e extração de elementos, reestruturação, ordenação e expressões regulares de caminho;

b) *Típicas de bancos de dados*, tais como junção, quantificadores universal e existencial, agrupamento e funções de agregação;

c) *Típicas de documentos*, tais como manutenção da ordem dos elementos e consultas sobre essa ordem (consulta indexada).

Nesse capítulo, são analisadas cinco linguagens de consulta: Lorel [ABI 97] e [GOL 98], XML-QL [DEU 99], XQL [ROB 98], YATL [CLU 00] e Quilt [CHA 00a]. Algumas dessas características já foram analisadas em outros estudos, tais como [FER 00],

[BON 00] e [IVE 00]. As consultas apresentadas como exemplo são semelhantes àquelas apresentadas em [FER 00], com a diferença de que a forma de apresentação foi modificada, para enfatizar as três classes apresentadas acima. Além disso, apresentam-se recursos de XQL não tratados em [FER 00] e discute-se a linguagem Quilt, que não é considerada naquele trabalho. No final, apresenta-se um quadro-resumo das características das linguagens apresentadas.

Como exemplo para as consultas, utilizar-se-á a DTD a seguir, que representa um esquema para uma base de dados bibliográfica.

```
<!ELEMENT biblio (livro | artigo)*>
<!ELEMENT livro (autor+,titulo,editora)>
<!ATTLIST livro ano CDATA>
<!ELEMENT titulo (#PCDATA)>
<!ELEMENT editora (nome,endereço)>
<!ELEMENT nome (#PCDATA)>
<!ELEMENT endereço (#PCDATA)>
<!ELEMENT artigo (autor+, titulo, ano?, (versaoresumo|versaocompleta))>
<!ATTLIST artigo tipo CDATA>
<!ELEMENT nome (#PCDATA)>
<!ELEMENT versaoresumo EMPTY>
<!ELEMENT versaocompleta EMPTY>
<!ELEMENT autor (nome?,sobrenome)>
<!ELEMENT sobrenome (#PCDATA)>
```

A DTD é formada por um elemento raiz `biblio`, que pode conter uma ou mais ocorrências de elementos `livro` e/ou `artigo`. Um livro pode ter um ou mais autores, bem como um título e uma editora, além de um atributo contendo o ano de sua publicação. Cada artigo pode ter, obrigatoriamente, um ou mais autores, um título e um elemento informando se o artigo é um resumo ou uma versão completa. Além disso, o artigo deve possuir um atributo que informa o seu tipo e, opcionalmente, um elemento que informa o ano de publicação. A editora deve ter o nome e o endereço, e o autor deve ter o sobrenome e, opcionalmente, o nome. Apenas para efeito de referência nas consultas, assume-se que os dados podem ser localizados no endereço “http://www.inf.ufrgs.br/biblio.xml”.

## 2.1 Seleção e extração

O tipo de consulta mais básico em XML consiste em selecionar os elementos de interesse e extrair aqueles que não são desejáveis. A seguir, considera-se a consulta “retornar o título e o ano de todos os livros publicados pela editora Saraiva em 1999” para todas as linguagens em questão.

### Lorel

A estrutura de uma consulta em Lorel segue a sintaxe básica de SQL e OQL, “select-from-where”. A cláusula *from* contém padrões que combinam com a estrutura do documento de origem, aos quais são associadas variáveis. Essas variáveis podem ser filtradas na cláusula *where*. Os elementos que farão parte do resultado devem ser especificados na cláusula *select*.

```
select xml(biblio:{
```

```
  (select xml(livro:{titulo:t, @ano:a})
   from biblio.livro l, l.titulo t, l.ano a
   where l.editora = ''Saraiva'' and a > 1999)})
```

**XML-QL**

Em XML-QL, a cláusula `WHERE` estabelece padrões que buscam similaridade na estrutura dos documentos e liga variáveis a esses padrões, que podem ser referenciadas na cláusula `CONSTRUCT` para a construção de um novo documento resultado. No exemplo abaixo, a cláusula `WHERE` gera tuplas contendo valores para o par de variáveis (`$a, $t`), que são utilizados na cláusula `CONSTRUCT`.

```
CONSTRUCT <biblio> {
  WHERE
    <biblio>
      <livro ano=$a>
        <titulo>$t</titulo>
        <editora><nome>Saraiva</nome></editora>
      <livro>
    </biblio> IN ''www.inf.ufrgs.br/biblio.xml'',
    $a > 1999
  CONSTRUCT <livro ano=$a><titulo>$t</titulo></livro>
}
```

**XQL**

Em XQL, a consulta consiste em uma expressão de caminho, que navega pelo documento. Opcionalmente, podem ser expressos filtros sobre elementos e atributos, conforme se observa no exemplo abaixo.

```
document("http://www.inf.ufrgs.br/biblio.xml")/biblio {
  livro[editora/nome = "Saraiva" and @ano > 1999] {
    @ano | titulo
  }
}
```

**YATL**

Nas consultas YATL, o resultado é especificado na cláusula `make`, enquanto que os padrões aplicados sobre o documento aparecem na cláusula `match`, e critérios de seleção na cláusula `where`. Da mesma forma que XML-QL, YATL também cria tuplas para a combinação de variáveis da cláusula `match`.

```
make
  biblio [ *livro [ @ano [ $a ],
                    titulo [ $t ] ] ]
match
  biblio [ *livro [ @ano [ $a ],
                    titulo [ $t ],
                    editora [ nome [ $n ] ] ]
where
  $n = "Saraiva" and $a > 1999
```

### Quilt

A sintaxe de Quilt permite que consultas simples possam ser construídas pelo uso de expressões de caminho, a exemplo de XQL. Entretanto, a sintaxe mais abrangente é a que utiliza as cláusulas `FOR`, `LET`, `WHERE` e `RETURN` (chamadas expressões FLWR, as quais se lê *flower*). As cláusulas `FOR` e `LET` associam uma ou mais variáveis aos valores resultantes de uma expressão. Essas variáveis podem ser filtradas na cláusula `WHERE` e utilizadas como parte do resultado, especificado na cláusula `RETURN`.

```
FOR $l IN document("http://www.inf.ufrgs.br/biblio.xml")//livro
WHERE $l/editora/nome = "Saraiva" AND $l/@ano > "1999"
RETURN
  <biblio>
    <livro ano=$l/@ano>
      $l/titulo
    </livro>
  </biblio>
```

## 2.2 Ordenação

Na consulta da seção anterior, não foi especificada a ordem em que os livros devem aparecer no resultado. As consultas dessa seção são basicamente as mesmas da seção anterior, com a única diferença de que os resultados são ordenados pelo título do livro.

### Lorel

Em Lorel, indica-se a ordem dos elementos no resultado pela cláusula `order by`, seguida pelo nome das variáveis que representam os elementos que se deseja ordenar.

```
select xml(biblio:{
  (select xml(livro:{titulo:t, @ano:a})
   from biblio.livro l, l.titulo t, l.ano a
   where l.editora = "Saraiva" and a > 1999
   order by t)})
```

### XML-QL

O esquema de ordenação de XML-QL é basicamente o mesmo de Lorel. Existe uma cláusula, `ORDER-BY`, que deve conter o nome das variáveis cujo conteúdo se deseja ordenar.

```
CONSTRUCT <biblio> {
  WHERE
    <biblio>
      <livro ano=$a>
        <titulo>$t</titulo>
        <editora><nome>Saraiva</nome></editora>
      <livro>
    </biblio> IN "www.inf.ufrgs.br/biblio.dtd",
    $a > 1999
  ORDER-BY $t
```

```
  CONSTRUCT <livro ano=$a><titulo>$t</titulo></livro>
}
```

**XQL**

Da forma como foi proposta inicialmente, XQL não possuía recursos explícitos de ordenação. A cláusula `sortby` foi adicionada somente na última especificação da linguagem, conhecida como XQL'99.

```
document("http://www.inf.ufrgs.br/biblio.xml")/biblio {
  livro[editora/nome = "Saraiva" and @ano > 1999] {
    @ano | titulo
  } sortby titulo
}
```

**YATL**

A ordenação em YATL é indicada pela construção `o(variável)` na cláusula `make`, conforme se pode observar pela consulta a seguir. Nesse caso, o resultado da consulta será ordenado pelo conteúdo da variável `$t`, ou seja, pelos títulos dos livros.

```
make
  biblio [ *o($t) livro [ @ano [ $a ],
                          titulo [ $t ] ] ]
match
  biblio [ *livro [ @ano [ $a ],
                    titulo [ $t ],
                    editora [ nome [ $n ] ] ]
where
  $n = "Saraiva" and $a > 1999
```

**Quilt**

Em Quilt, a ordenação é indicada pela cláusula `sortby`, que pode ser, opcionalmente, acompanhada pela palavra `ASCENDING`, que é o padrão e indica ordenação ascendente, ou `DESCENDING`, que indica ordenação descendente.

```
FOR $l IN document("http://www.inf.ufrgs.br/biblio.xml")//livro
WHERE $l/editora/nome = ''Saraiva'' AND $l/@ano > "1999"
RETURN
  <biblio>
    <livro ano=$l/@ano>
      $l/titulo
    </livro> SORTBY (titulo ASCENDING)
  </biblio>
```

## 2.3 Reestruturação

Reestruturação é uma característica desejável às linguagens de consulta a XML, no sentido em que o usuário pode desejar obter no resultado um documento com estrutura

diferente do documento original. Características de reestruturação são implementadas nas linguagens de consulta utilizando três mecanismos:

- consultas aninhadas, apresentados por XML-QL, YATL, Lorel e Quilt;
- funções *Skolem*, que estão disponíveis em XML-QL e Lorel;
- operadores específicos de agrupamento, a exemplo do que está disponível em YATL.

Para exemplificar essas características, utiliza-se a consulta "agrupar cada autor com os títulos dos livros que ele escreveu".

**Lorel**

Reestruturação é suportada em Lorel por meio de consultas aninhadas, bem como funções *Skolem*, conforme demonstrado a seguir. O primeiro exemplo resolve o problema utilizando consultas aninhadas. Neste caso, a consulta interna seleciona os livros, que são agrupados para cada autor gerado pela consulta externa.

```
select xml(resultados:{
  select xml(result:{autor: a,
                     (select xml(titulo:t)
                      from biblio.livro l, l.titulo t
                      where l.autor.nome = a.nome and
                            l.autor.sobrenome = a.sobrenome)})
  from biblio.livro.autor a })
```

A consulta a seguir tem o mesmo resultado que a anterior, utilizando, para isso, duas funções *Skolem*. A primeira, `Root()`, não recebe parâmetros e cria um único elemento, com múltiplos sub-elementos `result`. Um sub-elemento `result` é criado pela outra função, `Autor(n,s)`, para cada par distinto de nome e sobrenome do autor, indicados por `n` e `s`, respectivamente. Os elementos criados por `Autor` têm sub-elementos para o autor e para os títulos dos livros.

```
select Root()->result->Autor(n,s),
       Autor(n,s)->autor->a,
       Autor(n,s)->titulo->t
from biblio.livro l, l.autor a, a.nome n, a.sobrenome s, b.titulo t
```

**XML-QL**

Da mesma forma que Lorel, XML-QL também suporta agregação por consultas aninhadas e por funções *Skolem*. No primeiro caso, as variáveis `$n` e `$s` são "amarradas" no `WHERE` externo, e testadas por igualdade no `WHERE` interno. A consulta interna constrói os títulos dos livros para cada par formado por nome e sobrenome de autor.

```
CONSTRUCT <resultados> {
  WHERE
    <biblio>
      <livro>
        <autor><nome>$n</nome><sobrenome>$s</sobrenome></autor>
```

```
      </livro>
    </biblio> IN "www.inf.ufrgs.br/biblio.xml"
  CONSTRUCT
    <resultado>
      <autor><nome>$n</nome><sobrenome>$s</sobrenome></autor>
      {
      WHERE
        <biblio>
          <livro>
            <titulo>$t</titulo>
            <autor><nome>$n</nome><sobrenome>$s</sobrenome></autor>
          </livro>
        </biblio> IN "www.inf.ufrgs.br/biblio.xml"
      CONSTRUCT <titulo>$t</titulo>
      }
    </resultado>
}
```

A consulta com função *Skolem* inclui no elemento resultado um atributo XML do tipo ID para cada combinação de nome e sobrenome. Atributos do tipo ID são identificadores de elementos, ou seja, dado um tipo de elemento, não podem haver duas ocorrências do mesmo elemento com valores idênticos para o atributo do tipo ID.

```
CONSTRUCT <resultados> {
  WHERE
    <biblio>
      <livro>
        <titulo>$t</titulo>
        <autor><sobrenome>$s</sobrenome><nome>$n</nome></autor>
      </livro>
    </biblio> IN "www.inf.ufrgs.br/biblio.xml"
  CONSTRUCT
    <resultado ID=autor($n,$s)>
      <titulo>$t</titulo>
      <autor><sobrenome>$s</sobrenome><nome>$n</nome></autor>
    </resultado>
}
```

**XQL**

A especificação original de XQL trabalhava especificamente com expressões de caminho, o que inviabilizaria a construção dessa consulta. Revisões posteriores, entretanto, adicionaram a possibilidade de agrupamento por valor, permitindo expressar a consulta, conforme abaixo.

```
document("www.inf.ufrgs.br/biblio.xml")/bib {
  autor { nome |
    for ($t := livro/itulo) {
      <livro> { $t }
    }
  }
}
```

**YATL**

Essa consulta pode ser expressa em YATL de duas formas: utilizando consultas aninhadas ou por meio do operador de agrupamento da linguagem. No primeiro caso, a consulta é semelhante à de XML-QL.

```
make
  resultados [
    *resultado [
      autor [ sobrenome [ $s ], nome [ $n ] ],
      ( make
          *titulo [ $t ]
        match "www.inf.ufrgs.br/biblio.xml" with
          bib [ *livro [ *autor [ sobrenome [ $s ], nome [ $n ] ],
                         titulo [ $t ] ] ] ) ] ]
match "www.inf.ufrgs.br/biblio.xml" with
  biblio [ *livro [ *autor [ sobrenome [ $s ], nome [ $n ] ] ] ]
```

O outro tipo de consulta utiliza o operador de agrupamento, representado por `($n,$s)` entre o `*` e o `resultado`, na cláusula `make`. Isso indica que o resultado será agrupado pela combinação do nome e do sobrenome do autor.

```
make
  resultados [ *($n,$s) resultado [ autor [ sobrenome [ $s ],
                                            nome [ $n ] ],
                                    *titulo [ $t ] ] ]
match "www.inf.ufrgs.br/biblio.xml" with
  biblio [ *livro [ titulo [ $t ],
                    *autor [ sobrenome [ $s ],
                             nome [ $n ] ] ] ]
```

**Quilt**

Em Quilt, a consulta é feita utilizando-se uma consulta aninhada, cuja estrutura é semelhante à de XML-QL e YATL.

```
FOR $l IN document("www.inf.ufrgs.br/biblio.xml")//livro
RETURN
  $l/autor/nome,
  <autores>
    FOR $t IN document("www.inf.ufrgs.br/biblio.xml")//livro
        [autor/nome = $l/autor/nome]
    RETURN $t/titulo
  </autores>
```

## 2.4 Expressões de caminho regular

Em determinadas situações, pode ser necessário restringir a consulta a um caminho específico dentro da árvore de documentos, utilizando-se expressões de caminho regular (*regular-path expressions*). O exemplo, nesse caso, se baseia na DTD a seguir, que define um elemento, `secao`, que é recursivo.

```
<!ELEMENT capitulo (titulo, secao*)>
<!ELEMENT secao    (titulo, secao*)>
<!ELEMENT titulo   (#PCDATA)>
```

Percebe-se na DTD que o elemento `secao` pode conter outros elementos de mesmo nome, em qualquer nível de aninhamento. A consulta proposta é “recupera todos os elementos `secao` ou `capitulo` que contenham a palavra ’XML’, independente do nível de aninhamento”.

**XML-QL**

Em XML-QL, aplica-se uma expressão de caminho regular (no exemplo, `capitulo.(secao)*`). Expressões de caminho regular em XML-QL podem ser combinadas com os operadores de alternância (`|`), concatenação (.) e repetição (*).

```
CONSTRUCT <resultado> {
  WHERE
    <capitulo.(secao)*>
      <titulo>$t</titulo>
    </> IN "livros.xml",
  $t like ’*XML*’
  CONSTRUCT
    <titulo>$t</titulo>
}
```

**Lorel**

Em Lorel, expressões de caminho regular podem ser associadas a variáveis na cláusula `from`. No exemplo, a expressão `capitulo(.secao)* s` liga a variável `s` a todos os elementos que são atingidos seguindo-se um capítulo e uma seqüência de seções.

```
select xml(resultado:{
  select xml(titulo:t)
  from capitulo(.secao)* s, s.titulo t
  where t like "*XML*" })
```

**XQL**

XQL não suporta expressões de caminho regular, mas possui os operadores `/` e `//`, que permitem o acesso aos descendentes diretos e indiretos (em qualquer nível) do elemento atual. Assim, pode-se escrever `//secao` para atingir as seções do documento, em qualquer nível. Deve-se observar, entretanto, que a consulta abaixo não obriga que as seções estejam aninhadas dentro dos capítulos, conforme exige o enunciado do problema.

```
document("livros.xml")->resultados {
  capitulo[titulo contains "XML"] { titulo } |
  .//secao[titulo contains "XML"] { titulo }
}
```

**YATL**

A linguagem atualmente não suporta expressões de caminho regular.

**Quilt**

A linguagem atualmente não suporta expressões de caminho regular. A consulta proposta, entretanto, pode ser expressa com a utilização do operador `FILTER`, que recebe dois operadores: o primeiro especifica uma floresta de nodos como origem da consulta, enquanto que o segundo estabelece aqueles nodos da árvore que serão mantidos no resultado. No exemplo abaixo, o primeiro operador é `document("livros.xml")`, e o segundo é `//capitulo | //capitulo/titulo | //secao | //secao/titulo`. Adicionalmente, pode-se manter também a estrutura hierárquica, ou seja, as seções de um mesmo capítulo continuarão associadas a esse capítulo. A consulta, por outro lado, não obriga que seções estejam aninhadas dentro de capítulos.

```
<resultado>
document("livros.xml") FILTER //capitulo | //capitulo/titulo |
                              //secao | //secao/titulo
</resultado>
```

## 2.5 Junções

A característica de junção, em bancos de dados relacionais, permite combinar dados de duas ou mais tabelas, com base no conteúdo comum de algum atributo. Em XML, essa operação possibilita combinar elementos de fontes de dados diferentes, com base no conteúdo destes elementos.

