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

DIEGO DE VARGAS FEIJÓ

Consultando XML por meio de Modelos Conceituais: extensão e formalização de CXPath

Dissertação apresentada como requisito parcial
para a obtenção do grau de
Mestre em Ciência da Computação

Prof. Dr. Carlos Alberto Heuser
Orientador

Prof. Dr. Álvaro Freitas Moreira
Co-orientador

Porto Alegre, junho de 2005

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Feijó, Diego de Vargas

Consultando XML por meio de Modelos Conceituais: extensão e formalização de CXPath / Diego de Vargas Feijó. – Porto Alegre: PPGC da UFRGS, 2005.

55 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2005. Orientador: Carlos Alberto Heuser; Co-orientador: Álvaro Freitas Moreira.

1. XML. 2. Modelo Conceitual Global. 3. Esquema Conceitual. 4. XPath. 5. CXPath. 6. Fontes de Dados XML. 7. Formalização da Linguagem de Consulta. 8. Integração de Dados Semi-estruturados. 9. Validação da Linguagem Contra um Esquema. I. Heuser, Carlos Alberto. II. Moreira, Álvaro Freitas. III. Título.

AGRADECIMENTOS

Agradeço aos meus familiares e amigos pelo apoio, carinho e atenção. Sem vocês nada disso seria possível.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

CM	Conceptual Model
CXPath	Conceptual XPath
DTD	Document Type Definition
ER	Entidade-Relacionamento
MC	Modelo Conceitual
ORM	Object Role Model
SQL	Structured Query Language
W3C	World Wide Web Consortium
XML	Extensible Markup Language

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Com o objetivo de realizar consultas em diferentes fontes XML, é necessário que se escreva uma consulta específica para cada fonte XML. Uma solução mais adequada é fazer uma única consulta sobre um esquema conceitual e então traduzi-la automaticamente para consultas XML para cada fonte XML específica.

CXPath é uma linguagem de consulta que foi proposta para consultar dados em nível conceitual. Este trabalho tem como objetivos formalizar a linguagem existente, estendê-la para permitir consultas que utilizem o conceito de herança e estender o mecanismo de tradução de consultas.

A formalização da linguagem é feita por meio de um conjunto de regras que definem formalmente um critério para validar uma consulta escrita nessa linguagem sobre um esquema conceitual. Essa formalização permite estender a linguagem para que ela passe a tratar os relacionamentos de herança e especialização. Outra contribuição dessa formalização é que ela apresenta o primeiro passo rumo à verificação formal de que a avaliação da consulta global traz os mesmos resultados obtidos pela avaliação da consulta resultante do processo de mapeamento de consultas proposto.

A extensão do mecanismo de tradução de consultas é necessária para traduzir relacionamentos representados no modelo conceitual para junções nas fontes de dados XML. Tal aspecto é fundamental para permitir a construção de modelos conceituais com relacionamentos semânticos e que não dependam de relacionamentos físicos existentes nos documentos fontes, mas apenas de junções tal como é feito em bases de dados relacionais.

Palavras-chave: XML, Modelo Conceitual Global, Esquema Conceitual, XPath, CXPath, Fontes de Dados XML, Formalização da Linguagem de Consulta, Integração de Dados Semi-estruturados, Validação da Linguagem Contra um Esquema.

Querying XML through Conceptual Models: extension and formalization of CXPath

ABSTRACT

In order to query different XML sources, one must write a specific query in accordance with the structure of each XML source. A better solution is to state a single query against a global conceptual schema and then translate it automatically into an XML query for each specific source.

CXPath has been proposed as a language to state such global queries. This work intends to propose a formalization of the existing language, to extend the *CXPath* language to allow queries using joins and to treat the behaviour of inheritance that exists in the used conceptual model and to extend the query translation mechanism.

The formalization is done by presenting a set of rules that formally specifies the criteria for validating a *CXPath* query against a conceptual model. The contribution of this work is the first step towards formally verifying that the evaluation of a *CXPath* query leads to the same results as those of *XPath* queries produced by the translation process.

The translation mechanism is necessary to translate the relationships represented in the conceptual model to joins in the XML sources. Such aspect is very important to allow the building of conceptual models that use semantic relationships and are not limited by the parent/child relationships that exists in the sources.

Keywords: XML, Global Conceptual Model, Conceptual Schema, XPath, CX-Path, XML Data Sources, Query Language Formalization, Semi-structured Data Integration, Query Validation against a Conceptual Schema.

1 INTRODUÇÃO

XML (WORLD WIDE WEB CONSORTIUM, 2005a) é a linguagem padrão da W3C utilizada por vários tipos de aplicações para representação de informações de forma semi-estruturada e troca de dados pela Internet (BRADLEY, 2000). Essa linguagem vem sendo usada para disponibilizar dados na Internet. Não são poucos os sítios que disponibilizam o conteúdo de suas páginas tanto em HTML como em XML.

Essa nova forma de disponibilização de dados permite que aplicações compartilhem dados. Dessa maneira, um sítio que disponibilize determinados dados, pode ter suas informações buscadas por outro sítio que delas necessite. Nessa situação, há a necessidade de garantir que a integração dessas informações se dê de forma consistente.

No contexto de aplicações que oferecem integração de fontes de dados semi-estruturados, a mesma informação em nível semântico pode possuir diferentes representações em XML. Nesse contexto, o principal desafio é lidar com essas diferentes representações de dados semanticamente equivalentes.

Pela própria liberdade que a linguagem XML oferece, uma mesma informação pode estar representada de diferentes maneiras. Isso ocorre porque o esquema de uma fonte XML deve definir uma estrutura hierárquica específica para representação de relacionamentos entre elementos XML. Portanto, fontes XML pertencentes a um mesmo domínio de aplicação podem possuir representações hierárquicas diferentes para um mesmo relacionamento muitos-para-muitos.

Considerando o exemplo de uma biblioteca, um relacionamento muitos-para-muitos entre *artigo* e *autor* pode possuir diferentes representações por dois esquemas XML diferentes:

(i) um *artigo* associado a muitos *autores*, ou

(ii) um *autor* associado a muitos *artigos*.

Na Figura 1.1(a), cada artigo possui vários autores. Já na Figura 1.1(b), cada autor está associado a vários artigos.

Um modelo conceitual representa diretamente relacionamentos muitos-para-muitos (muitos *artigos* associado a muitos *autores* e vice-versa), sem impor navegação uma ordem de navegação entre eles.

Portanto, uma consulta XML depende da estrutura de cada fonte específica. Ou seja, para que uma mesma consulta a nível conceitual seja executada, é necessário que seja elaborada uma consulta específica para cada documento XML que contenha as informações desejadas.

```
<artigos>
  <artigo titulo="Artigo 1">
    <autor nome="Fulano"/>
    <autor nome="Beltrano"/>
  </artigo>
  <artigo titulo="Artigo 2">
    <autor name="Fulano"/>
  </artigo>
  <artigo titulo="Artigo 3">
    <autor nome="Fulano"/>
  </artigo>
</artigos>
```

(a)

```
<autores>
  <autor nome="Fulano">
    <artigo titulo="Artigo 1"/>
    <artigo titulo="Artigo 2"/>
    <artigo titulo="Artigo 3"/>
  </autor>
  <autor nome="Beltrano">
    <artigo titulo="Artigo 1"/>
    <artigo titulo="Artigo 4"/>
    <artigo titulo="Artigo 5"/>
  </autor>
</autores>
```

(b)

Figura 1.1: Formas de representação de um relacionamento muitos-para-muitos

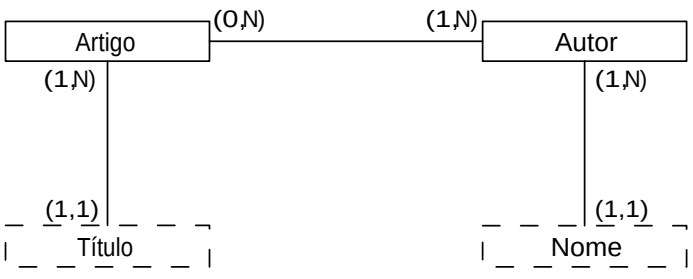

Figura 1.2: Representação conceitual de um relacionamento muitos-para-muitos

Essa solução exige o prévio conhecimento de cada um dos documentos XML, tornando o processo de realizar consultas sobre os documentos um processo demorado. Uma solução mais adequada é a elaboração de consultas sobre um esquema conceitual global, tal como é mostrado na Figura 1.2, e sua tradução automática para a representação XML de cada um dos documentos fontes que compuseram o esquema conceitual global.

Essa solução permite que uma única consulta seja escrita sobre o modelo conceitual e, por meio da tradução automática, a consulta é traduzida para os fontes sem trabalho extra. Além disso, como a tradução para os fontes é automática, quem escreve a consulta não precisa conhecer a estrutura desses documentos.

Tal solução necessita do seguinte (ELMARGAMID; RUSINKIEWCZ; SHETH, 1999):

1. uma representação global (unificada) do esquema de todas fontes XML, evitando que o usuário deva conhecer o esquema de cada fonte XML para elaborar as consultas;

2. uma linguagem para elaboração de consultas em relação a esse esquema conceitual global e um mecanismo de tradução para converter uma consulta global em consultas XML de acordo com o esquema de cada fonte, evitando que várias consultas tenham de ser elaboradas em relação a cada fonte de dados XML;

3. um mecanismo de integração de instâncias para unificar os resultados das consultas vindos de várias fontes em um resultado de acordo com o esquema global.

Um solução para a representação conceitual global para várias fontes de dados XML foi apresentada em (MELLO; HEUSER, 2001a,b). Esses trabalhos descrevem em detalhes o processo de construção do modelo conceitual partindo-se do esquema DTD de cada uma das fontes.

A linguagem *CXPath* (CAMILLO; HEUSER; MELLO, 2003) foi proposta para elaboração de consultas conceituais sobre esquemas conceituais. Essa linguagem foi desenvolvida para simplificar o processo de tradução de consultas em nível conceitual para consultas em nível de estrutura da fonte. Ao mesmo tempo, a linguagem tem como objetivo a facilidade de aprendizado de sua sintaxe. Por essa razão, sua sintaxe é bastante semelhante à da linguagem *XPath* utilizada para consultar documentos XML.

A definição original da linguagem *CXPath* dada em (CAMILLO; HEUSER; MELLO, 2003), no entanto, não contempla uma visão formal da linguagem. Por essa razão, a questão da verificação se uma consulta está bem formada em relação a um modelo conceitual não pode ser resolvida. Nesse trabalho, o mecanismo de tradução proposto assume que as consultas *CXPath* são bem formadas em relação ao esquema conceitual.

Uma das contribuições dessa dissertação é justamente a especificação formal de um critério para validar consultas *CXPath* contra um modelo conceitual. Essa formalização é feita por meio de um conjunto de regras de inferência. Processo similar é utilizado para descrição de sistemas de tipos e semântica dinâmica de linguagens de programação (PIERCE, 2002).

Esse tipo de formalização vem sendo utilizado ultimamente pela comunidade da área de banco de dados em trabalhos relacionados (BIERMAN, 2003; SIMÉON; WADLER, 2003; WORLD WIDE WEB CONSORTIUM, 2003) que especificam linguagens de consulta com propósitos semelhantes.

A formalização resultante fornece uma descrição concisa e precisa da linguagem que pode ser usada como referência para implementadores e elaboradores de consultas globais. É também essencial para provar propriedades e resultados fundamentais como se a avaliação de uma consulta *CXPath* leva aos mesmos resultados da avaliação das consultas *XPath* produzidas pelo mecanismo de tradução.

Outra contribuição é a definição formal da operação de herança nas consultas *CXPath*. Apesar de a informação de herança estar presente no modelo conceitual, ela foi ignorada na definição original da linguagem. Essa formalização define a semântica dessa operação.

Como última contribuição desse trabalho, tem-se ainda que o mecanismo de tradução proposto no trabalho original foi estendido para permitir que a navegação por relacionamentos no modelo conceitual seja traduzida para junções em *XPath*. O trabalho anterior exigia que as consultas que necessitassem de junções sobre os dados fossem escritas explicitamente. Com a extensão proposta, as navegações são descritas previamente para o mecanismo de tradução. Portanto, ao consultar os dados, o mecanismo reescreve a consulta para os fontes criando as junções necessárias.

Este trabalho está organizado da seguinte forma. O Capítulo 2 apresenta o contexto no qual este trabalho foi desenvolvido. Ele apresenta o modelo conceitual, a linguagem *CXPath* de consulta e a o tipo de formalização escolhido para a descrição formal da linguagem *CXPath*. O Capítulo 3 apresenta a primeira parte da contribuição dessa dissertação, que consiste na inclusão do conceito de herança na linguagem e a formalização da linguagem *CXPath* por meio de um conjunto de regras de inferência, que especificam um critério para validação das consultas *CXPath* contra um modelo conceitual. O Capítulo 4 apresenta um conjunto de exemplos de como as consultas *CXPath* são validadas contra um modelo conceitual. O Capítulo 5 descreve como o mecanismo de tradução foi estendido para permitir consultas utilizando a operação de junção. O Capítulo 6 é dedicado à conclusão e a uma discussão sobre os trabalhos futuros.