Para os exemplos a seguir, considera-se, além da fonte de dados utilizada até aqui, um outro documento que contém títulos de livros, resumos e preços. Assume-se que os dados, representados pela DTD abaixo, estão no endereço “http://www.inf.ufrgs.br/resumos.xml”. A consulta proposta consiste em “retornar o título e o ano de publicação do livro, juntamente com o resumo e o preço”.

```
<!ELEMENT marketing (livro)*>
<!ELEMENT livro (titulo, resumo, preço)>
<!ELEMENT titulo (#PCDATA)>
<!ELEMENT resumo (#PCDATA)>
<!ELEMENT preço (#PCDATA)>
```

**Lorel**

Em Lorel, indica-se o elemento `marketing` como fonte de dados na cláusula `from`, juntamente com o elemento `biblio`. A junção é feita pelo teste de igualdade do título do livro em ambas as fontes de dados.

```
select xml(preços-de-livros:{
  (select xml(livro:{titulo:tl, ano:a, resumo:r, preço:p})
   from biblio.livro l, l.titulo tl, l.ano a
        resumos.marketing m, m.titulo tm, m.resumo r, m.preço p
   where tl = tm )})
```

**XML-QL**

Em XML-QL, as duas fontes de dados que deverão ser combinadas são indicadas na cláusula `WHERE`. O uso da mesma variável (`$t`) para representar o título do livro em

ambas as fontes força a combinação dos valores desses elementos, resultando em uma junção pelo seu valor.

```
CONSTRUCT <preços-de-livros> {
  WHERE
    <biblio>
      <livro ano=$a>
        <titulo>$t</titulo>
      <livro>
    </biblio> IN ‘‘www.inf.ufrgs.br/biblio.xml’’,
    <marketing>
      <livro>
        <titulo>$t</titulo>
        <resumo>$r</resumo>
        <preço>$p</preço>
      </livro>
    </marketing> IN ‘‘www.inf.ufrgs.br/resumos.xml’’
  CONSTRUCT <livro ano=$a>
              <titulo>$t</titulo>
              <resumo>$r</resumo>
              <preço>$p</resumo>
            </livro>
}
```

**XQL**

Junções em XQL são feitas pela ligação de uma variável ao conteúdo de um determinado elemento, e sua posterior utilização para restringir os valores selecionados. No exemplo, atribui-se à variável `$t` o valor dos títulos de livros e, posteriormente, utiliza-se o predicado `titulo=$t` para fazer a associação com o documento que contém os resumos.

```
document("www.inf.ufrgs.br/biblio.xml")/biblio -> preços-de-livros {
  livro[$t:=titulo] {
    @ano | titulo |
    document("www.inf.ufrgs.br/resumos.xml")/marketing/livro
              [titulo=$t]/resumo |
   document("www.inf.ufrgs.br/resumos.xml")/marketing/livro
              [titulo=$t]/preço
  } sortby titulo
}
```

**YATL**

Desconsiderando-se as diferenças sintáticas, a consulta YATL é praticamente idêntica à de XML-QL.

```
make
  preços-de-livros
    *livro [ @ano [ $a ],
             titulo [ $t ],
             resumo [ $r ],
             preço [ $p ] ]
```

```
match "www.inf.ufrgs.br/biblio.xml" with
  biblio [ *livro [ @ano [ $a ],
                    titulo [ $t ] ] ],
      "www.inf.ufrgs.br/resumos.xml" with
  marketing [ *livro [ titulo [ $t ],
                       resumo [ $r ],
                       preço  [ $p ] ] ]
```

**Quilt**

A sintaxe para junções em Quilt é semelhante à de XQL, ou seja, uma variável é associada a um determinado elemento e, após, o valor da variável é comparado com o valor de outro elemento.

```
FOR $l IN document("www.inf.ufrgs.br/biblio.xml")//livro,
    $r IN document("www.inf.ufrgs.br/resumos.xml")//livro[titulo = $l/titulo]
RETURN
  <preços-de-livros>
    <livro ano=$l/@ano>
      $l/titulo,
      $r/resumo,
      $r/preço
    </livro>
  </preços-de-livros>
```

## 2.6 Quantificadores existencial e universal

Em determinadas consultas, pode ser importante verificar se uma determinada característica ocorre para um ou para todos os elementos do resultado. Por exemplo, pode-se desejar saber todos os pares de livros que contém exatamente os mesmos autores.

**Lorel**

O quantificador existencial *for all ... exists* é utilizado para resolver essa consulta em Lorel. O primeiro filtro verifica se os autores de `x` são os mesmos de `y`, enquanto que o segundo filtro faz exatamente o contrário.

```
select xml(livro1: x, livro2: y)
from biblio.livro x, biblio.livro y
where for all z in x.autor: exists w in y.autor: z = w and
      for all t in y.autor: exists s in x.autor: t = s;
```

**YATL**

Em YATL, dois conjuntos podem ser testados para verificar se eles contém exatamente os mesmos elementos. No exemplo, `$a1` e `$a2` contém os conjuntos de autores para cada livro `$l1` e `$l2`, enquanto que o filtro `$a1 = $a2` testa pela igualdade dos conjuntos.

```
make
  * [ livro1 [ $l1 ],
```

```
      livro2 [ $l2 ] ]
match URL with
  *livro($l1) { *($a1) autor },
      URL with
  *livro($l2) { *($a2) autor },
where $a1 = $a2
```

**Quilt**

A linguagem dispõe dos quantificadores existencial (`SOME`) e universal (`EVERY`). A solução do problema proposto envolve a utilização do quantificador universal, conforme abaixo.

```
FOR $l1 IN //livro,
    $l2 IN //livro
WHERE $l1 != $l2 AND
    EVERY $a1 IN $l1/autor SATISFIES $a1 = $l2/autor AND
    EVERY $a2 IN $l2/autor SATISFIES $a2 = $l1/autor
RETURN
  <mesmos-autores>
     <livro1>$l1/titulo</livro1>,
     <livro2>$l2/titulo</livro2>
  </mesmos-autores>
```

**XQL**

Quantificadores universal (`all`) e existencial (`any`) têm um uso mais restrito em XQL, funcionando apenas dentro de uma expressão. No exemplo, buscam-se os títulos de todos os livros que possuem ao menos um autor cujo nome seja “João da Silva”.

```
document("www.inf.ufrgs.br/biblio.xml")/livro/titulo
          [any autor/nome = 'João da Silva']
```

## 2.7 Funções de agregação

Em aplicações de bancos de dados tradicionais, é freqüente a necessidade de agrupar um determinado conjunto de valores e aplicar sobre ele uma função de agregação para obter a soma, a média ou o resultado de outros cálculos sobre os valores. Das linguagens estudadas, apenas Quilt e Lorel implementam funções de agregação, apesar de outras, como XML-QL, fazerem referência nas suas especificações. Em XQL, há apenas uma função de agregação, `count`, definida como uma extensão da linguagem. A consulta proposta é “retornar o preço médio dos livros publicados pela editora ’Saraiva’ no ano de 2000”.

**Lorel**

As funções de agregação já existentes em SQL e OQL estão também disponíveis em Lorel: `min`, `max`, `count`, `sum`, `avg`, para retornar o elemento de menor valor dentro de um conjunto; o de maior valor; o número de elementos do conjunto; a soma dos valores dos elementos do conjunto e a média dos valores do conjunto, respectivamente.

```
select xml(preços-médios:{
  (select xml(preco-medio:avg(r.preco))
   from biblio.livro b, resumos.livro r
   where b.editora = "Saraiva" and
         b.@ano="2000" and
         b.titulo = r.titulo )})
```

**Quilt**

Em Quilt, as seguintes funções de agregação estão disponíveis: `avg`, `sum`, `count`, `max` e `min`. A consulta a seguir utiliza a função `avg` para resolver o problema proposto.

```
FOR $l IN document("www.inf.ufrgs.br/biblio.xml")//livro,
    $r IN document("www.inf.ufrgs.br/resumos.xml")//livro[titulo = $l/titulo]
WHERE $l/editora/nome = "Saraiva" AND $l/@ano="2000"
RETURN <preço-médio> avg($r/preço) </preço-médio>
```

Quando se trabalha com dados originários de bancos de dados relacionais, normalmente a ordem dos elementos não é importante. No processamento de documentos, entretanto, a ordem em que os elementos aparecem no documento deve ser respeitada quando do processamento de consultas sobre eles. Não se admite, por exemplo, que a ordem das seções de um artigo apareça trocada no resultado de uma consulta efetuada sobre ele. Assim, as características que uma linguagem de consulta deve possuir para consultar documentos XML dizem respeito, especialmente, à manutenção da ordem dos elementos no resultado e ao poder de consultar elementos por sua posição dentro de um conjunto de elementos.

## 2.8 Manutenção da ordem dos elementos

De acordo com [FER 00], XQL sempre preserva a ordem do documento, enquanto que XML-QL, YATL e Lorel exigem que o ordenamento seja indicado explicitamente, ou seja, deve-se especificar a condição de ordenação na consulta. Em Quilt, a ordem dos elementos do documento de origem sempre é preservada no documento-resultado.

## 2.9 Consultas sobre a ordem (índices)

Em documentos que possuem uma ordem intrínseca, pode ser desejável que se busque um elemento com base na ordem em que ele se encontra dentro do documento. A consulta proposta busca o título do livro, juntamente com o primeiro e o segundo autores; caso haja três ou mais autores, será incluído um elemento `<et-al/>`.

**Lorel**

Em Lorel, utilizam-se duas consultas aninhadas, nas quais um índice é aplicado sobre o elemento `<autor>` para testar a sua posição dentro do conjunto. Na segunda consulta, um quantificador existencial é utilizado para testar a existência do terceiro autor.

```
select xml(biblio:{
```

```
  selct xml(livro:{ titulo t,
                    (select l.autor[1-2]),
                    (select xml(et-al {})
                     where exists l.autor[3]) })
  from biblio.livro l, l.titulo t })
```

**XML-QL**

A consulta em XML-QL não é muito diferente da de Lorel; a principal diferença é a utilização de uma variável (chamada *variável de índice*) para armazenar o índice e testar a posição do elemento. No exemplo, `$i` é a variável de índice, e seu valor inicia em zero (que, nesse caso, corresponde ao elemento `<titulo>`, e é incrementado para cada um dos elementos de mesmo nível, no caso, os elementos-filhos de `<livro>`. Assim, `<titulo>` tem `$i` = 0, o primeiro autor tem `$i` = 1, o segundo tem `$i` = 2, e assim sucessivamente.

```
CONSTRUCT <biblio> {
  WHERE
   <biblio>
     <livro>
       <titulo>$t</titulo>
       <autor[$i]>$a</autor>
     </livro>
   </biblio> IN "www.inf.ufrgs.br/biblio.xml"
  CONSTRUCT
   <livro ID=titulo($t)>
     <titulo>$t</titulo>
     { WHERE $i <= 2 CONSTRUCT <autor>$a</autor> }
     { WHERE $i = 3 CONSTRUCT <et-al/> }
   </livro>
}
```

**YATL**

Em YATL, a consulta é semlhante à de XML-QL, com a diferença de que a variável de índice é indicada por `$$`.

```
make
  biblio *livro [ titulo [ $t ],
                ( make *autor [ $a ]
                  match $as with *($$i) autor [ $a ]
                  where $$i <= 2 ),
                ( make [ et-al ]
                  match $as with *($$i) autor
                  where $$i = 3 ) ]
match "www.inf.ufrgs.br/biblio.xml" with
  biblio [ *livro [ titulo [ $t ],
                    *($as) autor ] ]
```

**XQL**

A consulta XQL utiliza subscritos para indicar índices. Um subscrito pode conter números simples, intervalos ou qualquer combinação deles. XQL também suporta os

operadores BEFORE e AFTER, que testam se um elemento está localizado no documento antes ou depois do elemento especificado.

```
document("www.inf.ufrgs.br/biblio.xml")/biblio/livro {
         titulo | autor[1 to 2] | autor[3] -> et-al { }
}
```

**Quilt**

A consulta em Quilt utiliza um teste condicional (IF-THEN-ELSE) para verificar a existência do terceiro autor. Os dois primeiros autores são encontrados por um teste de intervalo, que faz uso do operador RANGE. Além disso, Quilt também suporta os operadores BEFORE e AFTER, da mesma forma que XQL.

```
FOR $l IN document("www.inf.ufrgs.br/biblio.xml")//livro
RETURN
  <autores>
     $p/autor[RANGE 1 TO 2]
     IF count($p/autor) >= 3
     THEN <et-al/>
  </autores>
```

## 2.10 Coerção

Uma das características mais marcantes de Lorel é a *coerção*, ou seja, a linguagem força que comparações entre objetos e/ou valores façam "a coisa mais intuitiva", em vez de resultar em erro, quando são comparados objetos ou valores de tipos diferentes. Dois tipos de coerção são executados por Lorel: de tipos de dados (string, inteiro, etc.) e de valores contra objetos complexos. Como a noção de tipos de dados ainda não está bem definida em XML, o exemplo dessa seção trata do segundo tipo de coerção.

Em Lorel, variáveis podem ser atribuídas a valores atômicos, objetos atômicos, objetos complexos ou conjuntos de objetos. Em linguagens tradicionais, comparações entre tipos de objetos diferentes produziria um erro, o que não acontece em Lorel, em função da coerção. O caso mais interessante de coerção é o da comparação entre um valor ou objeto atômico contra um conjunto de objetos. Por exemplo, considerando-se a seguinte consulta:

```
select xml(biblio:{
  selct xml(livro:{ titulo: t })
  from biblio.livro l
  where l.titulo = "Projeto de Banco de Dados" })
```

a condição `l.titulo` é um conjunto, formado por todos os sub-elementos `titulo` de `l`. Nesse caso, Lorel trata essa condição como se de fato fosse:

```
where exists X in l : X = "Projeto de Banco de Dados"
```

evitando, assim, a ocorrência de erros.

## 2.11 Processamento de alternativas

Estruturas alternativas podem ser especificadas em documentos XML, utilizando-se o recurso de escolha na DTD. Assim, por exemplo, na DTD bibliográfica, consultas poderiam ser feitas sobre livros ou artigos. A consulta proposta consiste em retornar o título e editora, caso o elemento seja um livro; ou título e tipo, se o elemento for um artigo.

**YATL**

Em YATL, o operador `match` liga uma série de variáveis, que podem ser utilizadas em outras partes da consulta. A opção é feita pelo símbolo de escolha "`|`".

```
make
  biblio * ( match $o with
             | * livro
             make
               livro [ titulo [ $t ],
                       editora [ $e ] ]
             | * artigo [ titulo [ $t ], editora [ $e ] ]
             make
               artigo [ titulo [ $t ], editora [ $e ] ]
match "www.inf.ufrgs.br/biblio.xml" with biblio($o)
```

**XML-QL**

Consultas aninhadas paralelas são o recurso de XML-QL para o tratamento de alternativas.

```
WHERE <biblio>$b</biblio>
IN "www.inf.ufrgs.br/biblio.xml"
CONSTRUCT <bib> {
  {WHERE <livro>
           <titulo>$t</titulo>
           <editora>$e</editora>
         </livro> in $b
   CONSTRUCT <livro><titulo>$t</titulo>
                    <editora>$e</editora>
             </livro>
  }
  {WHERE <artigo $tp=tipo>
           <titulo>$t</titulo>
         </artigo> in $b
   CONSTRUCT <artigo tipo=$tp>
               <titulo>$t</titulo>
             </artigo> in $b
  }
}
```

## 2.12 Resumo das características das linguagens

A tab. 2.1 resume as características discutidas para cada uma das linguagens. Observa-se na tabela que as linguagens possuem, em geral, recursos similares, com poucas diferenças. Percebe-se também que há uma tendência de convergência entre os paradigmas – bancos de dados e documentos –, no sentido de agregar as funcionalidades mais importantes de ambos em uma única linguagem de consulta. Esse é o objetivo, por exemplo, de Quilt, que combina recursos de todas as outras linguagens apresentadas. Isso fez com que essa linguagem fosse a base para a XQuery [CHA 01], proposta pelo W3C. Embora ainda esteja em fase de definição no momento da conclusão desse trabalho (trata-se de um *Working Draft*, editado em 15 de fevereiro de 2001), percebe-se a clara influência de Quilt, inclusive no objetivo de consultar documentos XML sob os pontos de vista de bancos de dados e de documentos.

TABELA 2.1 – Resumo das características das linguagens de consulta a XML

| Característica | Lorel | XML-QL | YATL | XQL | Quilt |
|---|---|---|---|---|---|
| Seleção e extração | Sim | Sim | Sim | Sim | Sim |
| Reestruturação | CA/SF* | CA/SF | CA | Agrupamento | CA |
| Ordenação | Sim | Sim | Sim | Sim | Sim |
| Expressões de caminho regular | Sim | Sim | Não | Não | Não |
| *Joins* | Sim | Sim | Sim | Sim | Sim |
| Quantificadores | Sim | Não | Não** | Sim*** | Sim |
| Funções de agregação | Sim | Não | Não | só count() | Sim |
| Manut. ordem dos elementos | Não | Não | Não | Sim | Sim |
| Consultas sobre a ordem | Sim | Sim | Sim | Sim | Sim |

* CA=Consulta Aninhada; SF=Skolem Function.

** O quantificador universal pode ser simulado em YATL pela comparação de conjuntos.

*** Aplicação limitada.

Deve-se ressaltar, além disso, que todas as linguagens apresentadas exigem que o usuário conheça a estrutura dos documentos a fim de expressar as consultas. Essa restrição, apesar de poder ser relaxada por alguns recursos, tais como expressões regulares, ainda dificulta a imposição de consultas por usuários que desconhecem a hierarquia dos elementos dentro dos documentos. Além disso, situações em que a mesma informação é apresentada de forma diferente têm que ser tratadas por consultas diferentes, o que dificulta a recuperação da informação. Embora algumas linguagens possuam recursos para o tratamento de alternativas, isso não chega a ser uma solução, uma vez que o usuário deve especificar explicitamente quais são as situações possíveis.

# 3 Linguagens visuais de consulta

Nos últimos anos, os sistemas de informação se propagaram para as mais diversas áreas de atividades humanas. Isso fez com que o acesso a esses sistemas, antes restrito a usuários já acostumados com linguagens de computação, passasse a ser oferecido, cada vez em maior número, a usuários menos experientes ou leigos em informática. Esses usuários, em sua grande maioria, não têm conhecimento nem sequer habilidade suficiente para dominar linguagens de consulta textuais, que é o tipo de interface normalmente oferecido para recuperar informações de bancos de dados.

Em resposta a esse problema, um grande esforço de pesquisa passou a ser dispendido no desenvolvimento de interfaces e/ou linguagens visuais de consulta. De acordo com [AND 96], pode-se visualizar vários aspectos da interface dos sistemas de informação. O primeiro aspecto diz respeito à definição da estrutura do banco de dados, que originou notações gráficas para os esquemas de bancos de dados, sendo que o mais conhecido é, sem dúvida, o modelo Entidade-Relacionamento proposto por Chen [CHE 75].

Outro aspecto relevante são as linguagens para consulta e manipulação do banco de dados, o que levou, entre outros, às linguagens para bancos de dados baseadas no paradigma de manipulação direta de Schneiderman. Sistemas construídos de acordo com esse paradigma permitem ao usuário manipular diretamente os objetos de interesse, na forma de uma representação visual, em oposição aos sistemas que oferecem acesso a esses objetos indiretamente, como, por exemplo, por meio de linguagens textuais. Schneiderman cita as seguintes vantagens das interfaces de manipulação direta:

- representação contínua da realidade de interesse;
- ações físicas ao invés de sintaxe complexa;
- operações rápidas, incrementais e reversíveis, cujo impacto sobre o objeto de interesse é imediatamente visível;
- abordagem em camadas do aprendizado, o que permite o uso com conhecimento mínimo.

O aspecto final se preocupa com a visualização dos resultados produzidos pelas consultas. As pesquisas nesse tópico fazem parte de uma área mais ampla, conhecida como *visualização da informação*, cujo objetivo é desenvolver métodos para apresentar em uma maneira compreensível grandes e complexas estruturas de dados para os usuários de sistemas de informação.

O objetivo deste trabalho se concentra no segundo aspecto, no sentido em que se propõe uma linguagem visual de consulta para bancos de dados XML. A importância da aplicação do paradigma de manipulação direta já era reconhecido por Schneiderman, apud [AND 96], quando afirmou que “representações gráficas podem ser especialmente úteis quando há múltiplos relacionamentos entre objetos, e quando a representação é mais compacta do que o objeto detalhado.”

Da mesma forma, Glinert, apud [AND 96], afirma que “a habilidade do computador de representar em uma forma visível aspectos normalmente abstratos e efêmeros do processo de computação, tais como recusividade, concorrência e a evolução de estruturas de dados tem tido um impacto marcante e positivo tanto na produtividade dos programadores, quanto no grau de satisfação com o ambiente de trabalho.”

Enfatizando especificamente a área de banco de dados, Gerstendörfer e Rohr, apud [AND 96], afirmam que "tarefas estruturais são difíceis de compreender se não forem apresentadas em figuras ou, de modo mais geral, por meio de auxílios visuais. (...) Tarefas com características estruturais são encontradas, por exemplo, em todas as aplicações de bancos de dados."