2 TRABALHOS RELACIONADOS

Esta dissertação está inserida em um contexto de trabalhos anteriores que têm como objetivo permitir a integração de dados semi-estruturados. O trabalho que descreve o processo de construção de um modelo conceitual a partir de DTDs de documentos XML pode ser encontrado em (MELLO; HEUSER, 2001a,b). Uma descrição sucinta do modelo conceitual é apresentada na seção 2.1.

Outro trabalho que serve como base para esta dissertação é a descrição informal da linguagem *CXPath* (CAMILLO; HEUSER; MELLO, 2003). Esse trabalho apresenta a linguagem *CXPath - Conceptual XPath* e uma variante *CXQuery - Conceptual XQuery*. Apresenta também o processo de mapeamento de consultas escritas nessas linguagens para as consultas XML correspondentes (*XPath* e *XQuery*, respectivamente). Essa linguagem é apresentada na seção 2.2.

Sistemas de tipos (PIERCE, 2005) foi a técnica usada para descrever as características da linguagem *CXPath*. Regras de inferência são construídas com base na gramática da linguagem. Tais regras descrevem o comportamento da linguagem e como consultas escritas nessa linguagem podem ser validadas contra um esquema conceitual. Uma descrição da formalização usada nessa dissertação pode ser encontrada na seção 2.3.

2.1 Descrição do Modelo Conceitual

O modelo conceitual escolhido para a representação do esquema conceitual global resultante da integração de fontes XML suporta conceitos léxicos e não-léxicos, relacionamentos de associação e de herança (MELLO; HEUSER, 2001a).

Um conceito léxico modela uma informação que possui um conteúdo textual associado. São exemplos de conceitos claramente léxicos: idade e nome.

Um conceito não-léxico modela informação que é composta por outras informações. São exemplos de conceitos que possuem esse tipo de característica: universidade, que possui um nome, vários alunos, vários cursos.

Um relacionamento de associação é um relacionamento binário com restrições de cardinalidade. Nomes de relacionamentos e nomes de papéis podem ser opcionalmente definidos no modelo. Um exemplo de relacionamento de associação é o existente entre “Pessoa” e “Artigo”. Nesse caso, cada pessoa pode estar relacionada a vários artigos e cada artigo a várias pessoas.

Um relacionamento de herança é um relacionamento binário que descreve um conceito genérico e outro especializado. Informações adicionais a respeito do relacionamento de herança serão dadas na seção 3.1.

A representação formal do Modelo Conceitual-MC é dada pela tupla

(NL, L, RH, RA) onde:

- NL é o conjunto de conceitos não-léxicos
- L é o conjunto de conceitos léxicos
- RH é o conjunto dos relacionamentos de herança entre conceitos
- RA é o conjunto dos relacionamentos de associação entre conceitos

Um conceito especial chamado *Root* se distingue dos demais porque possui um relacionamento único com todos outros conceitos, tanto léxicos quanto não-léxicos. Portanto, é sempre possível navegar a partir do conceito *Root* para qualquer outro dentro do modelo conceitual. A Figura 2.1 demonstra como o conceito *Root* se relaciona com os demais em um modelo conceitual.

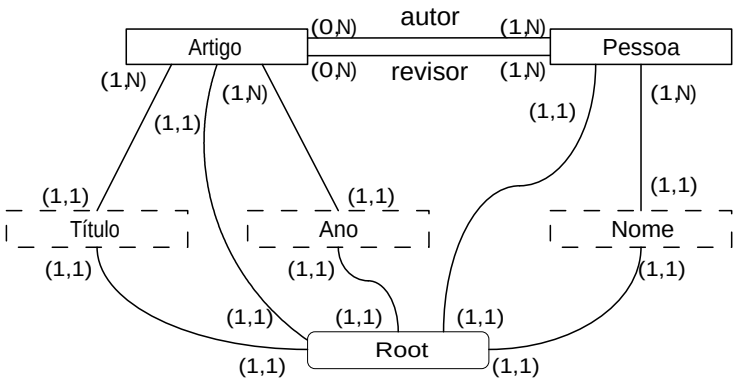

Figura 2.1: Modelo Conceitual com a representação do conceito *Root*

***Observação* 1** *Como mencionado anteriormente, o conceito Root está sempre presente em qualquer modelo conceitual. Por simplicidade de notação, nos próximos exemplos o conceito Root estará propositadamente omitido.*

Um conceito léxico é representado por um par (c, t), onde c representa o nome do conceito, e t representa o seu tipo.

Um relacionamento de herança dentro do conjunto RH é representado por um par (c_g, c_e), onde c_g representa o nome do conceito genérico e c_e representa o nome do conceito especializado. Ambos c_g e c_e devem ser conceitos não-léxicos pertencentes a NL. Essa restrição garante que somente conceitos pertencentes ao mesmo conjunto podem ser herdados. Esse é o mesmo comportamento dado por linguagens de programação que não permitem que tipos prmitivos sejam estendidos ou que sejam tipos especializados.

Um relacionamento de associação pertencente a RA é representado por uma tupla $(c_1, c_2, c_d, c_i, r, p_1, p_2)$, onde c_1 é nome do conceito de origem, c_2 é o nome do conceito de destino, c_d representa a cardinalidade direta (a partir de c_1 para c_2), c_i representa a cardinalidade inversa (a partir de c_2 para c_1) e r é o nome do

relacionamento. Quando a tupla representa um auto-relacionamento, ou seja c_1 é igual a c_2, tem-se que p_1 é o nome do papel desempenhado por c_1 no relacionamento e p_2 o nome do papel assumido por c_2.

A Figura 2.4 exemplifica um auto-relacionamento chamado de *casamento*. Uma pessoa pode estar relacionadas a outra pelo relacionamento *casamento*. Nessa situação, os nomes de papéis *marido* e *mulher* são necessários para identificar quem é o marido e a esposa.

***Observação* 2** *Com a finalidade de validar consultas CXPath contra um modelo conceitual, restrições de cardinalidade não são necessárias. Por essa razão, e para manter o tratamento formal simples, será omitida essa informação das tuplas que representam relacionamentos de associação.*

2.2 Descrição da Linguagem CXPath

CXPath (Conceptual XPath) (CAMILLO; HEUSER; MELLO, 2003) é uma linguagem para consultar uma base conceitual resultante da integração de fontes XML. Embora essa linguagem tenha sido criada para trabalhar com um modelo conceitual, tanto a linguagem como o processo de tradução não dependem exclusivamente desse modelo de dados particular e devem ser facilmente adaptáveis a outros variantes do modelo ER.

Um consulta *CXPath* define uma *expressão de caminho* no estilo *XPath 1.0* para obter a informação que deve ser alcançada, com predicados de seleção opcionais que podem ser definidos para aquele caminho. Uma consulta *CXPath* pertence ao conjunto definido pela gramática mostrada na Figura 2.2.

Embora baseada na sintaxe de *XPath 1.0*, *CXPath* e *XPath 1.0* possuem diferentes semânticas porque são aplicadas sobre diferentes modelos de dados. *XPath* é adequada para navegação em documentos XML, que possuem estruturas hierárquicas baseadas em árvore (WORLD WIDE WEB CONSORTIUM, 2005b). Por outro lado, *CXPath* é adequada para navegação sobre modelos conceituais, que possuem estruturas baseadas em grafo (CAMILLO; HEUSER; MELLO, 2003). Essas diferenças no modelo de dados acarretam várias diferenças na semântica das linguagens.

Essas diferenças são descritas a seguir:

- *nomes de conceitos ao invés de nomes de elementos*
  Em *XPath*, elementos XML são referenciados por seus rótulos (nomes de elementos). Em *CXPath*, nomes de conceitos são usados ao invés de nomes de elementos. Um nome de conceito se refere a todas instâncias desse conceito na base conceitual.

- *elemento raiz e expressões de caminho absolutas*
  Uma instância XML possui um *elemento raiz*. Uma expressão *XPath* que inicia com uma barra (uma *expressão de caminho absoluta*) inicia a navegação a partir do elemento raiz. O modelo conceitual é representado por um grafo, diferentemente da representação XML que é baseada em árvore. Nesse caso, não há um nodo (conceito) raiz para ser usado como ponto de partida. Nesse aspecto, todas expressões são relativas. Uma expressão absoluta tem a semântica de *navegue para o conceito "Root"*. Esse conceito "Root" está relacionado a todos conceitos do modelo. Portanto, sempre existe uma navegação possível

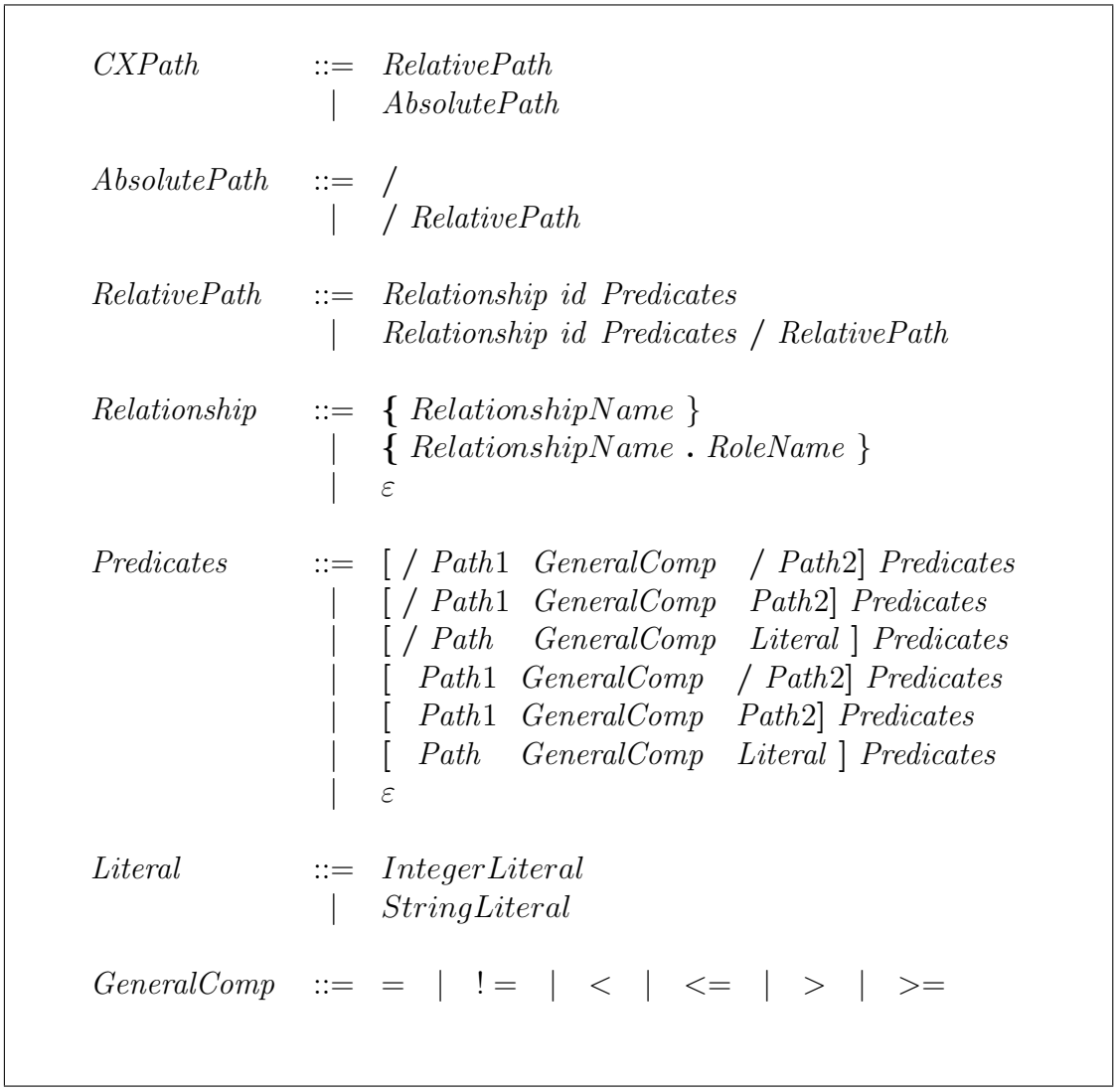

Figura 2.2: Gramática *CXPath*

de qualquer conceito para o "Root". E, a partir desse conceito, para qualquer outro do modelo também é possível pelo mesmo motivo.

A semântica *CXPath* para *expressão absoluta de caminho* é que a origem da navegação é o conceito "Root", que permite a navegação para qualquer outro conceito da base inteira. Embora o conceito "Root" possua um relacionamento com cada um dos demais conceitos do modelo, ele só pode ser navegado quando se tratar de uma *expressão absoluta de caminho*. Somente a barra no início de uma expressão *CXPath* implica navegação para esse conceito.

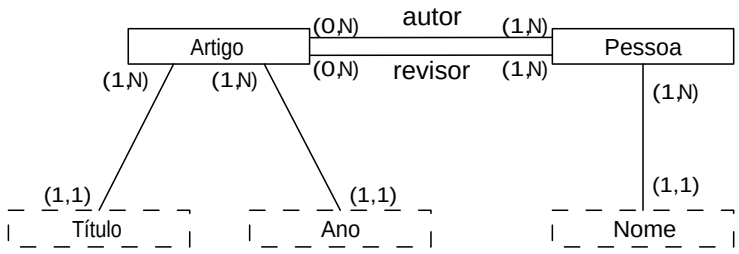

Figura 2.3: Modelo Conceitual com restrições de cardinalidade

Para os exemplos mostrados a seguir, considere o modelo conceitual apresentado na Figura 2.3.