No que se refere aos paradigmas de representação visual, [CAT 97] as classifica em quatro grupos:

- *baseado em formulários*: os dados são apresentados em forma de tabelas, que se aproveitam da estrutura bi-dimensional da tela do computador. Normalmente, a parte intensional do banco de dados é apresentada ao usuário, que deve completar a consulta preenchendo dados que representam a parte extensional. O exemplo mais marcante desse paradigma de representação é a QBE [ZLO 75];

- *baseado em diagramas*: nesse paradigma, os dados (o esquema do banco de dados) são representados por meio de figuras geométricas simples, tais como retângulos, linhas e círculos. Os exemplos mais utilizados são o diagrama Entidade-Relacionamento e o diagrama de classes e objetos;

- *baseado em ícones*: nesse caso, ícones são utilizados para representar a informação;

- *híbrido*: é uma combinação de qualquer um dos três paradigmas apresentados anteriormente. Normalmente, as combinações mais comuns são formulários e diagramas, diagramas e ícones e formulários, diagramas e ícones.

Neste trabalho, o paradigma de representação visual adotado foi o baseado em formulários, conforme se pode observar no capítulo 5.

Conforme se pode observar, um dos problemas que existem em extrair informações de bancos de dados é que estruturas complexas dificultam a compreensão do usuário. Além disso, as linguagens textuais são difíceis de serem utilizadas por usuários menos experientes. Ambos os problemas aplicam-se também aos dados XML. A estrutura de documentos XML normalmente é complexa, o que dificulta ao usuário a sua compreensão. Linguagens textuais de consulta enfrentam o mesmo problema que em bancos de dados relacionais. Nesse sentido, o desafio de consultar XML motivou, além das linguagens textuais, também o surgimento de linguagens visuais.

Nessa seção, são analisadas três linguagens visuais: a precursora QBE, que se utiliza do paradigma baseado em formulários, além de duas linguagens específicas para XML: XML-GL, que utiliza uma abordagem baseada em diagramas que representam grafos; e Xing, que, apesar de também utilizar uma abordagem baseada em diagramas, os interpreta como documentos.

## 3.1 Query By Example

Uma das primeiras linguagens visuais para consultas a bancos de dados relacionais, e que serviu como inspiração para diversos sistemas interativos de consulta, foi a *Query By Example*, proposta por [ZLO 75] e mais conhecida por sua sigla, QBE. As consultas em QBE, ao contrário das linguagens textuais como SQL, não exigem a construção de nenhuma expressão textual, mas sim a representação de *modelos* ou *templates*, que servem como *exemplos* para filtrar os dados e selecionar aqueles que devem constar do resultado.

Para apresentar a linguagem QBE serão utilizados exemplos de consultas, que demonstram os recursos da linguagem. A base para essas consultas é o esquema de um banco de dados de uma loja de departamentos, composto pelas seguintes tabelas:

- a tabela EMP contém o nome, salário, gerente e departamento de cada empregado;
- a tabela VENDAS é uma listagem dos itens vendidos pelos departamentos;
- a tabela FORNECIMENTO é uma lista dos itens fornecidos pelos fornecedores;
- a tabela TIPO descreve a cor e o tamanho de cada item.

As consultas em QBE são formuladas pelo preenchimento das linhas da tabela com um exemplo de uma possível resposta. Para formular consultas simples, o usuário deve distinguir dois tipos de entidades:

1. O *elemento exemplo* (variável), o qual deve ser sublinhado;
2. o *elemento constante*, que não deve ser sublinhado.

Além disso a função P. possui o sentido de "imprimir" (*print*), ou seja, o usuário insere um P. antes de qualquer dado que ele deseja incluir no resultado.

### 3.1.1 Exemplos

Nessa seção, apresentam-se alguns exemplos de consulta em QBE, que permitem compreender o poder de expressão da linguagem.

| TIPO | ITEM | COR | TAMANHO |
|---|---|---|---|
| | P. CANETA | VERMELHO | |

CONSULTA 3.1: Imprimir os itens da cor "vermelho".

Nesse caso, VERMELHO é um *elemento constante* e, portanto, não deve ser sublinhado. Por outro lado, o elemento sublinhado CANETA é um *elemento exemplo*, pois representa um exemplo de uma possível resposta. O elemento exemplo, nesse caso, está de acordo com o domínio do atributo *item*. Não é, entretanto, obrigatório que seja assim; poder-se-ia substituir a palavra "caneta" por qualquer outra, e até por uma variável X, sem modificar o sentido da consulta. Conforme demonstram os exemplos a seguir, a utilização do elemento exemplo pode, inclusive, ser restrita somente àqueles atributos que ligam duas tabelas.

| TIPO | ITEM | COR | TAMANHO |
|---|---|---|---|
| | TINTA | P. PRETO | |

CONSULTA 3.2: Quais cores de tintas estão disponíveis?

Nesse caso, P. está na coluna COR, porque o que se deseja obter é uma lista de cores. PRETO é o elemento exemplo.

Na consulta a seguir, o usuário deve preencher elementos de duas tabelas: VENDAS e FORNECEDOR.

| VENDAS | DEPT | ITEM |
|---|---|---|
| | P. BRINQUEDOS | BOLA |

| FORNECEDOR | ITEM | FORNECEDOR |
|---|---|---|
| | BOLA | SILVA |

CONSULTA 3.3: Encontrar o(s) departamento(s) que vendem itens fornecidos pelo fornecedor “Silva”.

Nesse caso, o elemento exemplo BOLA foi incluído nas duas tabelas, implicando que se um item é vendido pelo departamento em questão, o mesmo item tem que ser fornecido por “Silva”.

| VENDAS | DEPT | ITEM |
|---|---|---|
| | BRINQUEDOS | BOLA |

| FORNECEDOR | ITEM | FORNECEDOR |
|---|---|---|
| | BOLA | P. SILVA |

CONSULTA 3.4: Encontrar o(s) fornecedor(es) que fornecem itens vendidos pelo departamento de brinquedos.

A consulta 3.4 é praticamente idêntica à anterior, com apenas uma diferença: o elemento que se deseja obter não é mais o departamento, mas sim o fornecedor.

| EMP | NOME | SAL | GER | DEPT |
|---|---|---|---|---|
| | P. SOUZA | P. 2000 | P. SILVA | BRINQUEDOS |

CONSULTA 3.5: Listar os nomes, salários e gerentes dos empregados do departamento de brinquedos.

Nesta consulta, são selecionados para impressão três colunas: o nome, o salário e o nome do gerente da tabela empregado.

Além das operações ilustradas pelas consultas 1 a 5, podem-se formular consultas utilizando os seguintes operadores:

- comparações numéricas: $=$ $\neq$ $<$ $\leq$ $>$ $\geq$;
- operador de negação: $\neg$;
- os operadores JOIN, ALL e ALL D.;
- as funções pré-definidas SUM, COUNT, AVE, MAX, MIN, etc.

| VENDAS | DEPT | ITEM |
|---|---|---|
| | BRINQUEDOS | BOLA |

| FORNECEDOR | ITEM | FORNECEDOR |
|---|---|---|
| | BOLA | SILVA |

| JOIN: VENDAS/FORNECEDOR | DEPT | ITEM | FORNECEDOR |
|---|---|---|---|
| | P. BRINQUEDOS | P. BOLA | P. BIC |

CONSULTA 3.6: Listar todos os departamentos, juntamente com os itens que eles vendem e os fornecedores desses itens.

O operador JOIN indica que será feita uma junção das tabelas VENDAS e FORNECEDOR. O elemento exemplo BOLA aparece em ambas as tabelas, o que denota uma *junção natural* no atributo comum ITEM.

| EMP | NOME | SAL | GER | DEPT |
|---|---|---|---|---|
| | P. SOUZA | P. >2000 | P. SILVA | |
| | SILVA | 2000 | | |

CONSULTA 3.7: Encontrar o nome de todos os empregados que ganham mais do que seus gerentes.

Se SILVA é um exemplo de um gerente e se ele, por exemplo, ganha 2000, então SOUZA é um exemplo de um empregado que ganha mais de 2000 (o que é indicado pelo símbolo $>$) e, portanto, mais do que o seu gerente.

| VENDAS | DEPT | ITEM |
|---|---|---|
| | P. BRINQUEDOS | BOLA |
| | BRINQUEDOS | PATINS |

CONSULTA 3.8: Encontrar os departamentos que vendem bolas *e* patins.

Nessa consulta, o mesmo elemento exemplo BRINQUEDOS aparece em duas linhas, para indicar um *and*, denotando que o mesmo departamento deve vender os dois itens.

| VENDAS | DEPT | ITEM |
|---|---|---|
| | P. BRINQUEDOS | BOLA |
| | P. ESPORTES | PATINS |

CONSULTA 3.9: Encontrar os departamentos que vendem bolas *ou* patins.

Nessa consulta, dois elementos exemplo, BRINQUEDOS e ESPORTES, aparecem para indicar um *or*, porque o departamento que vende o item BOLA não precisa, necessariamente, vender também o item PATINS.

| EMP | NOME | SAL | GER | DEPT |
|---|---|---|---|---|
| | | P.SUM.ALL 2000 | | |

CONSULTA 3.10: Encontrar o total de salários pagos aos empregados do departamento BRINQUEDOS.

Nesse caso, a consulta é executada sobre um *multiset* (também chamado de *bag*) pois, caso dois funcionários possuam o mesmo salário, ambos os valores são computados. Caso a consulta exija a exclusão dos valores duplicados, deve-se utilizar o operador ALL D., onde "D." significa diferente ou distinto (equivalente ao *distinct* de SQL).

| TIPO | ITEM | COR | TAMANHO |
|---|---|---|---|
| | PATINS | P.COUNT.ALL D. PRETO | |

CONSULTA 3.11: Quantas cores diferentes de patins existem na loja?

Nesse exemplo, devem ser eliminados os valores duplicados, sob a pena do valor do resultado não corresponder à realidade. Por isso, utilizou-se o operador ALL D.

| EMP | NOME | SAL | GER | DEPT |
|---|---|---|---|---|
| | | (SUM.ALL 2000) $>$ 22000 | | BRINQUEDOS |

| VENDAS | DEPT | ITEM |
|---|---|---|
| | P. BRINQUEDOS | BOLA |

CONSULTA 3.12: Entre todos os departamentos cujo salário total for maior do que 22 mil, encontre aqueles que vendem bolas.

Nesta consulta, o operador SUM ALL foi utilizado em um teste, para se obter o total dos salários do departamento em questão.

| TIPO | ITEM | COR | TAMANHO |
|---|---|---|---|
| | P. PATINS | $\neg$ VERDE | |

CONSULTA 3.13: Encontrar os itens que não estejam disponíveis na cor verde.

Esta consulta demonstra a utilização do operador de negação, para excluir do resultado os itens da cor verde.

| TIPO | ITEM | COR | TAMANHO |
|---|---|---|---|
| | PATINS | VERDE | |
| | P. $\neg$ PATINS | | |

CONSULTA 3.14: Listar todos os itens, exceto aqueles que estão disponíveis na cor verde.

Esta consulta demonstra outro exemplo do operador de negação. Neste caso, o que se deseja é excluir do resultado os itens que estão disponíveis na cor verde.

## 3.2 XML-GL

XML-GL é uma linguagem proposta por [CER 98], que tem por base um modelo de dados chamado *XML Graphical Data Model* – XML-GDM, utilizado tanto para representar as DTDs quanto os próprios documentos XML, além das expressões de consulta.

A fig. 3.1 apresenta a DTD que servirá como base para os exemplos de expressões da linguagem. Ela representa documentos de um domínio de vendas de livros, com quatro elementos principais: pedido (`order`), pessoa (`person`), livro (`book`) e autor (`author`), cada um com seus respectivos atributos.

Na fig. 3.2, apresenta-se o modelo XML-GDM para a DTD da fig. 3.1. Percebe-se a distinta representação que é dada aos componentes da DTD. Os retângulos indicam elementos não-terminais, os quais se conectam por setas a pequenos círculos vazados, que representam os elementos terminais, ou a outros elementos não-terminais. Os atributos são representados por setas que se originam no elemento ao qual eles pertencem e apontam para pequenos círculos, que são preenchidos quando o tipo do atributo é ID.

```
<!ELEMENT order (shipto, contact?, item+, date)>
<!ATTLIST order number CDATA #REQUIRED>
<!ELEMENT shipto (fulladdress|reference)>
<!ELEMENT reference EMPTY>
<!ATTLIST reference customer IDREF>
<!ELEMENT person (firstname?, lastname, fulladdress)>
<!ATTLIST person id ID>
<!ELEMENT company (#PCDATA)>
<!ELEMENT addressline (#PCDATA)>
<!ELEMENT city (#PCDATA)>
<!ELEMENT date (day, month, year)>
<!ELEMENT day (#PCDATA)>
<!ELEMENT month (#PCDATA)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT item (book, quantity, discount?)>
<!ELEMENT book (isbn, title?, price, author*)>
<!ELEMENT author (firstname, lastname)>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT isbn (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT price (#PCDATA)>
<!ELEMENT quantity (#PCDATA)>
<!ELEMENT discount (#PCDATA)>
```

FIGURA 3.1 – DTD para pedidos de vendas de livros

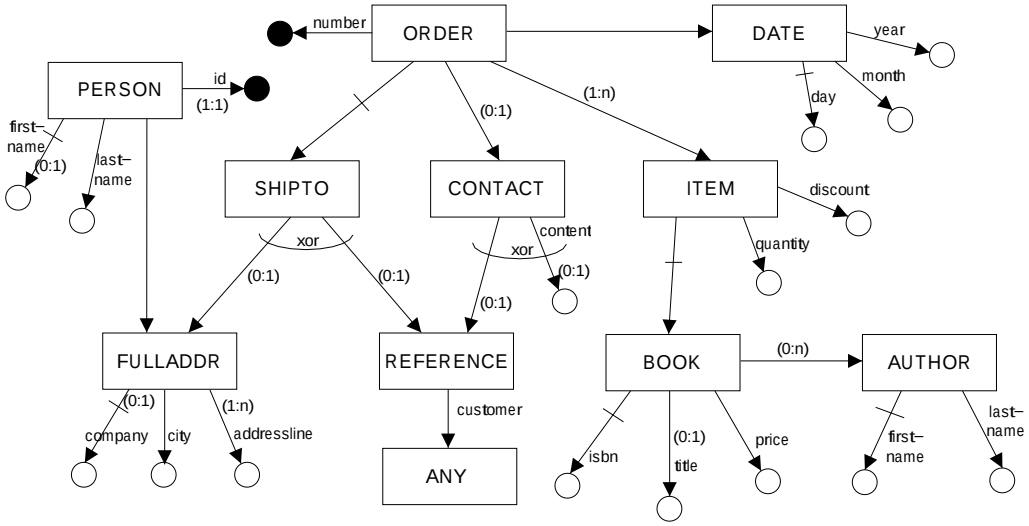


FIGURA 3.2 – Modelo XML-GDM para a DTD da fig. 3.1

Elementos alternativos são ligados por um arco, rotulado pela palavra *xor*. Atributos do tipo IDREF são conectados a um tipo especial de elemento, pré-definido pelo modelo, chamado *ANY*.

A ordem dos elementos é indicada da seguinte maneira: o primeiro sub-elemento tem a seta cortada por uma pequena linha, e os demais são desenhados na direção contrária ao movimento dos ponteiros do relógio.

A cardinalidade dos elementos é representada por números mínimos e máximos colocados ao lado das setas que conectam os elementos e atributos. Assim, uma marcação (0:1) indica que um elemento é opcional, podendo ocorrer no mínimo zero e no máximo uma vez.

As consultas XML-GL podem ser aplicadas tanto a um documento como a um conjunto de documentos, e sempre produzem como resultado documentos XML. Uma consulta XML-QL é composta por quatro partes:

1. a *extract part* identifica o escopo da consulta, indicando tanto os documentos de origem quanto os elementos que o compõem. Comparativamente a SQL, a parte *extract* corresponde à cláusula *from*;

2. a *match part* é opcional, e especifica condições lógicas que os documentos de origem devem satisfazer para se tornarem parte do resultado da consulta. Comparativamente a SQL, a parte *match* corresponde à cláusula *where*;

3. a *clip part* especifica quais são os elementos que constarão do resultado da consulta. Comparativamente a SQL, a parte *clip* corresponde à cláusula *select*;

4. a *construct part* é opcional, e especifica quais os sub-elementos dos elementos extraídos pela parte *match* que serão mantidos no documento final. Comparativamente a SQL, a parte *construct* pode ser vista como uma extensão do comando *create view*, o qual permite a criação de uma nova relação a partir do resultado de uma consulta. A parte *construct* permite a criação de novos elementos, a definição de novos links, e a reestruturação de informações locais a um dado elemento.

Deve-se ressaltar que XML-GL não é simplesmente uma interface gráfica, mas sim uma linguagem de consulta completa, que permite a construção de expressões complexas. As consultas em XML-GL são representadas de forma gráfica, sendo compostas de um par de grafos XML-GDM, dispostos lado a lado e separados por uma linha vertical. O grafo do lado esquerdo representa as partes *extract* e *match*, enquanto que o grafo do lado direito corresponde às partes *clip* e *construct*. Dessa forma, por associação, o grafo do lado direito representa a DTD do resultado, que está sempre presente, mesmo que a consulta seja aplicada para documentos *well-formed*, sem uma DTD. A fig. 3.3 apresenta o tipo mais simples de consulta XML-GL, chamada consulta *extract-clip*.

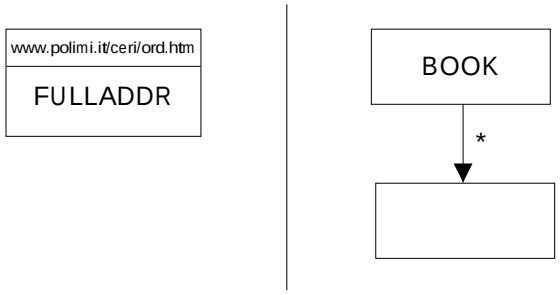


FIGURA 3.3 – Consulta XML-GL do tipo *extract-clip*.

Na consulta da fig. 3.3, no lado esquerdo (a parte *extract*), a origem é o endereço `www.polimi.it/ceri/ord*.xml`. Isso significa que, desse diretório, são selecionados todos os elementos do tipo `book` dos arquivos XML que começam com "ord".

No lado direito da consulta, o retângulo superior, rotulado como `book`, indica o elemento inicial, enquanto que o inferior, sem rótulo, indica qualquer elemento. O símbolo de asterisco que rotula a seta que conecta os dois retângulos indica que todos os elementos, independentemente do nível de aninhamento, serão selecionados.

A expansão do tipo básico de consulta permite que sejam estabelecidos quaisquer tipos de predicados de seleção sobre os elementos dos documentos de origem. Isso também possibilita a produção de qualquer combinação de elementos no resultado. Além disso, a linguagem possui construções para especificação de junções e agregações.

A parte *match* estende o lado esquerdo do grafo da consulta, permitindo expressar uma grande quantidade de predicados de seleção. A condição expressa na parte *match* envolve a aplicação de operadores lógicos para atributos e elementos PCDATA, tais como operadores de comparação ($>$, $<$, $>=$, $<=$, $=$ e $<>$), bem como operadores de *strings* (_ e %). Também podem ser construídas expressões de consulta sobre vários elementos, de forma similar às consultas *select-join* de SQL. Essas características são ilustradas na consulta da fig. 3.4, que busca todos os livros escritos por um autor com o mesmo nome de uma pessoa que comprou livros.

A condição de junção é indicada na parte *match* da consulta pela expansão do grafo do elemento `book`. Isso permite visualizar o elemento `author`, cujo sub-elemento `lastname` é comparado com o sub-elemento de *person* que possui o mesmo nome.

As partes *extract*, *match* e *clip* das consultas XML-GL possuem grande poder de expressão, mas não suportam a reestruturação dos elementos. Caso se queira produzir no resultado estruturas diferentes daquelas presentes nos documentos de origem, deve-se utilizar a cláusula *construct*, que permite construir novos elementos. Novos elementos po-

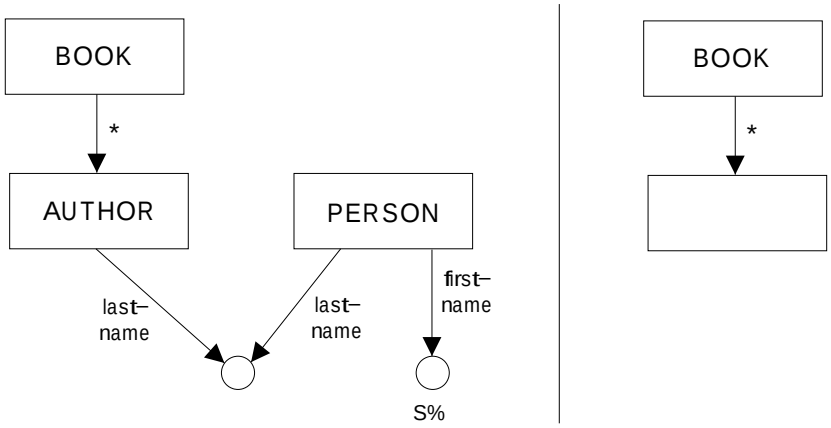


FIGURA 3.4 – Exemplo de consulta *Extract-Match-Clip* com junção.

dem ser construídos de duas formas: embutindo conteúdo extraído em novos elementos, ou estendendo um elemento com informações de outros elementos.

No primeiro caso, o conteúdo extraído via *extract* e *match* é inserido em um novo elemento, que pode ser:

- *elemento construído:* cada elemento extraído pela parte *extract-match* é inserido em uma instância distinta de um novo elemento;
- *lista:* todos os elementos extraídos pela parte *extract-match* são embutidos dentro de um novo elemento;
- *lista agrupada:* ocorrências do mesmo elemento extraído pela parte *extract-match* são embutidos dentro de listas múltiplas definidas por um critério de agrupamento.

As três alternativas são ilustradas na fig. 3.5: o *elemento construído* é representado por um triângulo; a *lista*, por um retângulo; e a *lista agrupada* por um retângulo com linhas horizontais, que se assemelha a uma página de documento.

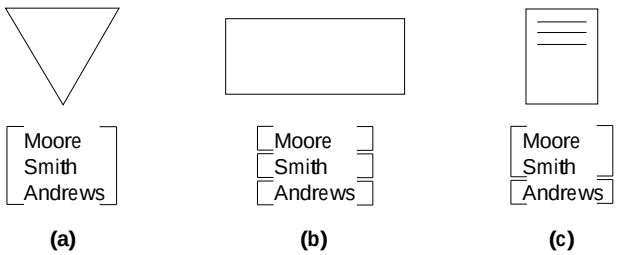


FIGURA 3.5 – Primitivas de construção de XML-GL: (a) elemento; (b) lista; (c) lista agrupada.

Por meio da estensão de elementos, pode-se incluir elementos de um documento em outro, ou estender os elementos de um documento com informações vindas de elementos relacionados dentro do mesmo documento. Considere-se, por exemplo, a consulta da fig. 3.6, que busca os pedidos que contenham um livro cujos primeiro e segundo nomes aparecem também em um elemento do tipo `person`, e produzem um novo elemento `extorder`, onde o endereço é adicionado a cada autor.

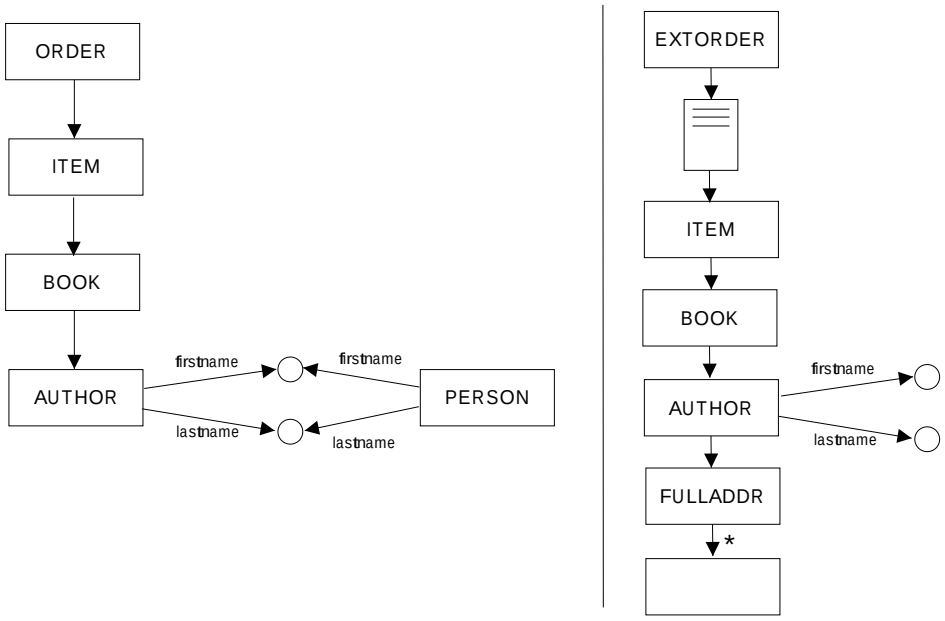


FIGURA 3.6 – Exemplo de extensão de um elemento na parte *construct*

No resultado, um elemento `extorder` é construído para cada grupo de itens pertencentes ao mesmo pedido recuperado na parte *extract-match*. Além disso, cada elemento `author` é estendido com a inclusão do elemento `fulladdress` vindo de `person`.

Operações de aninhamento e *flattening* são representáveis em XML-GL, utilizando-se os relacionamentos hierárquicos entre elementos. O exemplo da fig. 3.7 ilustra essa característica, representando uma consulta que busca os pedidos que possuem elementos `shipto` e `item`, além de um atributo `number`, e produz como resultado uma lista plana de "triplas", cada uma contendo um pedido, com o seu número, informações de envio e o título do livro de um dos seus itens.

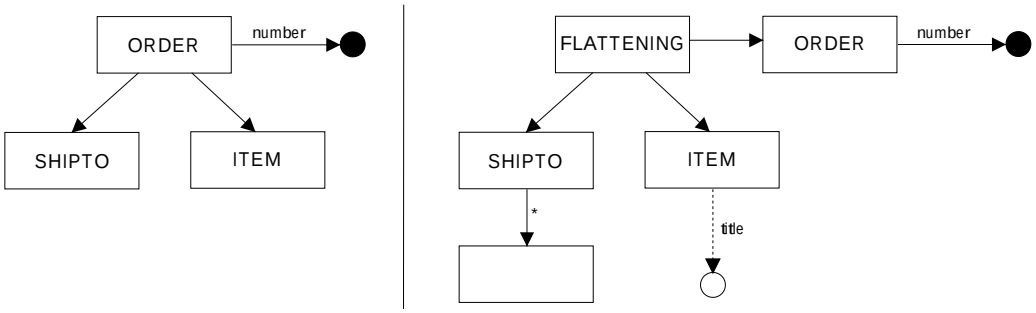


FIGURA 3.7 – Exemplo de *flattening* de um elemento na parte *construct*

## 3.3 Xing

A linguagem XML-GL, apresentada anteriormente, utiliza a metáfora de grafos para representar documentos XML. Já a linguagem *XML in Graphics – Xing* [ERW 00] (pronuncia-se *crossing*) utiliza a metáfora de documentos (mais precisamente, formulários), compostos por campos, sendo que cada um deles possui um *cabeçalho* e um *valor*. O cabeçalho é uma descrição textual do conteúdo do campo, enquanto que o valor pode ser textual ou uma estrutura composta por outros campos.

Uma expressão visual de consulta em Xing é composta por dois padrões: o padrão à esquerda na regra cria ligações que são utilizadas pelo padrão à direita para construir novos documentos. A construção dos exemplos de consulta se fará sobre um documento que contém dados bibliográficos (chamado, daqui por diante, de *bib*), que é apresentado na fig. 3.8.

```
<bib>
  <book year='1998'>
    <title>Concrete Mathematics</title>
    <author>Graham</author>
    <author>Knuth</author>
    <author>Patashnik</author>
  </book>
  <article year='1998'>
    <title>Linear Probing and Graphs</title>
    <author>Knuth</author>
    <journal>Algorithmica</journal>
  </article>
</bib>
```

FIGURA 3.8 – Dados-exemplo para as consultas Xing.

A fig. 3.9 ilustra os dados-exemplo da fig. 3.8 utilizando uma expressão Xing. Cada elemento é representado como um retângulo com os cantos arredondados, com o nome do elemento no cabeçalho, que aparece no canto superior esquerdo. Os nomes dos elementos são escritos em negrito, enquanto que os dos atributos são escritos em caracter regular. A construção "**nome do elemento:** valor" é utilizada como abreviatura para essa notação, quando o valor do campo é textual.

A forma mais simples de consulta consiste em um *padrão de documento* (*document pattern*), que é semelhante a uma expressão, com exceção de que as *tags* dos elementos e o nome dos atributos são utilizados como variáveis. A fig. 3.10 apresenta dois exemplos de padrões de documentos. O primeiro busca todos os livros, enquanto que o segundo busca todas as publicações cujo autor seja "Knuth".

As consultas por padrões de documentos são avaliadas pela aplicação dos padrões sobre o conteúdo dos documentos XML e pelo retorno do sub-documento combinante. No exemplo (a), por exemplo, quando *B* é aplicado sobre *bib*, as *tags* de ambos os elementos mais externos combinam, e a *tag* `book`, que é utilizada aqui como uma simples variável, combina com o primeiro dos dois registros bibliográficos e liga-o a `book`. O resultado é a bibliografia contendo apenas o livro.

Já, no exemplo (b), retornam tanto o livro quanto o artigo, pois o padrão utiliza

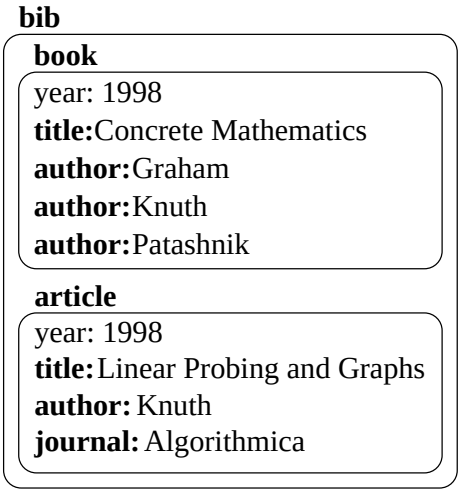


FIGURA 3.9 – Dados-exemplo como uma expressão Xing.

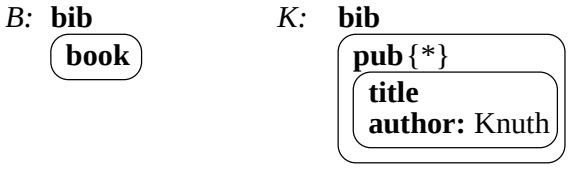


FIGURA 3.10 – Exemplos de consulta por padrão de documento em Xing.

uma expressão regular ({*}) como uma *tag*, cujo resultado é ligado à variável `pub`. A *tag* `title` é utilizada somente como uma variável, combinando, portanto, com qualquer sub-elemento equivalente. A *tag* `author`, entretanto, retorna somente aqueles elementos cujo valor combine com "Knuth".

### 3.3.1 Padrões e ligações

Quatro tipos de padrões podem ser utilizados em consultas Xing:

- *Padrões de texto* são expressões regulares sobre *strings*, ou seja, constantes, tais como "Knuth" ou "1998", e expressões como "(Con|Dis)crete Math*".
- *Padrões de nome.* Existem quatro diferentes versões:

  1. um *padrão de atributo* é dado por um padrão de texto, e é ligado somente àqueles atributos cujos nomes combinam com o padrão;
  2. um *padrão de tag* é também dado por um padrão de texto, e é aplicado a uma lista de elementos. Se algum dos elementos contiver uma *tag* que combine com o padrão, ele é retornado;
  3. um *padrão de alias* consiste de um nome de *tag*, tal como **new**, e um padrão *P*, e é escrito como **new**$\{P\}$. Todos os elementos ligados ao padrão *P* são também ligados a **new**.

4. um *padrão de variável* como *title*, por exemplo, combina tanto com atributos quanto com elementos do mesmo nome. Esse padrão é útil quando a estrutura do documento é desconhecida.

- *Padrões-ou* são úteis para combinar dados que estão em diferentes elementos. Por exemplo, **book** | **article** retornam tanto os dados do elemento *book* quanto os do elemento *article*.

- *Padrões aninhados* são aqueles que consistem de um cabeçalho, o qual é um padrão de *tag*, e um corpo, o qual é dado por uma seqüência não-vazia de padrões arbitrários. Por exemplo, "**author**:Knuth" é um padrão aninhado, assim como os padrões *B* e *K* apresentados na fig. 3.10. Dois tipos especiais de padrões aninhados são os padrões de lista *universal* e *existencial*. Por exemplo, os padrões da fig. 3.11 combinam com toda a lista de autores somente se pelo menos um elemento `author` for igual a "Knuth" (∃) ou somente se todos os elementos `author` forem iguais a "Knuth" (∀).

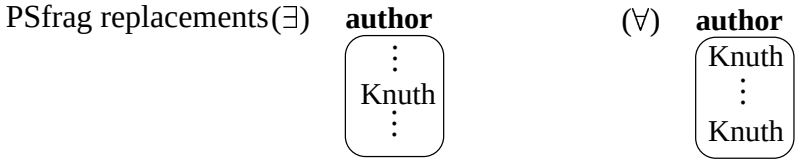


FIGURA 3.11 – Padrões Xing existencial e universal.

- *Padrões profundos.* Ao se prefixar qualquer padrão de nome, padrão-ou ou padrão aninhado *P* com o símbolo "⋱", obter-se-á uma versão *profunda* daquele padrão, que buscará *P* em qualquer ponto do documento, independentemente do nível de profundidade. O padrão da fig. 3.12, por exemplo, recupera todas as atividades de publicação de "Patashnik". Esse padrão é genérico sob três pontos de vista:

  1. ele busca todos os tipos de entradas bibliográficas (artigos, livros, etc.);
  2. ele busca todos os tipos de elementos que possuem "Patashnik" como conteúdo (autor, editor, etc.);
  3. ele busca essa informação em qualquer nível. Por exemplo, publicações que estão aninhadas dentro de elementos de coleção serão encontrados, da mesma forma que aqueles que se encontram no nível mais superior.

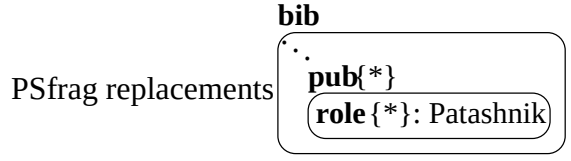


FIGURA 3.12 – Exemplo de padrão *profundo* em Xing.

### 3.3.2 Regras e consultas básicas

Conforme apresentado anteriormente, a forma mais simples de uma consulta consiste em um padrão de documento, como *B* e *K* na fig. 3.10. Na verdade, cada um desses padrões utilizados como uma consulta é somente uma abreviatura para uma *regra de documento*, que possui a forma $P \Rightarrow R$, onde *P* e *R* são padrões, sendo que o primeiro é chamado *padrão argumento*, enquanto que o segundo é o *padrão resultado*. Assim, um padrão simplesmente escrito *B* é somente um atalho para a regra $B \Rightarrow B$.

Para produzir o resultado da aplicação dos padrões, algumas regras devem ser consideradas:

1. a semântica das consultas foi definida para incluir, por *default*, todos os atributos, em virtude de que eles são mais ligados aos elementos do que os sub-elementos o são. Pode-se suprimir a apresentação de todos os atributos colocando-se "-" após o nome do elemento. A seleção de atributos a serem mostrados pode ser feita pelo uso de padrões de atributo no corpo de um padrão aninhado;

2. uma variável é sempre amarrada a um elemento completo, e se essa variável for utilizada sem restrições adicionais, o elemento é mostrado com todos os seus sub-elementos no resultado. Entretanto, se um elemento de um padrão resultado contiver sub-elementos, somente eles serão mostrados no resultado. Assim, para mostrar as entradas bibliográficas completas, pode-se simplesmente utilizar um padrão de resultado separado, contendo somente a variável **pub**, conforme ilustrado na fig. 3.13.

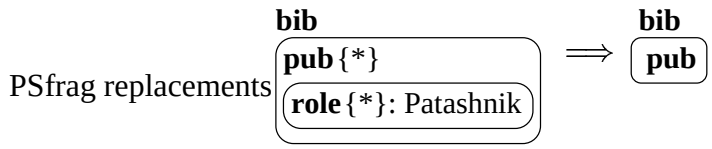


FIGURA 3.13 – Exemplo de padrão resultado em Xing.

Além das características apresentadas, a combinação de padrões Xing permite também a reestruturação dos documentos por agrupamento, além da expressão de junções e acompanhamento de referências.

## 3.4 Conclusões

As linguagens XML-GL e Xing apresentam, embora com paradigmas diferentes, praticamente as mesmas características. Em termos paradigma, XML-GL apresenta os documentos e as expressões de consulta como grafos, enquanto que Xing os representa utilizando a metáfora de documentos.

As linguagens visuais para XML possuem maior usabilidade que as linguagens textuais, por serem mais intuitivas. Deve-se ressaltar, entretanto, que elas apresentam o mesmo problema que as linguagens textuais: o usuário é obrigado a conhecer a estrutura dos documentos para tirar o máximo proveito dos recursos da linguagem, e, caso a

mesma informação seja representada de formas diferentes, devem ser especificadas duas consultas diferentes, uma para cada tipo de representação.

Além das linguagens apresentadas nesse capítulo, cabe destacar também a *Querying semistructured data By Example* – QSByE [EVA 01], que faz parte do projeto *Data Extraction By Example* – DEByE [LAE 99]. A interface da QSByE é baseada na linguagem QBE, e utiliza tabelas aninhadas para representar e consultar dados hierárquicos. A interface QSByE possui um módulo que extrai o esquema de um documento, apresentado-o ao usuário para facilitar a elaboração da consulta. Isso, apesar de ajudar o usuário na formulação da consulta, não resolve o problema da mesma informação ser representada de formas diferentes nos documentos.

# 4 Ontologias e XML

A linguagem XML permite que a mesma informação possa ser representada de formas diferentes. Por exemplo, em um domínio de informações acadêmico, no qual deseja-se representar a relação existente entre professores e disciplinas, pode-se, em uma instância XML, representar cada professor seguido de suas disciplinas, enquanto que, em outra instância XML, cada disciplina é seguida de seus professores. Essa multiplicidade de representações advém da estrutura hierárquica de XML, na qual, dados dois elementos de informação relacionados entre si, é necessário sempre indicar um deles como sendo hierarquicamente superior ao outro.

A representação dos mesmos dados de maneiras diferentes traz dificuldades para muitos tipos de aplicação, tais como consultas e integração de dados. Por exemplo, caso as informações sobre professores e disciplinas sejam armazenadas em documentos XML com estruturas diferentes, é necessário aplicar consultas com sintaxes diferentes para obter as informações de um ou outro documento.

Uma solução para esse problema consiste em associar um modelo conceitual a conjuntos de documentos XML. Um modelo conceitual descreve *quais* os conceitos que existem em um domínio de aplicação e *como* esses conceitos se relacionam. Dessa forma, quando associado a um conjunto de documentos XML, um modelo conceitual serve como uma representação abstrata dos elementos de informação presentes naquele conjunto de documentos. Assim, pode-se expressar consultas sobre o conjunto de documentos utilizando o modelo conceitual para extrair informações sem que se conheça com exatidão a estrutura hierárquica dos documentos. Neste trabalho, o modelo conceitual para conjuntos de documentos XML é representado por uma *ontologia*, que descreve os conceitos e as relações existentes entre eles em um determinado domínio de problema.

O termo *ontologia*, originário da área da filosofia, começou a aparecer no início da década de 1990 na área de IA, estendendo-se, a partir daí, para diversas outras áreas da computação que trabalham com conceituação e modelos conceituais. A definição mais citada de ontologia no campo de IA é a de Gruber [GRU 93]: "uma ontologia é uma especificação explícita de uma conceitualização". Em função de que essa definição permite uma ampla interpretação, principalmente sobre a abrangência do termo *especificação*, o próprio autor, citado por [GUA 95], ampliou-a, definindo ontologias como "acordos acerca de conceitualizações *compartilhadas*. Conceitualizações compartilhadas incluem estruturas conceituais para modelar conhecimento de domínio; protocolos de conteúdo específico para comunicação entre agentes inter-operantes; e acordos acerca da representação de teorias de domínio particulares. No contexto do compartilhamento de conhecimentos, ontologias são especificadas na forma de definições de um vocabulário representativo. Um caso muito simples seria uma hierarquia de tipos, especificando classes e seus relacionamentos de dependência. Esquemas de bancos de dados também funcionam como ontologias, especificando as relações que podem existir em algum banco de dados compartilhado e as restrições de integridade que devem existir para eles." Um estudo mais abrangente dos conceitos de ontologias pode ser encontrado em [MEL 00].