Exemplo 1. Obter todas instâncias do conceito "*Artigo*".

```
/Artigo
```

- *operador de navegação (operador barra) e expressões de caminho relativas*
Em *XPath*, o operador barra, quando aparece em lugar diverso da posição inicial da expressão de caminho, possui a semântica de "navegue para os elementos filhos". Como o modelo conceitual possui a estrutura de um grafo e não uma árvore, essa semântica teve de ser adaptada para "navegue para os elementos relacionados".

  Para verificar se uma navegação entre dois conceitos é válida, é necessário analisar se existe tal relacionamento representado no modelo conceitual. Para isso, analisa-se se o conceito de origem e chegada estão representados entre os relacionamentos de associação.

  Exemplo 2. Obter todas instâncias do conceito "*Titulo*" que se relacionam com o conceito "*Artigo*".

```
/Artigo/Titulo
```

  Em *XPath*, uma expressão de caminho pode conter outras expressões de caminho. Por exemplo, a expressão de caminho `/Artigo[Ano="2004"]/Titulo` contém a expressão de caminho `Ano`. Esse é um caso de expressão de caminho relativa. O contexto de avaliação de expressões de caminho relativas é o conjunto de elementos filhos. Em *CXPath*, essa semântica foi alterada de "navegue para os elementos filhos" (*XPath*) para "navegue para os elementos relacionados" (*CXPath*).

Exemplo 3. Obter todas instâncias do conceito "*Titulo*" que se relacionam com as instâncias do conceito "*Artigo*" que são relacionadas com instâncias do conceito "*Ano*" com valor "2004".

```
/Artigo[Ano=2004]/Titulo
```

Nessa expressão de caminho, o contexto de avaliação da expressão de caminho `Ano` é o conceito *Artigo*. Portanto, a expressão de caminho `Ano` se refere ao conjunto de todas instâncias que estão relacionadas ao conceito *Artigo*.

- *operador de navegação qualificada - nome do relacionamento*
  Quando mais de um relacionamento relaciona dois conceitos, a identificação de um relacionamento específico para ser navegado pode ser necessária.

  Para verificar se uma navegação qualificada com nome de relacionamento existe, é necessário, além da verificação anterior, verificar se o relacionamento encontrado possui o nome do relacionamento desejado.

  Exemplo 4. Obter todos os nomes de pessoas que são autores de artigos produzidos no ano 2004:

  ```
  /Artigo[Ano=2004]/{autor}Pessoa/Nome
  ```

  Nesse exemplo, o operador de navegação qualificada "/{autor}" especifica a navegação a partir do conceito *Artigo* somente para aquelas instâncias do conceito *Pessoa* que se relacionam por meio do relacionamento *autor*.

- *operador de navegação qualificada - nome do papel*
  No caso de auto-relacionamentos, o nome do papel pode ser necessário para indicar a direção da navegação.

  Para verificar se uma navegação qualificada com nome de relacionamento e nome de papel é válida, é preciso, além das verificações anteriores, comparar se os conceitos de origem e partida possuem os papéis correspondentes.

  Exemplo 5. Considere o exemplo mostrado na Figura 2.4. Obter os nomes de todos os maridos:

  ```
  /Pessoa/{casamento.marido}Pessoa/Nome
  ```

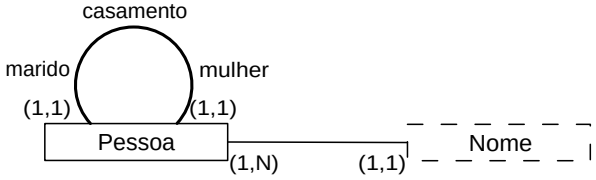

Figura 2.4: Representação de auto-relacionamentos

- *operadores hierárquicos*
  Todos operadores *XPath* que se referem explicitamente ou implicitamente a navegações para os antecessores ou descendentes não aparecem em *CXPath*. Como mencionado anteriormente, somente o operador barra foi mantido, mas com uma semântica diferente.

A descrição informal dessa linguagem não contempla a utilização de herança, conceito esse presente na descrição do modelo conceitual. Além disso, essa descrição não é baseada em critérios formais, por isso ela não pode ser utilizada para validar uma consulta escrita nessa linguagem contra um esquema conceitual. Essas restrições serviram de motivação para a extensão apresentada nesse trabalho.

2.3 Descrição da Formalização Utilizada

A engenharia de software moderna reconhece um vasto número de métodos formais para ajudar na verificação se um sistema se comporta corretamente em relação a alguma especificação, implícita ou explícita, de seu comportamento desejado.

Em um lado, há uma vasta gama de *frameworks* tais como lógica *Hoare* (PRATT, 1976), especificação algébrica de linguagens, lógica modal e semântica denotacional. Essas podem ser usadas para expressar a correção de várias propriedades gerais. No entanto, o seu uso é demasiado complicado e exige grande sofisticação por parte dos programadores para implementações.

Por outro lado, há técnicas cujo poder é bem mais modesto - modesto o suficiente de forma que verificadores automáticos podem ser construídos por meio de compiladores, ligadores e analisadores de programas. Por essa razão, o seu uso é facilitado e permite o uso mesmo por programadores não familiarizados com tais formalismos. O mais comum e utilizado com maior sucesso nessa gama de métodos formais leves é o de *sistemas de tipos*.

Um *sistema de tipos* é um método para provar formalmente a abstenção de certos comportamentos estáticos indesejados. Por ser uma verificação estática, um sistema de tipos pode provar formalmente a abstenção de certos comportamentos, mas não pode provar a sua presença. Nesse aspecto, algumas consultas seriam rejeitadas ainda que pudessem retornar resultados corretos quando executadas. Essa situação pode ser exemplificada por uma consulta que possua dois predicados ligados por um "OU" lógico ($< Teste_1 > \vee < Teste_2 >$). Ainda que o primeiro teste seja verdadeiro em tempo de execução, ambos os testes passarão pela verificação de tipos e, caso o segundo teste contenha um erro de tipo, a consulta será rejeitada.

Sistemas de tipos são utilizados em vários tipos de aplicações, dentre elas pode-se destacar as que seguem. A *detecção de erros* precoce permite que os erros sejam corrigidos mais facilmente. Consultas que não estão de acordo com determinado esquema devem ser rejeitadas sem sequer serem avaliadas por não fazerem sentido. Essa situação é mais confortável do que obter um conjunto resultado vazio e tentar entender se a base é que não possuía dados, os predicados não retornavam resultados ou a consulta é que não estava bem formada.