Para associar ontologias a documentos XML, entretanto, deve-se especificar qual a relação que existe entre uma ontologia e uma (ou mais) classes de documentos XML, representadas pelos seus esquemas. Dentre os estudos já desenvolvidos com o objetivo de se utilizar ontologias para recuperar informações de fontes XML, cabe destacar o de [ERD 00], em que é apresentado um processo de geração de uma única *Document Ty-*

*pe Definition* – DTD para cada ontologia dada; e o de [DOR 00], no qual se descreve um algoritmo para gerar todas as DTDs possíveis de serem obtidas a partir de uma ontologia.

Neste capítulo, inicialmente, apresenta-se a *Ontology Inference Layer*, uma linguagem para representação de ontologias; a seguir, apresentam-se os modelos formais, tanto de ontologias quanto de esquemas XML.

## 4.1 Ontology Inference Layer – OIL

Dentre os formalismos mais utilizados para a representação de ontologias estão o *Knowledge Interchange Format* – KIF e a lógica de descrição [MEL 00]. Apesar de logicamente abrangentes, permitindo a representação de expressões lógicas complexas, essas abordagens possuem dois pontos fracos:

a) a notação não foi projetada para ser lida por pessoas, mas sim processada por computador;

b) as linguagens não são indicadas para o intercâmbio de informações.

Para atender a essas necessidades, foram propostas linguagens de representação de ontologias com sintaxe XML, em função de que esse formato é adequado tanto à leitura por pessoas quanto ao processamento por computador, ao mesmo tempo em que apresenta como uma das suas principais vantagens o fato de ser adequada para o intercâmbio de informações. Dentre as propostas para a utilização de XML como linguagem de representação de ontologias estão a *XML-Based Ontology Exchange Language* – XOL [KAR 99] e a *Ontology Interface Layer* – OIL [HOR 00]. Semelhantes em muitos aspectos, as duas linguagens diferem em termos de suporte computacional, característica na qual OIL leva vantagem.

A *Ontology Inference Layer* – OIL – é uma linguagem para representação de ontologias proposta por Horrocks et al. [HOR 00], com o objetivo de:

- fornecer a maior parte das primitivas de modelagem utilizadas em ontologias baseadas em *frames*;
- possuir semânticas simples, claras e bem definidas, baseadas na lógica de descrição (*Description Logic* – DL);
- poder receber suporte automatizado de raciocínio (*reasoning*), como consistência de classe e verificação de dependências.

Para atingir esses objetivos, OIL tem por base três áreas já bem estabelecidas, conforme se observa na figura 4.1.

- *Lógica de descrição:* a DL descreve o conhecimento em termos de conceitos e restrições de papéis, que são utilizadas para derivar automaticamente taxonomias de classificação. A DL, também conhecida como *lógica de terminologia*, forma uma classe importante e poderosa de linguagens de representação do conhecimento baseadas em lógica. OIL herda das linguagens de lógica de descrição a *semântica formal* e o *suporte eficiente ao raciocínio*.

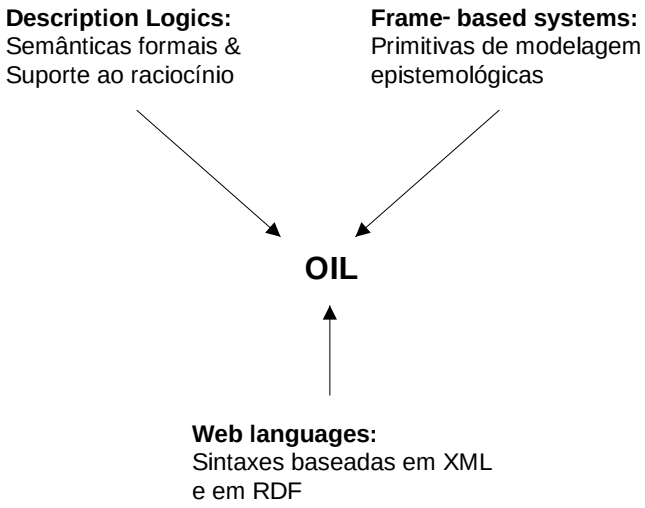


FIGURA 4.1 – Origens da OIL

- *Sistemas baseados em* frames*:* as primitivas de modelagem principais das lógicas de predicado são os predicados. Já as abordagens baseadas em *frames* e orientadas a objetos tomam um ponto de vista diferente: suas primitivas principais são classes (ou seja, *frames*), com certas propriedades chamadas *atributos*. Esses atributos não possuem um escopo global, portanto, são aplicáveis somente àquelas classes para as quais eles forem definidos e o "mesmo" atributo (ou seja, o mesmo *nome* de atributo) pode ser associado a diversas restrições de valor quando definido para classes diferentes. Nesse contexto, um *frame* fornece um certo contexto para modelar um aspecto de um domínio. OIL incorpora as *primitivas de modelagem essenciais* de sistemas baseados em *frames*, ou seja, OIL tem por base a noção de conceitos e a conseqüente definição de suas superclasses e atributos.

- *Padrões para a Web:* XML e RDF [LAS 99], juntamente com outras linguagens derivadas ou correlatas, estão se tornando padrão para a representação de conhecimento na Web, sendo que um de seus principais objetivos é suportar o intercâmbio de dados. Isso faz com que elas possam ser vistas como boas escolhas também para representar e favorecer o intercâmbio de ontologias. OIL estende uma linguagem XML para ontologias chamada *XML-based Ontology Exchange Language* – XOL [KAR 99], incluindo um suporte mais completo para a definição de classes, entre outras características. Ontologias OIL também podem ser representadas em RDF e RDFSchema [BRI 99], que são linguagens para descrição de recursos na Web.

De acordo com os autores, OIL estende XOL, fornecendo uma série de construções comuns em ontologias definidas em lógica de descrição, e que não são possíveis de se expressar em XOL. Além disso, OIL possui ferramentas de suporte que facilitam a edição e validação de ontologias, sendo que, até o momento, não se pode dizer o mesmo de XOL.

### 4.1.1 A linguagem OIL

A sintaxe XML de OIL, apesar de ser adequada para o intercâmbio de dados, é de difícil leitura, em função das *tags* dos elementos "poluírem" o documento. Assim, os

exemplos são apresentados em uma linguagem "pseudo-XML", onde as *tags* são representadas por texto em negrito, o agrupamento de sub-elementos é indicado por indentação e as *tags* de fechamento são omitidas. A notação para representar a ocorrência de elementos é a mesma utilizada nas DTDs, ou seja, elementos opcionais são indicados por **elemento**$^?$, os que podem ser repetidos uma ou mais vezes são indicados por **elemento**$^+$, e os que podem ser repetidos zero ou mais vezes por **elemento**$^*$.

Para descrever ontologias escritas em OIL, deve-se distinguir três níveis:

- O *nível de objeto*, onde instâncias concretas da ontologia são descritas. Neste trabalho, esse nível não é considerado, pois a linguagem é utilizada para representar a ontologia como esquema, e não como instâncias dos conceitos definidos por ela. As instâncias estão representadas em XML, em um banco de documentos.

- O *primeiro meta-nível*, onde as definições reais da ontologia são fornecidas, ou seja, a terminologia que pode ser instanciada no nível de objetos. É nesse nível que se concentram as principais contribuições de OIL;

- O *segundo meta-nível*, onde se definem informações sobre uma ontologia, tais como autor, data de criação, assunto, etc. Na representação desse nível, utiliza-se o *Dublin Core Metadata Element Set*, versão 1.1 [DUB 99], que é um conjunto de elementos de meta-dados cujo objetivo é facilitar a descoberta de recursos eletrônicos.

OIL preocupa-se em definir o primeiro e o segundo meta-níveis. O primeiro meta-nível é chamado de *definição da ontologia* (*ontology definition*), enquanto que o segundo é o *recipiente da ontologia* (*ontology container*).

### 4.1.2 Definição de ontologias em OIL

Além das definições do *ontology container*, uma ontologia consiste de um conjunto de outras definições:

**import**$^?$ é uma lista de referências para outros módulos OIL que são incluídos na ontologia. Cada referência consiste de um URI que especifica onde se encontra o módulo a ser importado, por exemplo, `http://www.ontosRus.com/animals/jungle.onto`. O mecanismo de importação de definições é similar ao de XML Schema [BEE 99], inclusive no sentido em que os nomes de conceitos são diferenciados pelo prefixo que indica a sua origem;

**rule-base**$^?$ é uma lista de regras (também chamadas de *axiomas* ou *restrições globais*) que se aplicam à ontologia. Até o momento, não existe uma definição da estrutura dessas regras, e elas não possuem significado semântico;

**definition**$^*$ consiste em zero ou mais definições de classes (**class-def**) e definições de *slots* (**slot-def**), cujas estruturas serão descritas a seguir.

Uma definição de classe (**class-def**) associa um nome de classe com uma descrição, e consiste dos seguintes componentes:

**type**$^?$ o tipo da definição pode ser **primitive** ou **defined**; caso seja omitido, será assumido como sendo **primitive**. Quando uma classe é **primitive**, sua definição (ou seja, a combinação dos seus componentes **subclass-of** e **slot-constraint**) é assumida como

sendo uma condição necessária mas não suficiente para ser membro da classe. Por exemplo, se uma classe elefante for definida como subclasse de animal, com um *slot constraint* dizendo que a cor-da-pele deve ser cinza, então todas as instâncias de elefante devem, necessariamente, ser animais com pele da cor cinza, mas podem haver animais de pele cor cinza que não são elefantes. Quando uma classe é defined, entretanto, a sua definição é tomada como uma condição necessária e suficiente para ser membro da classe. Se, por exemplo, uma classe carnívoro é dita defined, sendo uma sub-classe de animal com um *slot constraint* dizendo que ela come carne, então todas as instâncias de carnívoro devem necessariamente ser animais que comem carne, e todo animal que come carne também é uma instância de carnívoro.

**name** é um *string* que define o nome da classe;

**documentation**? é uma string que contém uma documentação descrevendo a classe;

**subclass-of**? é uma lista de uma ou mais **class-expressions**, cuja estrutura será descrita abaixo. A classe que está sendo definida nessa **class-def** deve ser uma sub-classe de cada uma das expressões de classe na lista;

**slot-constraint*** são zero ou mais *slot constraints*, um tipo especial de expressão de classe, cuja estrutura será descrita a seguir. A classe que está sendo definida nessa **class-def** deve ser uma sub-classe de cada **slot-constraint**.

Uma expressão de classe (**class-expression**) pode ser um nome de classe, um **slot-constraint**, ou uma combinação booleana de expressões de classe utilizando os operadores AND, OR ou NOT.

**AND:** uma lista de duas ou mais expressões de classe que são tratadas como uma conjunção. Por exemplo, Meat AND Fish define uma classe cujas instâncias são todos aqueles indivíduos que são instâncias de ambas as classes Meat e Fish.

**OR:** uma lista de duas ou mais expressões de classe que são tratadas como uma disjunção. Por exemplo, Meat OR Fish define uma classe cujas instâncias são todos aqueles indivíduos que são instâncias ou da classe Meat ou da classe Fish.

**NOT:** uma expressão que aceita como parâmetro uma única expressão de classe que será negada. Por exemplo, NOT Meat define uma classe cujas instâncias são todos aqueles indivíduos que não são instâncias da classe Meat.

Um **slot-constraint** é uma lista de uma ou mais restrições aplicadas a um *slot* (também chamado de *atributo* ou *papel*[1]). Um *slot* é um relacionamento binário, entretanto, um **slot-constraint** é, de fato, uma definição de classe cujas instâncias são aqueles indivíduos que satisfazem as restrições. Os componentes de um **slot-constraint** são:

**name** o nome do *slot*. O *slot* pode ou não ser definido na ontologia. Caso não seja definido, assume-se que seja um relacionamento binário sem restrições globais, ou seja, qualquer par de indivíduos poderia ser uma instância do *slot*;

[1]Em função disso, ao longo do texto, os termos *slot* e *atributo* serão utilizados como sinônimos.

**has-value**? é uma lista de uma ou mais **class-expressions**. Cada instância da classe definida pelo *slot-constraint* deve estar relacionada por meio do relacionamento do *slot* a uma instância de cada **class-expressions** na lista. A cláusula **has-value** corresponde ao *quantificador existencial* da lógica de predicados: para cada instância da classe, existe pelo menos um valor para esse *slot* que preenche as restrições de valor;

**value-type**? é uma lista de uma ou mais **class-expressions**. Se uma instância da classe definida pela *slot-constraint* estiver relacionada, por meio do relacionamento do *slot*, a algum indivíduo *x*, então *x* deve ser uma instância de cada **class-expression** na lista. A cláusula **value-type** corresponde ao *quantificador universal* (*para todos*) da lógica de predicados: para cada instância da classe, cada valor desse *slot* deve preencher as restrições de valor.

**max-cardinality**? um número não negativo seguido de uma **class-expression**. Uma instância da classe definida pela *slot-constraint* pode se relacionar a no máximo *n* instâncias distintas da classe indicada pela **class-expression** por meio do relacionamento do *slot*. Caso a **class-expression** seja omitida, a classe pode se relacionar com no máximo *n* indivíduos distintos, independentemente da classe.

**min-cardinality**? um número não negativo seguido de uma **class-expression**. Uma instância da classe definida pela *slot-constraint* pode se relacionar a no mínimo *n* instâncias distintas da classe indicada pela **class-expression** por meio do relacionamento do *slot*. Caso a **class-expression** seja omitida, a classe pode se relacionar com no mínimo *n* indivíduos distintos, independentemente da classe.

**cardinality**? um número não negativo seguido de uma **class-expression**. Essa definição é somente um atalho para a combinação de **min-cardinatity** e **max-cardinality**.

Uma definição de *slot* (**slot-def**) associa um nome de *slot* a uma descrição de *slot*. Uma descrição de *slot* especifica restrições globais para o relacionamento do *slot*, como, por exemplo, que trata-se de um relacionamento transitivo. Um **slot-def** consiste dos seguintes componentes:

**name** uma *string* que define o nome do *slot*;

**documentation**? é alguma documentação descrevendo o *slot*;

**subslot-of**? é uma lista de um ou mais *slots*. O *slot* que está sendo definido deve ser um *sub-slot* de cada um dos *slots* da lista;

**domain**? é uma lista de uma ou mais **class-expressions**. Se o par $(x,y)$ é uma instância do relacionamento do *slot*, então *x* deve ser uma instância de cada **class-expression** da lista;

**range**? é uma lista de uma ou mais **class-expressions**. Se o par $(x,y)$ é uma instância do relacionamento do *slot*, então *y* deve ser uma instância de cada **class-expression** da lista;

**inverse**? é o nome de um *slot S* que é o inverso do *slot* que está sendo definido. Se o par $(x,y)$ é uma instância do *slot S*, então $(y,x)$ deve ser uma instância do *slot* que está sendo definido;

**properties**? é uma lista de uma ou mais propriedades do *slot*. As propriedades válidas são:

**transitive** define que o *slot* é *transitivo*, ou seja, se $(x,y)$ e $(y,z)$ são instâncias do *slot*, então $(x,z)$ deve também ser uma instância do *slot*;

**symmetric** define que o *slot* é *simétrico*, ou seja, se $(x,y)$ é uma instância do *slot*, então $(y,x)$ também deve ser uma instância do *slot*.

## 4.2 Modelo formal de uma ontologia OIL

Uma ontologia OIL pode ser representada como um grafo direcionado, no qual as arestas são rotuladas pelas palavras reservadas da linguagem e os vértices pelos conceitos do domínio do problema definidos na ontologia. A fig. 4.2 apresenta um fragmento de uma ontologia OIL, representada com a sintaxe "pseudo-xml" da linguagem, no qual são definidas as classes *Pessoa*, *Professor*, *Disciplina* e *Curso*, com associações entre elas. A seção *ontology-container* não é representada, porque ela trata apenas de metadados sobre a ontologia, não influenciando a definição dos conceitos. Em virtude disso, a partir desse ponto do texto, a palavra "ontologia" será utilizada para representar a seção *ontology-definition* de OIL. O exemplo é apresentado na sintaxe "pseudo-xml" de OIL, que é mais legível do que as sintaxes XML e RDF da linguagem.

```
begin-ontology
   ...
   ontology-definitions
      slot-def ensina
         domain Professor
         range Disciplina
         inverse ensinadaPor
      slot-def oferece
         domain Curso
         range Disciplina
         inverse oferecidaPor
      class-def Pessoa
         slot-constraint nome
            value-type STRING
         slot-constraint endereço
            value-type STRING
      class-def Professor
         subclass-of Pessoa
         slot-constraint ensina
            min-cardinality 1 Disciplina
         slot-constraint colabora
            value-type Professor
      class-def Disciplina
         slot-constraint nome
            value-type STRING
         slot-constraint ensinadaPor
            min-cardinality 1 Professor
         slot-constraint oferecidaPor
            min-cardinality 1 Curso
      class-def Curso
         slot-constraint nome
            value-type STRING
         slot-constraint oferece
            min-cardinality 1 Disciplina
      equivalent Professor Mestre
end-ontology
```

FIGURA 4.2 – Exemplo de ontologia definida em OIL.

Apresenta-se, na fig. 4.3, o grafo que representa a ontologia da fig. 4.2, onde se pode observar a existência de um vértice rotulado como *ontology-definition*, o qual representa a ontologia como um todo e que é a origem dos demais vértices. Esses, por sua vez, são rotulados com os conceitos definidos na ontologia, e são ligados por arestas, cujos rótulos são as palavras reservadas de OIL. Assim, dado um grafo, pode-se obter a ontologia em OIL percorrendo seus vértices a partir daquele rotulado com *ontology-definition*.

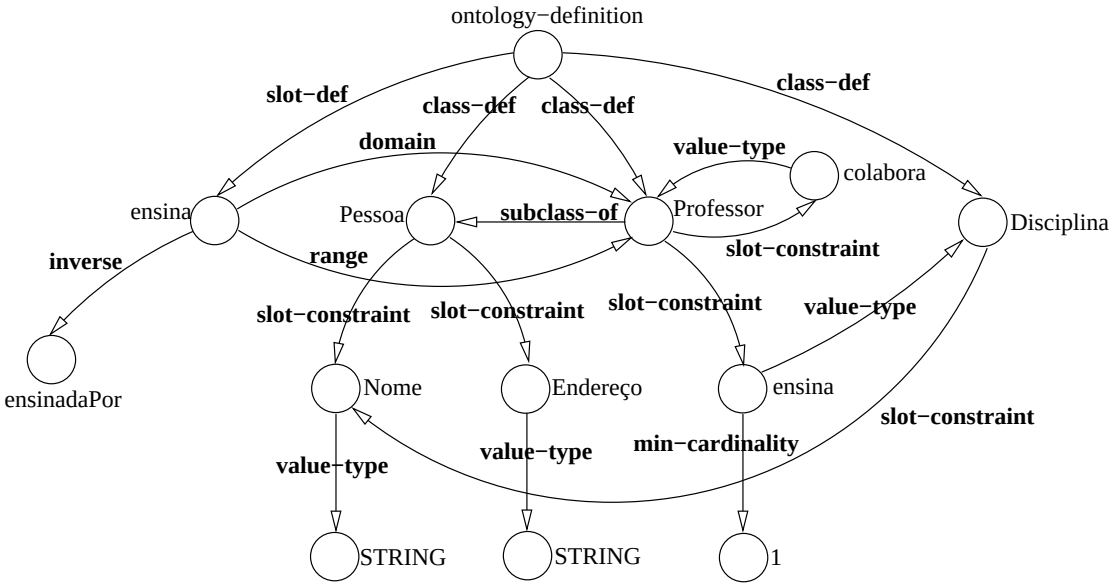


FIGURA 4.3 – Grafo que representa a ontologia OIL da figura 4.2.