Outro importante uso dos sistemas de tipos é auxiliar na definição de *abstrações de dados* forçando uma disciplina na programação. A construção de softwares é uma tarefa que envolve grandes abstrações. Conceitos são mapeados para estruturas na memória de computadores e seu uso irrestrito conduz facilmente a erros. A partir da definição de tipos para essas estruturas, o verificador pode analisar se o programador

se enganou em alguma referência deixando de converter algum tipo de dado ou tentou acessar alguma estrutura que não existe.

Além da maior abstração, o uso de um sistema de tipos ajuda na *documentação* de um sistema. Estruturas identificadas por meio dos tipos fornecem uma boa dica de seu uso. A definição de tipos é mais eficiente do que os comentários embutidos no próprio código, pois uma alteração no tipo dos dados pode tornar os comentários sem utilidade, mas o sistema de tipos forçosamente acusará um tipo ilegal evitando ficar obsoleto.

Muitas outras aplicações (HOSOYA; VOUILLON; PIERCE, 2000; PERST; SEIDL, 2002), além do tradicional uso em linguagens de programação, têm utilizado sistemas de tipos no ramo da Ciência da Computação. Uma área de aplicação que vem crescendo em importância é a de segurança de redes e computadores. Sistemas de tipos são utilizados no núcleo do modelo de segurança de Java (FREUND; MITCHELL, 2003) e da arquitetura JINI de dispositivos de rede. Além disso, no contexto das linguagens de programação, sistemas de tipos são utilizados como analisadores de tipos. No mesmo sentido, há trabalhos que visam a aplicar a teoria de linguagens de programação diretamente a problemas no domínio de segurança (MCKINNA; POLLACK, 1993).

Nas provas da teoria de autômatos (HOSOYA; PIERCE, 2001), sistemas de tipos são usados para representar proposições lógicas e provas.

O interesse em sistemas de tipos é crescente também na comunidade de banco de dados, com a explosão de "metadados *web*" na forma de DTDs e outros tipos de esquemas tais como o padrão XML Schema (WORLD WIDE WEB CONSORTIUM, 2005c) para descrição de dados XML estruturados. Novas linguagens para consultar e manipular XML (HOSOYA; PIERCE, 2000; CHAMBERLIN; ROBIE; FLORESCU, 2001; MILO; SUCIU; VIANU, 2003) provêm sistemas de tipos estáticos baseados diretamente nas linguagens de descrição de esquemas.

A notação usada nas regras de inferência do capítulo 3 deve ser entendida como é mostrado a seguir. A descrição de um sistema de tipos de uma linguagem começa, usualmente, pela definição de axiomas que associam um tipo a cada expressão atômica da linguagem. Um axioma tem a forma:

$$\Gamma \vdash \ e : \sigma$$

onde e é uma expressão atômica, σ é um tipo e Γ é um *contexto de tipo*. Esse axioma determina que a expressão e possui tipo σ no contexto Γ.

A forma geral de uma *regra de inferência* do sistema de tipos é:

$$\frac{\Gamma_1 \ \vdash \ e_1 : \sigma_1 \qquad \cdots \qquad \Gamma_n \ \vdash \ e_n : \sigma_n}{\Gamma \ \vdash \ e \ : \sigma} \qquad \text{(Regra1)}$$

onde:

- o nome da regra de inferência é mostrado à direira da regra;
- $\Gamma_i \vdash e_i : \sigma_i$, para $i = 1, \ldots, n$, são as hipóteses da regra;
- e $\Gamma \vdash e : \sigma$ é a conclusão;
- todas hipóteses precisam ser satisfeitas para verificar a validade da conclusão.

Intuitivamente, essa regra diz que, se para cada expressão e_1 tem tipo σ_i no contexto Γ_i, para $i = 1, \ldots, n$, então e tem tipo σ no contexto Γ.

Um expressão e é dita bem tipada em um dado contexto Γ, com relação a um sistema de tipos, se existe uma derivação de $\Gamma \vdash e : \sigma$, para algum σ,obtida mediante os axiomas e regras desse sistema de tipos.

Essa é a notação usualmente dada para sistemas de tipos usando regras de inferência. Neste trabalho, entretanto, nem sempre a verificação de tipagem é necessária.

Adicionalmente, quando o símbolo $\vdash$ for identificado, por exemplo por $\vdash_{APE}$ significa que o conjunto de regras que irá validar a expressão à direita do símbolo deve levar em conta o conjunto de regras por esse símbolo especificado. Portanto, o símbolo $\vdash$ pode ser analogamente entendido como uma função que verifica se a expressão à sua direita é válida e que recebe como argumentos a expressão a ser avaliada e o contexto de avaliação, indicada pelos conceitos à esquerda do símbolo.

3 FORMALIZAÇÃO DA LINGUAGEM

Esse capítulo apresenta a primeira parte do trabalho realizado nessa dissertação. Esse trabalho consiste na formalização da linguagem e na sua extensão para permitir herança nas expressões de consulta. A seção 3.1 descreve informalmente o conceito de herança introduzido na linguagem. A seção 3.2 descreve as regras que definem a semântica da uma consulta escrita nessa linguagem.

3.1 Inclusão de Herança em CXPath

Apesar de a linguagem *CXPath* permitir consultas sobre um modelo conceitual, algumas restrições impostas pela definição original da linguagem serviram de base para este trabalho.

A linguagem foi estendida para permitir que as consultas utilizassem herança, recurso disponível no modelo conceitual, mas não utilizado na definição original da linguagem.

Além disso, a definição original da linguagem não seguiu um rigor formal. Por essa razão, este trabalho tem como um de seus objetivos formalizar a linguagem e verificar se uma consulta é válida de acordo com a definição do modelo conceitual contra o qual a consulta foi elaborada.

A definição formal de herança na linguagem *CXPath* foi feita com o objetivo de permitir a navegação por conceitos que não necessariamente possuam relacionamentos diretos, mas por meio de seus ancestrais, ou seja, dos conceitos dos quais estes descendem.

A representação do relacionamento de herança no modelo conceitual é feita por um triângulo, como mostrado na figura 3.1. Esta figura ilustra a modelagem de um relacionamento de herança onde *Pessoa* representa o conceito genérico e *Autor* e *Revisor* são os conceitos especializados.

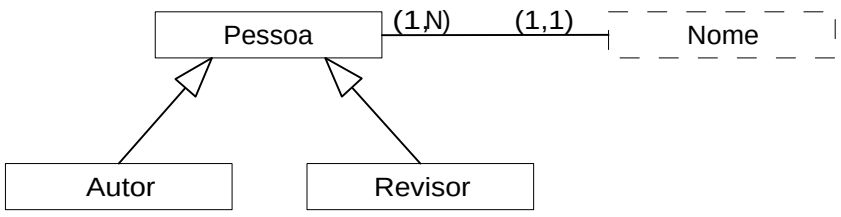

Figura 3.1: Representação do relacionamento de herança

Quando um conceito herda de outro, todos os relacionamentos são herdados junto. Nessa situação, caso um relacionamento específico não possua o relacionamento que se deseja utilizar, é necessário pesquisar em todos os relacionamentos de cada um dos conceitos mais genéricos.

Considerando o exemplo ilustrado na Figura 3.1, o conceito Autor não possui relacionamento direto com o conceito Nome. Contudo, esse conceito pode ser alcançado por meio do relacionamento existente entre o conceito Pessoa, do qual o conceito Autor herda os seus relacionamentos.

A consulta *CXPath*:

```
/Autor/Nome
```

é válida em CXPath, pois `Autor` herda todos os relacionamentos do conceito `Pessoa` com o qual possui um relacionamento de herança.

3.2 Formalização da Linguagem CXPath

Uma definição de *CXPath* concisa e precisa é necessária para ambos usuários e implementadores da linguagem. Um dos objetivos deste trabalho é formalizar os aspectos estáticos de sua semântica. Neste capítulo serão apresentadas regras de inferência que estabelecem as condições que devem ser satisfeitas para que uma consulta *CXPath* seja válida em relação a um modelo conceitual.

Uma linguagem de consulta não é nada mais do que uma linguagem de programação com propósitos especiais (DATE, 1984). Nesse sentido, o problema da validação da linguagem contra um esquema conceitual é semelhante ao problema da verificação de tipagem de dados em uma linguagem de programação. Por essa razão, para a formalização da linguagem serão adotadas regras de inferência para descrever a semântica da linguagem.

Essa técnica foi também escolhida porque é usada na definição formal da linguagem *XPath* e de sua especificação semântica (WORLD WIDE WEB CONSORTIUM, 2003), a qual descreve a semântica estática e dinâmica das linguagens *XPath* e *XQuery* (WORLD WIDE WEB CONSORTIUM, 2005d).

O termo *CXPath* denota uma expressão de consulta *CXPath* que segue a gramática mostrada na Figura 2.2 da página 18. Essa expressão será confrontada contra um modelo conceitual. Para verificar se uma dada expressão *CXPath* está semanticamente correta em relação a um modelo conceitual *CM*, é necessário verificar se se trata de uma expressão absoluta de caminho ou de uma expressão relativa.

Na gramática apresentada na Figura 2.2, a expressão *CXPath* pode ser tanto uma expressão absoluta de caminho *AbsolutePath* ou uma expressão relativa de caminho *RelativePath*. A seguir são apresentadas duas regras mostrando simplesmente que cada uma das possibilidades deve ser avaliada segundo o seu conjunto próprio de regras.

$$\frac{CM \ \vdash_{APE} \ AbsolutePath}{CM \ \vdash_{CXP} \ AbsolutePath} \quad \text{(CXP1)}$$

$$\frac{CM \ \vdash_{RPE} \ RelativePath}{CM \ \vdash_{CXP} \ RelativePath} \quad \text{(CXP2)}$$

3.2.1 Expressões de Caminho Absolutas

Expressões de caminho absoluta servem para indicar, a partir de qualquer contexto, uma expressão de caminho que não possui vínculo com o contexto em que está localizada. Dessa forma, não é necessário saber o caminho partindo daquele contexto até o desejado, mas apenas a expressão de caminho que descreve a informação a ser buscada.

Uma expressão de caminho absoluta inicia com a "/" e sua semântica é: *navegue para o conceito Root* . Qualquer expressão de caminho que seguir o conceito *Root* pode ser considerada como uma expressão de caminho relativa. Como mencionado anteriormente, o conceito *Root* possui um relacionamento não-identificado com todos outros conceitos do modelo, portanto sempre existe um caminho que permita a navegação a partir do conceito *Root* para qualquer outro conceito.

Há duas regras para validar as expressões de caminho absolutas: uma para cada cláusula na gramática mostrada na Figura 2.2 para *AbsolutePath* ("/" e "/ *RelativePath*").

Pela regra APE1 abaixo, a expressão de caminho absoluta "/" é sempre considerada bem-formada. Pela regra APE2, uma expressão de caminho absoluta com a forma "/ *RelativePath*" será bem formada quando a parte relativa da consulta for considerada como uma expressão de caminho relativa bem-formada usando o conjunto de regras que definem $\vdash_{RPE}$ e quando a "/" sozinha for considerada bem-formada (como visto anteriormente, a "/¨ sozinha é sempre bem-formada).

$$\frac{}{CM \ \vdash_{APE} \ /} \qquad \text{(APE1)}$$

$$\frac{CM, Root \vdash_{RPE} RelativePath}{CM \ \vdash_{APE} \ /RelativePath} \qquad \text{(APE2)}$$

3.2.2 Expressões de Caminho Relativas

Há duas cláusulas na gramática da Figura 2.2 para expressões de caminho relativas. Abaixo é mostrada uma regra de validação para cada uma das possibilidades. A regra RPE1 verifica se uma expressão de caminho relativa está no formato "*Relationship Id Predicates*".

$$\frac{CM, \ id_1, \ id_2 \vdash_{REL} \ Relationship \qquad CM, \ id_2 \vdash_{PRE} \ Predicates}{CM, \ id_1 \vdash_{RPE} \ Relationship \ id_2 \ Predicates} \qquad \text{(RPE1)}$$

Se existirem outras expressões de caminho seguindo a primeira, a próxima expressão de caminho deve ser avaliada levando em consideração o contexto da expressão anterior. O contexto é determinado ao avaliar qual é o último *Id* da expressão anterior.

$$\frac{\begin{array}{c} CM, \ id_1, \ id_2 \vdash_{REL} \ Relationship \\ CM, \ id_2 \vdash_{PRE} \ Predicates \\ CM, \ id_2 \vdash_{RPE} \ RelativePath \end{array}}{CM, \ id_1 \vdash_{RPE} Relationship \ id_2 \ Predicates \ / \ RelativePath} \qquad \text{(RPE2)}$$

Importante notar que as primeiras duas premissas da regra RPE2 são as mesmas da regra RPE1. A terceira premissa válida *RelativePath* contra o modelo conceitual e avaliando o último identificador (conceito) das expressões de caminho anteriores.

3.2.3 Relacionamentos

A notação $id_1 \prec id_2$ mostrada nas regras seguintes significa o relacionamento de herança, onde id_1 é o conceito especializado e id_2 é o conceito genérico.