Uma vez compreendida a relação entre uma ontologia em OIL e o grafo que a representa, pode-se formalizar essa definição, considerando-se que:

- $\mathcal{L}_V$ é o conjunto formado pelos nomes de todos os conceitos do domínio do problema definidos na ontologia;
- $\mathcal{L}_A$ é o conjunto formado por todas as palavras reservadas de OIL.
- o *nível* de um vértice *v* é o menor número de arestas que devem ser percorridas para se atingir *v*, partindo-se do vértice rotulado com *ontology-definition*.

**Definição 1 (Ontologia)** *uma ontologia em OIL pode ser representada como um grafo direcionado,* $G = (V, A, o, l_V, l_A)$*, onde:*

- $\{v_1, v_2, \ldots, v_n\} \in V, n \in \mathbb{N}$ *é um conjunto de vértices;*
- $A \subset V \times V$ *é um conjunto de arestas que formam o grafo;*
- $A_H \subset A$ *é um conjunto de arestas de herança, ou seja, que ligam uma classe à sua superclasse;*
- $A_E \subset A$ *é um conjunto de arestas de equivalência, ou seja, que ligam conceitos que são sinônimos na ontologia;*
- $o \in V$ *é o vértice da ontologia, a partir do qual se pode acessar quaisquer outros vértices;*
- $l_V : V \rightarrow \mathcal{L}_V$ *é uma função que associa um rótulo em* $\mathcal{L}_V$ *a cada vértice;*
- $l_A : A \rightarrow \mathcal{L}_A$ *é uma função que associa um rótulo em* $\mathcal{L}_A$ *a cada aresta;*
- $n : V \rightarrow \mathbb{N}$ *é uma função que retorna o nível do vértice* $v \in V$*.*

Com base na definição da ontologia e nos seguintes predicados:

- converge$(a,v)$, que determina que uma aresta $a \in A$ converge para um vértice $v \in V$;
- diverge$(v,a)$, que determina que uma aresta $a \in A$ diverge de um vértice $v \in V$;
- conecta$(a,v_1,v_2)$, que determina que uma aresta $a \in A$ conecta dois vértices, $v_1$ e $v_2$;
- descendentes$(v)$, que retorna todos os vértices que descendem de um vértice $v \in V$;
- convergentes$(v)$, que retorna todos os vértices que são ascendentes a um vértice $v \in V$;
- equivalentes$(v)$, que retorna todos os vértices que são equivalentes a um vértice $v \in V$;

podem-se especificar as restrições que devem ser obedecidas para que um grafo $G$ seja considerado uma representação de uma ontologia OIL. São apresentadas apenas algumas restrições, uma vez que o objetivo desse trabalho não é produzir uma especificação formal de toda a linguagem OIL, mas apenas completa o suficiente para a compreensão do conteúdo que segue.

i) $l_V(o) =$ ontology-definition
(O rótulo do vértice que representa a ontologia deve ser *ontology-definition*);

ii) $\forall v \in V : n(v) = 1 \Rightarrow \exists a \in A :$ converge$(a,v) \wedge$
$l_A(a) \in \{$slot-def, class-def, disjoint, covered, disjoint-covered$\}$
(Os vértices associados diretamente ao vértice da ontologia devem ser de um dos seguintes tipos: *slot-def, class-def, disjoint, covered, disjoint-covered*);

iii) $\forall v \in V, a_1 \in A :$ diverge$(a_1,v) \wedge$
$l_A \in \{$subslot-of, domain, range, inverse, properties transitive, properties symmetric, properties functional$\} \Rightarrow \exists a_2 \in A :$ converge$(a_2,v) \wedge l_A(a_2) =$ slot-def
(Os vértices dos quais divergem arestas rotuladas com *subslot-of, domain, range, inverse, properties transitive, properties symmetric, properties functional* devem ter uma aresta que converge para si, rotulada como *slot-def*);

iv) $\forall v_1, v_2 \in V, a \in A : l(a) = equivalent \Rightarrow$ (descendentes$(v_1) =$ descendentes$(v_2)) \wedge$
(ascendentes$(v_1) =$ ascendentes$(v_2)$)
(Os vértices descendentes e ascendentes de duas classes definidas como equivalentes são os mesmos).

v) $\forall v_1, v_2 \in V, a \in A : l(a) =$ subclass-of $\Rightarrow$ descendentes$(v_2) =$ descendentes$(v_1) \cup$
descendentes$(v_2)$
(O conjunto de vértices descendentes de uma classe é o resultado da união do conjunto dos seus descendentes com o conjunto das classes das quais ela descende)

## 4.3 Modelo formal de um esquema XML

A forma como os elementos XML são organizados dentro do documento é definida por meio de *esquemas*.[2]

Esquemas XML são representações hierárquicas de informações. Em função disso, eles podem ser representados por meio de árvores direcionadas rotuladas. Considerando-se, por exemplo, o esquema XML da fig. 4.4, pode-se representá-lo por meio da árvore ilustrada na fig. 4.5. Observa-se que os nomes dos elementos XML são utilizados como rótulos dos vértices do grafo, e que a estrutura hierárquica dos elementos é representada pelo sentido das arestas que associam os vértices.

```
<!ELEMENT Professor (Nome, Endereço, Disciplina+)>
<!ELEMENT Disciplina (Nome)>
<!ELEMENT Nome (#PCDATA)>
<!ELEMENT Endereço (#PCDATA)>
```

FIGURA 4.4 – Esquema XML exemplo.

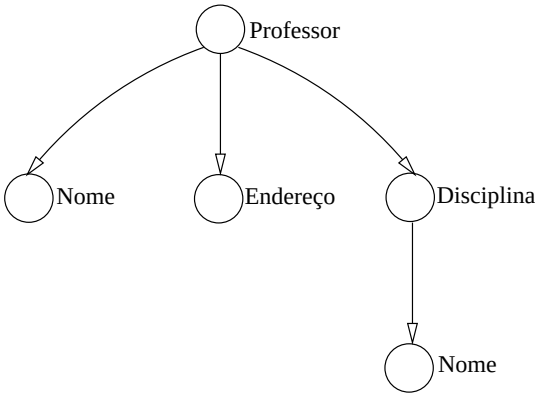


FIGURA 4.5 – Grafo para o esquema XML da figura 4.4.

Considerando-se $\mathcal{L}'$ o conjunto de rótulos formado pelo nome das *tags* dos elementos XML, pode-se definir um esquema XML como:

**Definição 2 (Esquema XML)** *um esquema XML é uma árvore direcionada rotulada, indicada por* $G' = (V', A', r, l)$*, onde:*

- $V'$ *é um conjunto de vértices, sendo que* $V'_N \subset V'$ *é o conjunto dos vértices não terminais da árvore, e* $V'_T \subset V'$ *é o conjunto dos vértices terminais da árvore;*
- $A' \subset V' \times V'$ *é o conjunto das arestas que formam a árvore;*
- $r$ *é a raiz da árvore, de modo que qualquer vértice* $v \in V'$ *pode ser atingido a partir de* $r$*;*

[2]Existem duas propostas de linguagens para definir esquemas em XML: a *Document Type Definition* – DTD, e a *XML Schema* [FAL 01]. A palavra *esquema*, neste trabalho, é utilizada sem fazer referência a nenhuma proposta em particular.

```
<professor>
  <nome>João da Silva</nome>
  <disciplina>
    <nome>Banco de Dados I</nome>
  </disciplina>
  <disciplina>
    <nome>Banco de Dados II</nome>
  </disciplina>
</professor>
```

```
<disciplina>
  <nome>Banco de Dados I</nome>
  <professor>
    <nome>João da Silva</nome>
  </professor>
</disciplina>
<disciplina>
  <nome>Banco de Dados II</nome>
  <professor>
    <nome>João da Silva</nome>
  </professor>
</disciplina>
```

FIGURA 4.6 – Duas representações para informações sobre professores e disciplinas.

- $l : V' \rightarrow \mathcal{L}'$ *é uma função que associa um rótulo em* $\mathcal{L}'$ *a cada vértice em* $V'$.

Conforme mencionado anteriormente, esquemas XML são árvores e determinam uma relação hierárquica entre os elementos. Em função disso, a mesma informação pode ser representada por árvores diferentes, dependendo da escolha que se fizer sobre os relacionamentos hierárquicos. Retomando-se o mesmo exemplo apresentado anteriormente, verifica-se que informações sobre professores e disciplinas podem ser representadas de duas formas: os professores com as suas disciplinas, ou as disciplinas com os seus professores. A fig. 4.6 exemplifica ambas as situações. Percebe-se claramente que a representação XML força uma assimetria inexistente no domínio de problema, uma vez que a relação entre professores e disciplinas é simétrica. Essa diversidade de representações para as mesmas informações pode gerar problemas sérios na recuperação e integração de dados XML. Para resolver esse problema, pode-se associar um *modelo conceitual* a conjuntos de documentos XML. Esse modelo conceitual representa o domínio do problema, permitindo a representação fiel das associações entre conceitos.

# 5 Linguagem visual para consulta XML baseada em ontologias

A importância das ontologias para a representação e recuperação de informações é que ela pode ser interpretada como um modelo conceitual, que descreve os conceitos e relações entre conceitos dentro de um determinado domínio. Sob esse aspecto, informações armazenadas em bancos de dados podem ser associadas a uma ontologia, permitindo que instruções de recuperação de informações possam ser baseadas nos conceitos por ela definidos. Com isso, consultas expressas em termos da ontologia podem ser utilizadas para recuperar informações do banco de dados.

No contexto específico desse trabalho, as linguagens de consulta para XML apresentadas no capítulo 2 possuem como característica comum expressões com ênfase sintática, o que significa dependência total da estrutura dos documentos, que o usuário é obrigado a conhecer. Isso acaba por limitar a flexibilidade na expressão das consultas e na construção do resultado.

Quando a consulta é construída com base na ontologia, entretanto, o usuário pode visualizar a base de documentos de acordo com o seu ponto de vista, construindo *visões* XML sobre os dados estruturados. Considerando-se, por exemplo, a ontologia da seção 4.2, pode-se imaginar que diversos usuários tenham visões diferentes dela. Assim, o sistema deve prover recursos para que os usuários possam organizar os conceitos de acordo com o seu ponto de vista.

Esse modelo de consulta se tornará ainda mais atrativo para o usuário se for oferecido um mecanismo de construção que o exima de conhecer e escrever expressões em forma textual. Nesse sentido, a utilização de uma linguagem visual aproxima ainda mais o usuário dos dados a serem consultados.

Para que esse esquema possa ser disponbilizado, entretanto, é necessário inicialmente definir quais as relações existentes entre ontologias e esquemas XML, o que é feito na seção a seguir, para, depois, apresentar-se a proposta de linguagem visual para consultas a XML.

## 5.1 Documentos XML compatíveis com a ontologia

Uma vez definida a ontologia do domínio do problema, podem-se construir documentos XML compatíveis com as suas definições, utilizando esquemas XML adequados. Na bibliografia, descrevem-se várias técnicas que podem ser utilizadas para executar essa tarefa, dentre as quais as de [DOR 00] e [ERD 00a]. Essas técnicas utilizam ontologias para gerar DTDs automaticamente, sem intervenção por parte do usuário. Isso se deve ao fato de que o seu objetivo está relacionado à extração de dados de diferentes fontes, sem participação do usuário. Nesse trabalho, ao contrário, o objetivo é propor uma ferramenta que sirva como interface de consulta para dados já extraídos e armazenados em um banco de dados. Nesse sentido, considera-se o papel do usuário no processo de criação do esquema, motivo pelo qual utiliza-se uma abordagem diferenciada: cabe ao usuário, por meio de uma interface gráfica, definir, a partir dos conceitos da ontologia, qual é o esquema XML que representa a sua visão sobre os dados.

O primeiro passo na construção da consulta consiste em representar graficamente a ontologia. A representação gráfica gerada permitirá ao usuário:

- visualizar os conceitos que compõem o domínio do problema, bem como as associações entre eles;
- delimitar visualmente os conceitos que farão parte da consulta.

A partir da representação gráfica da ontologia, o usuário deverá especificar o esquema XML de seu interesse. A fim de garantir a compatibilidade entre a visão do usuário e a ontologia, o sistema deverá validar o esquema, evitando representações incorretas. Assim, para cada tipo de elemento inserido pelo usuário no esquema XML deverá haver um conceito equivalente na ontologia. Além disso, a relação entre elementos e conceitos deve ser consistente, seguindo as regras estabelecidas a seguir. As regras levam em conta que ontologias não são estruturas totalmente hierárquicas – é possível descrevê-las como grafos não-ordenados –, enquanto que documentos XML são hierarquicamente estruturados. Essa diferença exige algum tipo de mapeamento, conforme o que se apresenta na seção a seguir.

### 5.1.1 Regras de mapeamento entre ontologias e esquemas XML

Estudos como o de [DOR 00] e o de [ERD 00a] provam que esquemas XML podem ser derivados a partir de uma mesma ontologia, utilizando-se as definições de classes e as associações entre elas. É possível, portanto, que, dado um esquema XML, se possa verificar a sua *compatibilidade* com uma determinada ontologia. Propõe-se a utilização dos grafos oriundos das Definições 1 e 2 para a realização dessa verificação. Um possível mapeamento da associação existente entre os grafos das figuras 4.3 e 4.5 é apresentado graficamente na fig. 5.1. Nesse caso, os elementos que compõem o esquema XML representado à direita na figura são mapeados para os conceitos que compõem a ontologia. Esse mapeamento é indicado pelas linhas pontilhadas que apontam para os conceitos sombreados da ontologia, representada à esquerda na figura.

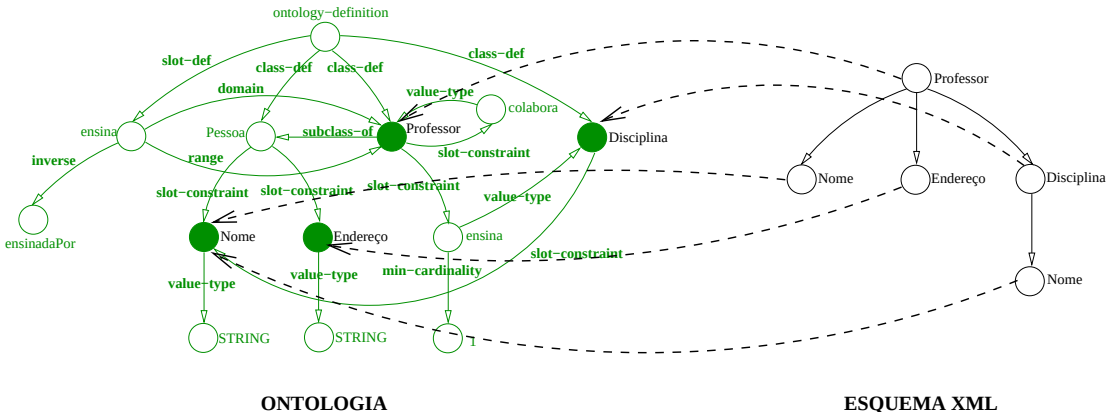


FIGURA 5.1 – Representação de um possível mapeamento entre os grafos que representam a ontologia e o esquema XML.

Conforme se pode observar na figura, cada elemento do esquema XML é mapeado para um conceito da ontologia, sejam eles classes ou atributos. Deve-se considerar, entretanto, que nem todas as ocorrências desses mapeamentos estão de acordo com o conceito de *compatibilidade* entre ontologias e esquemas XML. Para isso, devem ser respeitadas algumas restrições, que são apresentadas a seguir.

**Definição 3 (Compatibilidade ontologia-esquema XML)** *diz-se que um esquema XML é* compatível *com uma ontologia frente a um mapeamento $m : V' \rightarrow V$ quando:*

*i)* $\forall v' \in V', v \in V : v = m(v') \Rightarrow l(v') = l(v)$;
*(Para cada vértice do esquema XML existe um mapeamento para um vértice do grafo que representa a ontologia.)*

*ii)* $\exists r \in V \Rightarrow \exists a \in A, v \in V \wedge convergente(a,v) \wedge l(a) = class\text{-}def$;
*(A raiz de um esquema XML deve ser sempre uma classe da ontologia.)*

*iii)* $\exists r \in V \Rightarrow V_N \neq V_T$;
*(O esquema XML não pode ser composto apenas pela raiz.)*

*iv)* $\forall v\prime \in V'_N \Rightarrow \exists v \in V, a \in A : convergente(a,v) \wedge l_A(a) \in \{class\text{-}def, equivalent, has\text{-}value, value\text{-}type, range\} \wedge l(v\prime) = l_V(v)$;
*(Para cada vértice não-terminal do grafo que representa o esquema XML deve haver uma classe definida na ontologia.)*

*v)* $\forall v'_1 \in V_N, v'_2 \in V_T, a' \in A' : conecta(a', v'_1, v'_2) \Rightarrow \exists v_1, v_2, v_3 \in V, a_1, a_2 \in A : l(v'_1) = l(v_1) \wedge l(v'_2) = l(v_2) \wedge conecta(a_1, v_1, v_2) \wedge conecta(a_2, v_2, v_3) \wedge l_A(a_1) = slot\text{-}constraint \wedge l_V(v_3) \in \{STRING, INTEGER\}$;
*(Para cada vértice terminal do grafo que representa o esquema XML, deve haver na ontologia um atributo do tipo STRING ou INTEGER, subordinado a uma classe.)*

*vi)* $\forall v'_1, v'_2, v'_3 \in V', a'_1, a'_2 \in A' : conecta(a', v'_1, v'_2) \wedge conecta(a'_2, v'_2, v'_3) \Rightarrow \exists v_1, v_2, v_3 \in V, a_1, a_2 \in A : l_v(v_1) = l(v'_1) \wedge l_V(v_2) = l(v'_2) \wedge l_V(v_3) = l(v'_3 \wedge l(a\prime_1) = slot\text{-}constraint \wedge l(a'_2) \in \{value\text{-}type, has\text{-}value, min\text{-}cardinality, max\text{-}cardinality\}$;
*(Para cada estrutura do grafo que representa o esquema XML composta por três vértices não-terminais e duas arestas que os conectam, devem haver duas classes na ontologia, conectadas por um* slot-constraint.*)*

Esquemas XML são compatíveis quando respeitam todas essas regras. Pode-se considerar que um esquema XML compatível com uma determinada ontologia representa um *caminho* percorrido sobre ela. O exemplo da fig. 5.1 permite comprovar essa afirmação. A raiz do esquema XML, neste caso, é o elemento `<Professor>`, que é um mapeamento do conceito *Professor* da ontologia. Este, por sua vez, associa-se ao conceito *Pessoa* por um relacionamento de subclasse, indicado pela aresta **subclass-of**. O relacionamento de subclasse indica que *Professor* herda de *Pessoa* todos os seus atributos e relacionamentos. Assim, são atingidos os conceitos *Nome* e *Endereço*, relacionados a *Pessoa* por arestas do tipo **slot-constraint**. Por outro lado, *Disciplina* está acessível a partir de *Professor* por um **slot-constraint**, que leva ao conceito *ensina*, a partir do qual, seguindo-se uma aresta **value-type**, atinge-se a *Disciplina*. Finalmente, atinge-se o *Nome* novamente, dessa vez a partir de uma aresta **slot-constraint** que se origina em *Disciplina*. A DTD para o grafo XML apresentado na fig. 5.1, bem como dados de exemplo, são apresentados na fig. 5.2.

Além deste esquema XML, outros podem ser produzidos a partir da mesma ontologia, como, por exemplo, o da fig. 5.3. O exemplo representa em XML a relação de colaboração entre professores. Nesse caso, cada professor representa um mapeamento da classe "Professor" da ontologia, e ambos são ligados por um **slot-constraint** "colabora".

Esquemas XML que violarem qualquer uma das restrições acima não são considerados compatíveis com a ontologia. Tomando-se como exemplo a ontologia da fig. 4.2, pode-se testar a compatibilidade de um dado esquema como o da fig. 5.5.

```
<!DOCTYPE Professor [
<!ELEMENT Professor (Nome, Endereço, Disciplina+)>
<!ELEMENT Disciplina (Nome)>
<!ELEMENT Nome (#PCDATA)>
<!ELEMENT Endereço (#PCDATA)>
]>

<Professor>
  <Nome>João da Silva</Nome>
  <Endereço>Porto Alegre</Endereço>
  <Disciplina><Nome>Projeto de Banco de Dados</Nome></Disciplina>
  <Disciplina><Nome>Banco de Dados I</Nome></Disciplina>
</Professor>
```

FIGURA 5.2 – DTD e dados de exemplo para o grafo da fig. 5.1

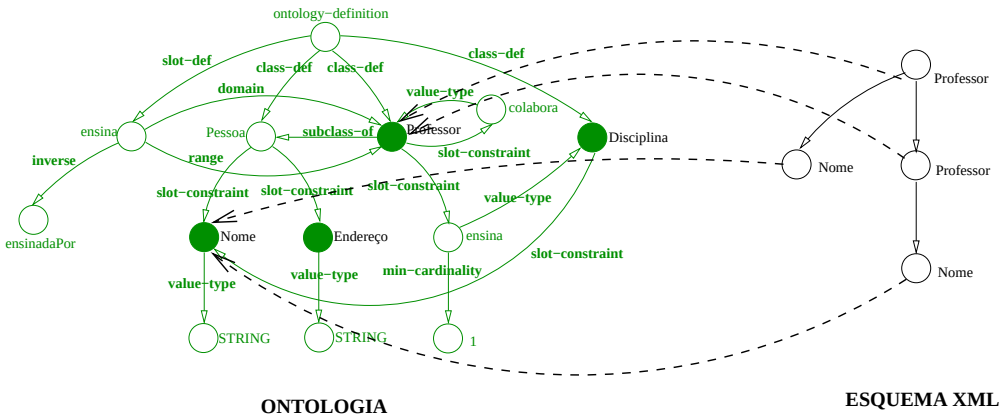


FIGURA 5.3 – Representação do mapeamento entre os grafos que representam a ontologia e o esquema XML – exemplo 2.

```
<!DOCTYPE Professor [
<!ELEMENT Professor (Nome, Professor)>
<!ELEMENT Disciplina (Nome)>
]>

<Professor>
  <Nome>João da Silva</Nome>
  <Professor>
     <Nome><José de Souza</Nome>
  </Professor>
</Professor>
```

FIGURA 5.4 – DTD e dados de exemplo para o grafo da fig. 5.3

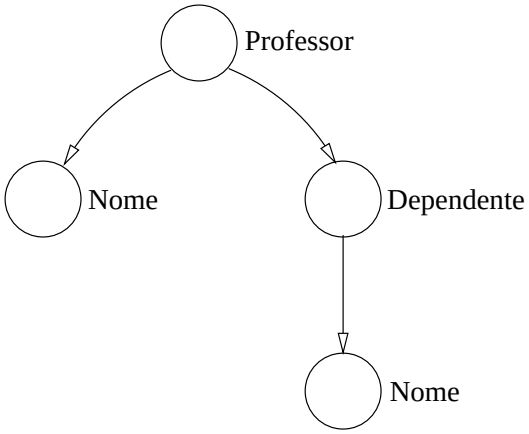


FIGURA 5.5 – Exemplo de esquema XML que viola a restrição de mapeamento *i*.

Tal esquema não pode ser considerado compatível, pois embora os vértices rotulados com *Professor* e *Nome* podem ser mapeados para vértices da ontologia, o mesmo não acontece com o vértice rotulado como *Dependente*. Isso decorre do fato de que não há na ontologia nenhum vértice cujo rótulo seja igual a "Dependente". Ocorre, portanto, uma violação da restrição *i*.

Por outro lado, analisando-se o esquema da fig. 5.6, verifica-se que ele está de acordo com a primeira restrição, pois todos os seus vértices podem ser mapeados para vértices da ontologia. Quando confrontado com a segunda restrição, entretanto, revela-se falho, uma vez que *Endereço* não corresponde ao nome de uma classe definida por meio de um **class-def** na ontologia e, entretanto, aparece como raiz do esquema XML. Ao mesmo tempo, o esquema viola a terceira restrição, por ser composto apenas pelo elemento raiz.


FIGURA 5.6 – Exemplo de esquema XML que viola as restrições de mapeamento *ii* e *iii*.

Já o esquema da fig. 5.7 satisfaz as restrições *i, ii* e *iii*, pois todos os seus vértices podem ser mapeados para vértices da ontologia; a raiz é uma classe definida com **class-def** e o esquema não é composto apenas pela raiz. A restrição *iv*, entretanto, não é satisfeita pelo esquema, pois há um vértice não-terminal, *Endereço*, para o qual não existe na ontologia uma classe com o mesmo nome. O esquema, portanto, não é compatível com a ontologia.

O esquema da fig. 5.8, por sua vez, satisfaz as restrições *i* a *iv*.A restrição *v*, entretanto, estabelece que os vértices terminais do esquema devem ser mapeados para atributos do tipo *string* ou *integer* na ontologia, os quais devem ser subordinados a uma classe. No exemplo, *Disciplina* é um vértice terminal do esquema, cujo rótulo é igual ao nome de uma classe na ontologia, violando, portanto, a restrição *v*. Em função disso, o esquema não é compatível com a ontologia.

O esquema da fig. 5.9(a) satisfaz as restrições *i* a *v*, violando, entretanto, a restrição *vi*, pois não existe um *slot-constraint* entre as classes *Professor* e *Disciplina* cujo nome

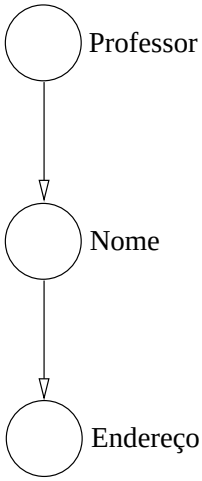


FIGURA 5.7 – Exemplo de esquema XML que viola a restrição de mapeamento *iv*.

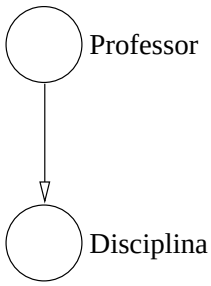


FIGURA 5.8 – Exemplo de esquema XML que viola a restrição de mapeamento *v*.

seja *ministra*. Existe, isso sim, um *slot-constraint* chamado *ensina*, o que faz com que o esquema da fig.5.9(b) seja compatível com a ontologia.

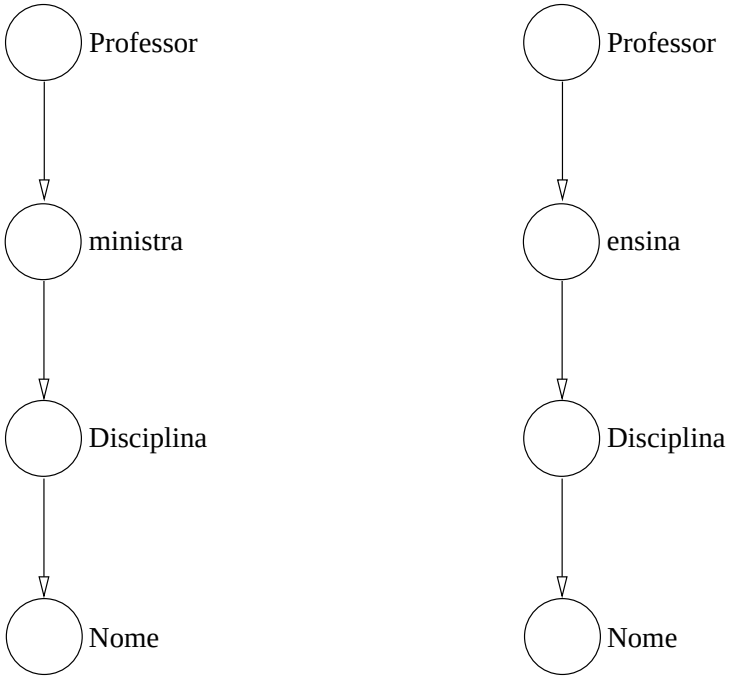


FIGURA 5.9 – Dois exemplos de esquemas XML, sendo que (a) viola a restrição de mapeamento *vi*, enquanto que (b) é compatível com a ontologia.

## 5.2 Linguagem visual para consultas a XML

Na seção anterior, definiu-se a relação de compatibilidade entre ontologias e esquemas XML. Com base nas definições apresentadas, pode-se elaborar consultas a fontes XML utilizando ontologias como modelos conceituais. As consultas poderiam ser expressas em uma linguagem textual, a exemplo daquelas apresentadas no capítulo 2. Entretanto, conforme discutido no capítulo 3, a usabilidade dessas linguagens não é a ideal para o usuário final. Assim, propõe-se nesse trabalho uma linguagem visual para consultas a XML usando a ontologia como modelo conceitual. Nesta seção, apresenta-se, inicialmente, a *XML Query By Example* – XQBE, uma linguagem visual proposta para consultar XML e, após, uma interface visual que combina essa linguagem com a utilização de ontologias.

### 5.2.1 XML Query By Example

Os exemplos apresentados na seção anterior mostram a forma como QBE utiliza a representação tabular do modelo de dados relacional para formular as consultas. A principal diferença um banco de dados relacional e um documento XML é justamente o modelo de dados. No caso do modelo de dados relacional, os dados são representados como *relações*, compostas por *atributos* e *tuplas*, formando uma estrutura tabular. Em XML, por outro lado, os dados são estruturados em *elementos* e *sub-elementos*, constituindo

uma estrutura *hierárquica*. Essa diferença exige adaptações na representação visual das consultas, embora a idéia básica possa permanecer a mesma.

A seguir, apresentam-se exemplos de consultas em XQBE, tomando-se como exemplo os dados da fig. 5.10. Para facilitar a compreensão, cada exemplo em XQBE será acompanhado de uma consulta equivalente em XQL[1], bem como do resultado da aplicação da consulta aos dados de exemplo.

Observando-se o primeiro exemplo, apresentado na consulta 5.1, percebe-se que a primeira coluna contém o esquema XML da fig 5.10 e as demais contém as restrições de filtragem sobre esses dados. O objetivo da consulta é retornar os nomes de todas as disciplinas cujo nome do professor seja "Pedro". Para isso, o operador P. foi escrito ao lado do elemento `<nome>` que está subordinado a `<disciplina>`, o que indica que ele deve fazer parte do resultado. Já a palavra "Pedro" escrita ao lado do sub-elemento `<nome>` de `<professor>` serve como um filtro, indicando que somente participarão do resultado aquelas instâncias do elemento cujos conteúdos contenham a palavra indicada.

| Esquema XML | |
|---|---|
| curriculo | |
| disciplina | |
| nome | P. |
| professor | |
| nome | Pedro |

(a)

```
document("curriculo.xml")/curriculo/
  disciplina[professor/nome='Pedro']/
    nome
```

(b)

```
<xql:result>
  <nome>Fundamentos de
        Banco de Dados</nome>
  <nome>Projeto de Banco
        de Dados</nome>
</xql:result>
```

(c)

CONSULTA 5.1: Retornar os nomes de todas as disciplinas ministradas pelo professor cujo nome seja "Pedro".

A consulta 5.2 é uma variação da consulta anterior, na qual se acrescenta o elemento `<disciplina>` ao resultado.

Ambas as consultas 5.1 e 5.2 utilizam apenas o operador P. e a constante que serve como filtro ao nome do autor. Essas construções são insuficientes para elaborar consultas condicionais com *e* e *ou*. Para elaborar esse tipo de consulta, é necessário utilizar *variáveis*, que são prefixadas pelo símbolo de sublinhado ("_"). Um exemplo é o da fig. 5.3, que busca os nomes de todas as disciplinas ministradas pelos professores "Pedro" *e* "Paulo". A condição *e* é indicada pelo uso da mesma variável "_BD" nas duas colunas da consulta, uma equivalente ao nome "Pedro" e outra ao nome "Paulo". Para expressar con-

[1]Durante a elaboração desse trabalho, o campo de pesquisas em linguagens de consulta a XML evoluiu bastante, o que culminou na proposta da XQuery do W3C. Entretanto, quando esta linguagem foi proposta, já se havia escolhido XQL para ser utilizada no presente trabalho. Estudos futuros considerarão a linguagem do W3C.

```
<?xml version="1.0"?>
<!DOCTYPE curriculo [
<!ELEMENT curriculo (disciplina+)>
<!ELEMENT disciplina (nome, professor+)>
<!ELEMENT nome (#PCDATA)>
<!ELEMENT professor (nome)>
]>

<curriculo>
  <disciplina>
    <nome>Fundamentos de banco de dados</nome>
    <professor>
      <nome>Pedro</nome>
    </professor>
  </disciplina>
  <disciplina>
    <nome>Projeto de banco de dados</nome>
    <professor>
      <nome>Pedro</nome>
    </professor>
    <professor>
      <nome>Paulo</nome>
    </professor>
  </disciplina>
  <disciplina>
    <nome>Bancos de dados distribuídos</nome>
    <professor>
      <nome>Paulo</nome>
    </professor>
  </disciplina>
  <disciplina>
    <nome>Estruturas de dados</nome>
    <professor>
      <nome>João</nome>
    </professor>
  </disciplina>
</curriculo>
```

FIGURA 5.10 – Dados de exemplo para as consultas em linguagem visual: professores e disciplinas.

| Esquema XML | |
|---|---|
| curriculo | |
| disciplina | P. |
| nome | P. |
| professor | |
| nome | Pedro |

(a)

```
document("curriculo.xml")/curriculo
  /disciplina[professor/nome='Pedro']
     { nome }
```

(b)

```
<xql:result>
  <disciplina>
    <nome>Fundamentos de
          banco de dados</nome>
  </disciplina>
  <disciplina>
    <nome>Projeto de banco
          de dados</nome>
  </disciplina>
</xql:result>
```

(c)

CONSULTA 5.2: Retornar todos os nomes das disciplinas ministradas pelo professor cujo nome seja “Pedro”.

sultas com *ou*, basta utilizar nomes de variáveis diferentes, conforme se pode observar no exemplo da fig. 5.4.

Consultas mais complexas podem ser elaboradas com a utilização de mais de um esquema XML. Na fig. 5.11, apresenta-se um esquema XML que representa os alunos com as disciplinas cursadas, juntamente com o semestre em que a mesma foi realizada e o desempenho do aluno.

```
<?xml version="1.0"?>
<!DOCTYPE boletim [
<!ELEMENT boletim (aluno, disciplina+)>
<!ELEMENT aluno (matricula, nome)>
<!ELEMENT disciplina (nome, ano_sem, nota?)>
<!ELEMENT nome (#PCDATA)>
<!ELEMENT ano_sem (#PCDATA)>
<!ELEMENT nota (#PCDATA)>
]>
```

FIGURA 5.11 – Esquema XML para os dados de exemplo da fig. 5.12.

A consulta 5.5 busca o nome dos alunos que foram aprovados no semestre 2000/2, considerando que a nota mínima para aprovação é 7.0. Para resolver essa consulta, aplica-se um teste condicional sobre o elemento <nota>, obrigando que o valor seja maior ou igual a 7.0.

A consulta 5.6 retorna o nome de todos os professores que têm alunos matriculados

| Esquema XML | | |
|---|---|---|
| disciplina | P. | |
| nome | P._BD | _BD |
| professor | | |
| nome | Pedro | Paulo |

(a)

```
document("curriculo.xml")/curriculo/
  disciplina[professor/nome='Pedro' and
             professor/nome='Paulo']
               { nome }
```

(b)

```
<xql:result>
  <disciplina>
    <nome>Projeto de Banco
          de Dados</nome>
  <disciplina>
</xql:result>
```

(c)

CONSULTA 5.3: Retornar todos os nomes das disciplinas ministradas pelos professores "Pedro" *e* "Paulo".

| Esquema XML | | |
|---|---|---|
| disciplina | P. | |
| nome | P._BD1 | _BD2 |
| professor | | |
| nome | Pedro | Paulo |

(a)

```
document("curriculo.xml")/curriculo/
  disciplina[professor/nome='Pedro' or
             professor/nome='Paulo']
               { nome }
```

(b)

```
<xql:result>
  <disciplina>
    <nome>Fundamentos de
          banco de dados</nome>
  </disciplina>
  <disciplina>
    <nome>Projeto de Banco
          de Dados</nome>
  </disciplina>
  <disciplina>
    <nome>Bancos de dados
          distribuídos</nome>
  </disciplina>
</xql:result>
```

(c)

CONSULTA 5.4: Retornar todos os nomes das disciplinas ministradas pelos professores cujos nomes sejam "Pedro" *ou* "Paulo".

```
<boletim>
  <aluno>
    <matricula>1001</matricula>
    <nome>João da Silva</nome>
    <disciplina>
      <nome>Fundamentos de banco de dados</nome>
      <ano_sem>2000/2</ano_sem> <nota>7.0</nota>
    </disciplina>
    <disciplina>
      <nome>Projeto de banco de dados</nome>
      <ano_sem>2001/1</ano_sem>
    </disciplina>
  </aluno>
  <aluno>
    <matricula>995</matricula>
    <nome>José de Souza</matricula>
    <disciplina>
      <nome>Fundamentos de banco de dados</nome>
      <ano_sem>2000/1</ano_sem> <nota>7.0</nota>
    </disciplina>
    <disciplina>
      <nome>Projeto de banco de dados</nome>
      <ano_s.em>2000/2</ano_sem> <nota>8.5</nota>
    </disciplina>
    <disciplina>
      <nome>Bancos de dados distribuídos</nome>
      <ano_sem>2001/1</ano_sem>
    </disciplina>
  </aluno>
  <aluno>
    <matricula>1005</matricula>
    <nome>Antônio dos Santos</nome>
    <disciplina>
      <nome>Projeto de banco de dados</nome>
      <ano_sem>2000/2</ano_sem> <nota>5.0</nota>
    </disciplina>
    <disciplina>
      <nome>Estruturas de dados</nome>
      <ano_sem>2001/1</ano_sem>
    </disciplina>
  </aluno>
</boletim>
```

FIGURA 5.12 – Dados de exemplo para as consultas em linguagem visual: boletim dos alunos.

| Esquema XML | |
|---|---|
| boletim | |
| aluno | P. |
| matricula | |
| nome | P. |
| disciplina | |
| nome | |
| ano_sem | 2000/2 |
| nota | >= 7.0 |

(a)

```
document("boletim.xml")/boletim
/aluno[disciplina/ano_sem='2000/2' and
       disciplina/nota gt 7.0]
         { nome }
```

(b)

```
<xql:result>
  <aluno>
    <nome>João da Silva</nome>
    <nome>José de Souza</nome>
  </aluno>
</xql:result>
```

(c)

CONSULTA 5.5: Retornar os nomes de todos os alunos aprovados no semestre 2000/2.

em suas disciplinas em 2001/1. Nesse caso, são necessários ambos os esquemas, pois são aplicados filtros sobre elementos de professores e também de alunos, que se encontram em esquemas diferentes. A variável "_DISC" aparece nos dois esquemas, indicando que será feito um *join* com o conteúdo dos nomes das disciplinas.

### 5.2.2 Consultas que não podem ser expressas em XQL

O poder de expressão da XQBE ultrapassa o de XQL; portanto, certas consultas válidas na primeira não podem ser expressas na segunda linguagem. Na consulta 5.6, foram selecionados para apresentação elementos de apenas um esquema XML. Se houver necessidade de apresentar elementos de mais de um esquema, deve-se definir a estrutura do resultado, construindo-se um novo esquema, conforme ilustrado na consulta 5.7. A construção do resultado faz-se da seguinte maneira:

1. variáveis são ligadas ao(s) esquema(s) de origem;

2. no esquema-resultado, aqueles elementos que não possuem variáveis ligadas são estruturais, ou seja, não contém valores de texto, enquanto que aqueles que possuem variáveis ligadas receberão os valores de texto dos elementos do(s) esquema(s) de origem aos quais as variáveis foram ligadas.

Esquemas cuja função é construir o resultado permitem transformações mais profundas nos esquemas de origem, permitindo que se obtenha no resultado documentos totalmente novos. XQL, entretanto, sempre reflete a estrutura do documento de origem no resultado, pois não possui instruções para produzir novos documentos no resultado.

| Esquema XML | |
|---|---|
| curriculo | |
| disciplina | |
| nome | _DISC |
| professor | P. |
| nome | P. |

| Esquema XML | |
|---|---|
| boletim | |
| aluno | |
| matricula | |
| nome | |
| disciplina | |
| nome | _DISC |
| ano_sem | 2001/1 |
| nota | |

(a)

```
document("boletim.xml")/boletim -> bolet_disc {
  /boletim[disciplina/ano_sem='2001/1'] -> bolet_disc[$d := disciplina] {
  document("curriculo.xml")/curriculo[disciplina = $d]/disciplina
   { nome | professor/nome }
  }
}
```

(b)

```
<xql:result>
    <professor>
      <nome>Pedro</nome>
    </professor>
    <professor>
      <nome>Paulo</nome>
    </professor>
    <professor>
      <nome>João</nome>
    </professor>
</xql:result>
```

(c)

CONSULTA 5.6: Retornar o nome de todos os professores que têm alunos matriculados em suas disciplinas em 2001/1.

| Esquema XML | |
|---|---|
| curriculo | |
| disciplina | |
| nome | _DISC |
| professor | |
| nome | _PROF |

| Esquema XML | |
|---|---|
| boletim | |
| aluno | |
| matricula | |
| nome | _ALUNO |
| disciplina | |
| nome | _DISC |
| ano_sem | 2001/1 |
| nota | |

| Esquema XML: resultado | |
|---|---|
| disciplina | |
| nome | _DISC |
| professor | |
| nome | _PROF |
| aluno | |
| nome | _ALUNO |

(a)

```
<xql:result>
  <disciplina>
    <nome>Projeto de banco de dados</nome>
    <professor>
      <nome>Pedro</nome>
    </professor>
    <professor>
      <nome>Paulo</nome>
    </professor>
    <aluno>
      <nome>João da Silva</nome>
    </aluno>
    <aluno>
      <nome>José de Souza</nome>
    </aluno>
    <aluno>
      <nome>Antonio dos Santos</nome>
    </aluno>
  </disciplina>
  <disciplina>
  ...
  </disciplina>
  ...
</xql:result>
```

(b)

CONSULTA 5.7: Buscar os nomes de todos os professores que têm alunos matriculados em suas disciplinas em 2001/1, construindo-os no resultado, juntamente com o nome da disciplina e os nomes dos alunos.

### 5.2.3 Consultas que não podem ser expressas em XQBE

Assim como há consultas que podem ser representadas em XQBE, mas não em XQL, também o inverso acontece. Isso decorre das restrições impostas pela representação visual da consulta. Um exemplo de consulta que não pode ser expressa em XQBE é aquela que procura por um determinado elemento, independentemente no nível de profundidade em que ele se encontra no documento XML. Por exemplo, para obter todos os elementos <nome> que se encontram em qualquer lugar no documento XML, pode-se escrever:

```
document("boletim.xml")//nome
```

Consultas deste tipo não são passíveis de representação em XQBE, uma vez que não existe construção na linguagem equivalente ao operador "//".

## 5.3 A interface de consulta

Na seção anterior, apresentou-se a XQBE, uma linguagem visual de consulta para XML que utiliza o paradigma baseado em formulários. Conforme discutido anteriormente, a usabilidade de linguagens de consulta a XML pode ser incrementada com a utilização de ontologias como modelos conceituais para documentos XML. Nesta seção, propõe-se uma interface de consulta a documentos XML que combina ontologias com expressões de consulta em XQBE. A interface ilustra a definição conceitual apresentada nos capítulos 4 e 5. Utiliza-se como exemplo a ontologia apresentada na fig. 4.2.

Os elementos principais da interface são:

- a *janela da ontologia*, na qual se pode ver uma representação gráfica da ontologia;
- as *janelas de consulta*, em que o usuário pode construir um esquema XML que represente a sua visão da ontologia, e também expressar as restrições sobre esses elementos, a fim de construir a consulta. Para uma mesma ontologia, diversas janelas de consulta podem ser construídas.

A fig. 5.13 apresenta uma sessão de consulta na interface, apresentando a janela da ontologia e uma janela de consulta.

Em uma sessão típica de consulta, o usuário irá desenvolver os seguintes passos:

1. abrir o arquivo da ontologia, selecionando a opção *Open ontology* do menu *File*. Nesse momento, será criada uma janela gráfica mostrando os elementos da ontologia e as associações entre eles.
2. criar uma ou mais janelas de esquemas, selecionando a opção *New query* do menu *File*.
3. selecionar com o mouse elementos da ontologia e arrastá-los para a janela de consulta escolhida.
4. alterar ou remover elementos da janela de consulta.
5. filtrar os elementos da consulta, preenchendo as colunas de condição nas células referentes a cada elemento desejado.

Deve-se ressaltar que cada conjunto de conceitos incluído na consulta será validado, de acordo com as regras estabelecidas nos capítulos 4 e 5 deste trabalho.

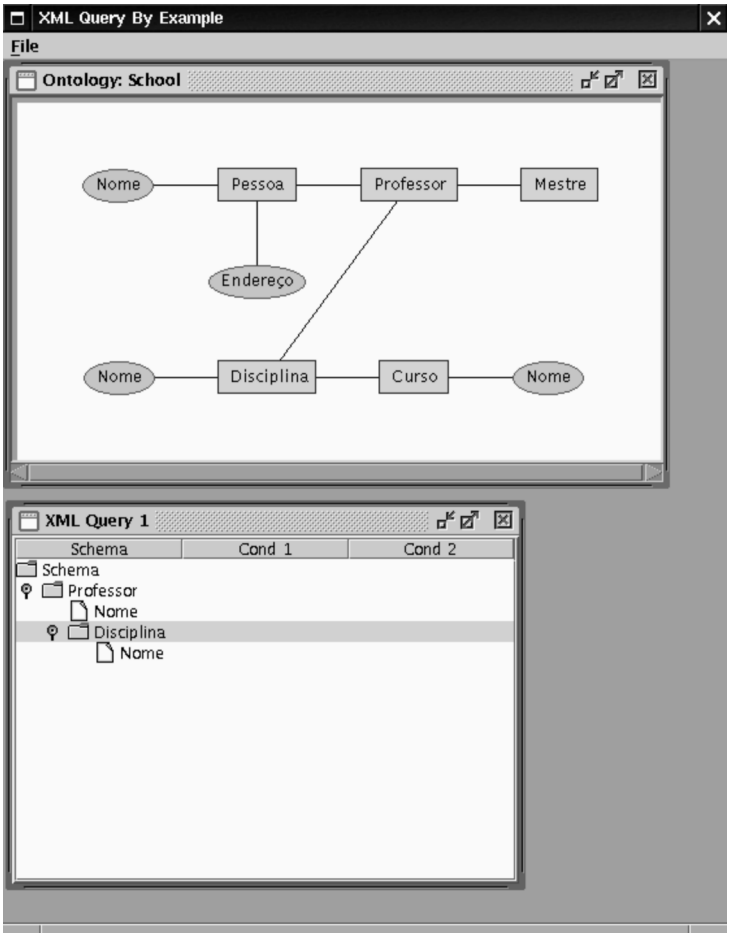


FIGURA 5.13 – Visualização da interface, mostrando uma janela de ontologia e uma janela de consulta.

## 5.4 Conclusões

Os conceitos apresentados neste capítulo, que incluem a definição de compatibilidade entre ontologias e esquemas XML, a linguagem XQBE e a interface de consulta, compõem o núcleo deste trabalho. O objetivo dessa proposta é permitir consultas mais flexíveis a documentos XML, utilizando ontologias como modelos conceituais.

# 6 Conclusões e trabalhos futuros

Neste trabalho, propôs-se uma linguagem visual de consulta para dados XML, com base em uma ontologia. A linguagem visual proposta foi inspirada em QBE, uma linguagem de consulta projetada para consultar bancos de dados relacionais.

O problema de consultar dados XML foi discutido com a apresentação dos requisitos exigidos para uma linguagem destinada a esse fim, ilustrada pela exposição das principais linguagens de consulta propostas na literatura: Lorel, XML-QL, XQL, YATL e Quilt. Apesar de adequadas para interfaces programáticas e para utilização por usuários experientes, essas linguagens possuem uma sintaxe textual, o que reduz a curva de aprendizado de usuários leigos ou com pouco conhecimento sobre a estrutura das informações armazenadas. Em função disso, discutiu-se também a importância das linguagens visuais para consulta a bancos de dados, em especial XML. Diversos estudos demonstram que linguagens de consulta visuais são mais facilmente assimiladas pelos usuários menos experientes, embora nem sempre tenham o mesmo poder de expressão das linguagens textuais.

Outro problema discutido nesse trabalho foi a dificuldade de se extrair informações por meio de consultas a fontes de dados semi-estruturadas, devido à estrutura variável dessas fontes. Em função disso, discutiu-se a alternativa de se utilizar ontologias para representar os conceitos presentes nos documentos XML e as relações entre eles. A fim de ilustrar essa idéia, apresentou-se uma linguagem para representação de ontologias – OIL – que possui sintaxe em XML. A partir daí, estabeleceu-se uma relação entre documentos XML e ontologias, por meio da definição de compatibilidade entre ontologias e esquemas XML, tendo como base representações formais de documentos XML e de ontologias OIL.

Por fim, apresentou-se a linguagem de consulta visual para XML, juntamente com a definição de uma interface visual como ambiente de utilização da linguagem. A linguagem visual permite ao usuário construir um esquema XML utilizando os conceitos definidos em uma ontologia de domínio. O usuário deve, a partir do esquema, especificar restrições sobre os elementos que formam o esquema XML, as quais servirão para filtragem dos dados.

Como principais contribuições desse trabalho, cabe salientar:

1. a definição de um modelo conceitual para ontologias e para esquemas XML, bem como das regras de mapeamento que permitem inferir a compatibilidade de um determinado esquema XML com uma ontologia. Essas definições servem como base para o processo de formulação de consultas a XML baseadas em ontologias.

2. a especificação de uma linguagem visual para consultas a XML baseada em ontologias, inspirada na QBE, tradicional linguagem de consulta visual a bancos de dados relacionais.

3. a definição de uma interface visual, que permite criar expressões de consulta a XML com base em ontologias.

Durante o tempo de elaboração desse trabalho, as propostas para linguagens de consulta para XML evoluíram de forma muito rápida. Entretanto, não há um padrão estabelecido nessa área, apenas propostas isoladas, algumas enfatizando o enfoque de bancos de dados e outras, com ênfase nas características de documentos. Apenas em meados de fevereiro de 2001 o W3C publicou um *Working Draft* definindo a primeira

versão para estudo de uma linguagem que deverá vir a se tornar padrão para consultas a XML, chamada XQuery [CHA 01]. A linguagem proposta pelo W3C inspirou-se em Quilt [CHA 00a], linguagem que, como o próprio nome indica, reúne características de várias outras linguagens, agrupando características para consultas a bancos de dados semi-estruturados, bem como outras típicas de consultas a documentos semi-estruturados.

Nesse trabalho, a linguagem que serviu como referência para a elaboração das consultas visuais foi XQL, cujo poder de expressão foi suplantado pela linguagem visual. Por esse motivo, pretende-se, como trabalho futuro, comparar o poder de expressão da XQuery com a linguagem visual proposta, buscando construir exemplos de consultas simples até as mais complexas.

Por outro lado, pretende-se complementar o protótipo de interface proposto, implementando uma versão totalmente funcional da interface de consulta.

# Bibliografia


[ABI 97] ABITEBOUL, S. et al. The Lorel query language for semistructured data. **International Journal on Digital Libraries**, v.1, n.1, p.68–88, 1997.

[AND 96] ANDRIES, M. **Graph rewriting systems and visual database languages**. Netherlands: Leiden University, Department of Computer Science, SEIS Group, 1996. Tese de Doutorado.

[BAR 98] BARU, C. et al. Features and requirements for an XML view definition language: lessons from XML information mediation. In: W3C WORKSHOP ON QUERY LANGUAGES (QL'98), 1998, Boston. **Anais...** [S.l.: s.n.], 1998. Disponível por www em <http://www.w3.org/TandS/QL/QL98/pp/xmas.html>. Acesso em: jun. 1999.

[BEE 99] BEECH, D. et al. **XML Schema Part 1**: structures. Disponível por www em <http://www.w3.org/TR/xmlschema-1>. Acesso em: mar. 2000.

[BON 00] BONIFATI, A.; CERI, S. Comparative analysis of five XML query languages. **SIGMOD Record**, v.20, n.1, p.68–79, Mar. 2000.

[BRI 99] BRICKLEY, D.; GUHA, R. **Resource Description Framework (RDF) Schema Specification**. Disponível por www em <http://www.w3.org/TR/PR-rdf-schema>. Acesso em: ago. 1999.

[CAT 97] CATARCI, T. et al. Visual query systems for databases: a survey. **Journal of Visual Languages and Computing**, v.8, n.2, p.215–260, 1997.

[CER 98] CERI, S. et al. XML-GL: a graphical language for querying and restructuring XML documents. In: W3C WORKSHOP ON QUERY LANGUAGES (QL'98), 1998, Boston. **Anais...** [S.l.: s.n.], 1998. Disponível por www em <http://www3.elet.polimi.it/people/ceri/xml-gl>. Acesso em: jun. 1999.

[CHA 00] CHAMBERLIN, D. et al. **XML Query Requirements**. Disponível por www em <http://www.w3.org/TR/xmlquery-req>. Acesso em: set. 2000.

[CHA 01] CHAMBERLIN, D. et al. **XQuery**: a query language for XML. Disponível por www em <http://www.w3.org/TR/xquery>. Acesso em: mar. 2001.

[CHA 00a] CHAMBERLIN, D.; ROBIE, J.; FLORESCU, D. **Quilt**: an XML query language for heterogeneus data sources. Disponível por www em <http://www.almaden.ibm.com/cs/people/chamberlin/quilt_lncs.pdf>. Acesso em: dez. 2000.

[CHE 75] CHEN, P. P. S. The entity-relationship model — toward a unified view of data. **Proceedings of the 1th Conference on Very Large Databases, Morgan Kaufman pubs. (Los Altos CA), Kerr(ed), pp.173**, 1975.

[CLU 00] CLUET, S.; SIMEON, J. **YATL**: a functional and declarative language for xml. Disponível por www em <http://www-db.research.bell-labs.com/user/simeon/icfp.ps>. Acesso em: set. 2000.

[CRA 99] CRANEFIELD, S.; PURVIS, M. **UML as an ontology modelling language**. Disponível por www em <http://citeseer.nj.nec.com/cranefield99uml.html>. Acesso em: dez. 2000.

[DEU 99] DEUTSCH, A. et al. A query language for XML. In: THE EIGHTH INTERNATIONAL WORLD WIDE WEB CONFERENCE (WWW8), 1999, Toronto. **Anais...** [S.l.: s.n.], 1999.

[DOR 00] DORNELES, C. **Extração de dados semi-estruturados com base em uma ontologia**. Porto Alegre, RS: PPGC-UFRGS, 2000. Dissertação de Mestrado.

[DUB 99] DUBLIN CORE METADATA INITIATIVE. **Dublin Core Metadata Element Set, Version 1.1**: Reference Description. Disponível por www em <http://www.purl.org/DC/documents/rec-dces-19990702.htm>. Acesso em: out. 2000.

[ERD 00] ERDMANN, M.; DECKER, S. **Ontology-aware XML-Queries**. Disponível por www em <http://www.aifb.uni-karlsruhe.de/ mer/Pubs/semantic-xql.webdb00.pdf>. Acesso em: out. 2000, Submitted to WebDB2000, Dallas, Texas.

[ERD 00a] ERDMANN, M.; STUDER, R. **How to structure and access XML documents with ontologies**. Disponível por www em <http://www.aifb.uni-karlsruhe.de/ mer/Pubs/xml-dke-final.pdf>. Acesso em: out. 2000, To appear in: *Data and Knowledge Engineering*, Special Issue on Intelligent Information Integration.

[ERW 00] ERWIG, M. A visual language for XML. In: IEEE SYMPOSIUM ON VISUAL LANGUAGES, 2000, Seattle. **Anais...** [S.l.: s.n.], 2000.

[EVA 01] EVANGELISTA FILHA, I.; LAENDER, A.; SILVA, A. Querying semistructured data by example: the QSByE interface. In: PROCEEDINGS OF THE INTERNATIONAL WORKSHOP ON INFORMATION INTEGRATION ON THE WEB 2001, 2001, Rio de Janeiro. **Anais...** [S.l.: s.n.], 2001.

[FAL 01] FALLSIDE, D. **XML Schema part 0**: primer. Disponível por www em <http://www.w3.org/TR/xmlschema-0>. Acesso em: maio 2001.

[FER 00] FERNANDEZ, M.; ROBIE, J. **XML Query Data Model**. Disponível por www em <http://www.w3.org/TR/query-datamodel>. Acesso em: ago. 2000.

[GOL 98] GOLDMAN, R.; MCHUGH, J.; WIDOM, J. From semistructured data to XML: migrating the Lore data model and query language. In:

W3C QUERY LANGUAGES WORKSHOP (QL'98), 1998, Boston. **Anais...** [S.l.: s.n.], 1998. Disponível por www em <http://www-db.stanford.edu/ widom/lorexml.ps>. Acesso em: jun. 1999.

[GRU 93] GRUBER, T. R. Towards Principles for the Design of Ontologies Used for Knowledge Sharing. In: Formal Ontology in Conceptual Analysis and Knowledge Representation, 1993, Deventer, The Netherlands. **Anais...** Kluwer Academic Publishers, 1993.

[GUA 95] GUARINO, N. Understanding, building and using ontologies. **International Journal of Human and Computer Studies**, v.43, n.5/6, 1995.

[HOR 00] HORROCKS, D. et al. **The ontology inference layer OIL**. Disponível por www em <http://www.cs.vu.nl/ dieter/oil/Tr/oil.pdf>. Acesso em: out. 2000.

[IVE 00] IVES, Z.; LU, Y. **XML query languages in practice**: an evaluation. Disponível por www em <http://www.cs.washington.edu/homes/zives/research/xmlquery.pd>. Acesso em: out. 2000, Web Age Information Management.

[KAR 99] KARP, P.; CHAUDHRI, V. K.; THOMERE, J. **XOL**: and XML-based ontology exchange language. Disponível por www em <http://www.ai.sri.com/pkarp/xol/xol.html>. Acesso em: jan. 2000.

[LAE 99] LAENDER, A.; SILVA, A.; SILVA, E. DEByE: uma ferramenta de extração de dados semi-estruturados. In: XIV SIMPóSIO BRASILEIRO DE BANCO DE DADOS, 1999. **Anais...** [S.l.: s.n.], 1999. p.155–169.

[LAS 99] LASSILA, O.; SWICK, R. **Resource Description Framework (RDF) Model and Syntax Specification**. Disponível por www em <http://www.w3.org/TR/REC-rdf-syntax/>. Acesso em: out. 1999.

[MEL 00] MELLO, R. **Aplicação de ontologias a bancos de dados semi-estruturados**. Porto Alegre, RS: PPGC-UFRGS, 2000. Exame de Qualificação.

[PAP 95] PAPAKONSTANTINOU, Y.; GARCIA-MOLINA, H.; WIDOM, J. Object exchange across heterogeneous information sources. In: PROCEEDINGS OF THE ELEVENTH INTERNATIONAL CONFERENCE ON DATA ENGINEERING, 1995, Taipei, Taiwan. **Anais...** [S.l.: s.n.], 1995. p.251–260.

[ROB 98] ROBIE, J.; LAPP, J.; SCHACH, D. XML Query Language (XQL). In: W3C WORKSHOP ON QUERY LANGUAGES (QL'98), 1998, Boston. **Anais...** [S.l.: s.n.], 1998. Disponível por www em <http://www.w3.org/TandS/QL/QL98/pp/xql.html>. Acesso em: jun. 1999.

[SIL 00] SILVA JUNIOR, A. S. **Materialização de visões relacionais para dados semi-estruturados**. Porto Alegre: PPGC da UFRGS, 2000. Dissertação de Mestrado.

[ZLO 75] ZLOOF, M. Query by Example. In: PROCEEDINGS OF THE NATIONAL COMPUTER CONFERENCE, 1975. **Anais...** [S.l.: s.n.], 1975. v.44, p.431–438.