A regra INH1 a seguir mostra quando $id_1 \prec id_2$ está representada no modelo conceitual CM (premissa da regra INH1). As regras INH2 e INH3 mostram que a relação $\prec$ é reflexiva e transitiva.

$$\frac{(id_1,\ id_2) \ \in \ CM_{RH}}{CM \ \vdash \ id_1 \ \prec \ id_2} \quad \text{(INH1)}$$

$$\frac{}{CM \vdash \ id \prec \ id} \quad \text{(INH2)}$$

$$\frac{CM \vdash \ id_1 \ \prec \ id_2 \qquad CM \vdash \ id_2 \ \prec \ id_3}{CM \vdash \ id_1 \ \prec \ id_3} \quad \text{(INH3)}$$

A verificação se um relacionamento entre conceitos numa expressão de caminho é válido é feita verificando-se três possíveis situações. A primeira situação verifica relacionamentos identificados $\{RelationshipName\}$; a segunda, além de verificar relacionamento identificados, verifica nomes de papéis $\{RelationshipName_{\cdot}RoleName\}$; a terceira quando não há nome de relacionamento tampouco nome de papel (ε).

Por simplicidade, a representação dos relacionamentos em um modelo conceitual é feita indicando-se um conceito origem e um conceito destino. No entanto, a navegação pode ser feita em qualquer direção. Portanto, o conceito origem pode atuar tanto como origem quanto como destino, e o conceito destino igualmente. Por essa razão, todas regras devem levar isso em consideração e sempre verificar se existe um relacionamento partindo do conceito origem (ou do destino) e se existe um conceito correspondente de destino (ou origem) que combine com os conceitos relacionados na expressão de caminho.

Há uma importante restrição que deve ser observada. Não é possível navegar de qualquer conceito para o conceito *Root*. Qualquer relacionamento que envolva o conceito *Root* só é possível quando este atuar como conceito origem, com qualquer outro conceito atuando como destino. Portanto, a única maneira de navegar para esse conceito é quando se tratar de uma expressão de caminho absoluta.

As regras que seguem, mostram que qualquer navegação a partir do conceito *Root* para qualquer outro só serão possíveis se existir um relacionamento não identificado entre eles na consulta. Se um nome de relacionamento aparecer na consulta, a consulta será rejeitada porque só existem relacionamentos não-identificados partindo do conceito *Root* para qualquer outro conceito.

$$\frac{\begin{array}{c} CM \ \vdash id_1 \prec id_{g_1} \\ CM \ \vdash id_2 \prec id_{g_2} \\ (c_1, c_2, RelationshipName, p_1, p_2) \ \in \ CM_{RA} \\ (c_1 = id_{g_1} \ \wedge \ c_2 = id_{g_2}) \ \vee \ (c_1 = id_{g_2} \ \wedge \ c_2 = id_{g_1}) \end{array}}{CM, id_1, id_2 \vdash_{REL} \ \{ \ RelationshipName \ \}} \quad \text{(REL1)}$$

Todo relacionamento identificado que possuir um nome de papel deve obrigatoriamente ser considerado um auto-relacionamento. Nesse caso, o conceito de origem e o conceito de destino devem ser o mesmo.

$$\frac{\begin{array}{c} id_1 = id_2 \\ (id_1, id_2, RelationshipName, p_1, p_2) \quad \in \quad CM_{RA} \\ p_1 = RoleName \quad \vee \quad p_2 = RoleName \end{array}}{CM, id_1, id_2 \vdash_{REL} \{\ RelationshipName\ .\ RoleName\ \}} \quad (\text{REL}2)$$

Quando houver um relacionamento não-identificado, é necessário que exista tal relacionamento não-identificado no modelo conceitual. Essa restrição visa a garantir que não exista uma situação onde mais de uma caminho seria possível de ser escolhido. Portanto, na construção do modelo conceitual é necessário fornecer a interpretação para relacionamentos não-identificados se esses serão usados nas consultas.

$$\frac{\begin{array}{c} id_1 \prec id_{g_1} \\ id_2 \prec id_{g_2} \\ (c_1,\ c_2,\ \varepsilon, p_1, p_2) \quad \in \quad CM_{RA} \\ (c_1 = id_{g_1} \ \wedge \ c_2 = id_{g_2}) \ \vee \ (c_1 = id_{g_2} \ \wedge \ c_2 = id_{g_1}) \end{array}}{CM,\ id_1,\ id_2 \vdash_{REL} \varepsilon} \quad (\text{REL}3)$$

3.2.4 Predicados

O predicado de seleção válido para expressões *CXPath* deve retornar *verdadeiro* ou *falso*, indicando uma condição satisfeita ou não-satisfeita. Esses valores são os resultados de comparações entre expressões ou entre expressões e literais.

Um predicado para ser considerado bem-formado precisa possuir como último conceito, antes do predicado, um conceito léxico. Essa restrição se deve ao fato de que os conceitos léxicos possuem um tipo associado, portanto é possível verificar se a comparação está usando tipos compatíveis.

As primeiras três regras são para predicados que iniciam com uma expressão de caminho absoluta como operador do lado esquerdo de *GeneralComp*. As três seguintes são para predicados que inciam com uma expressão de caminho relativa. A última regra é para o caso de predicado vazio cuja avaliação é sempre verdadeira.

$$\frac{\begin{array}{c} LastId(RelativePath_1) \ \in \ CM_L \\ LastId(RelativePath_2) \ \in \ CM_L \\ CM, Root \ \vdash_{RPE} \ RelativePath_1 \\ CM, Root \ \vdash_{RPE} \ RelativePath_2 \\ CM, id \vdash_{PRE} \ Predicates \\ CM \ \vdash \ LastId(RelativePath_1)\ GeneralComp\ LastId(RelativePath_2) \end{array}}{CM, id \vdash_{PRE} \ [\ /\ RelativePath_1\ GeneralComp\ /\ RelativePath_2]\ Predicates} \quad (\text{PR}1)$$

$$\frac{\begin{array}{c} LastId(RelativePath_1) \ \in \ CM_L \\ LastId(RelativePath_2) \ \in \ CM_L \\ CM, Root \ \vdash_{RPE} \ RelativePath_1 \\ CM, id \vdash_{RPE} \ RelativePath_2 \\ CM, id \vdash_{PRE} \ Predicates \\ CM \ \vdash \ LastId(RelativePath_1)\ GeneralComp\ LastId(RelativePath_2) \end{array}}{CM, id \vdash_{PRE} \ [\ /\ RelativePath_1\ GeneralComp\ RelativePath_2]\ Predicates} \quad (\text{PR}2)$$

$$\frac{\begin{array}{c} LastId(RelativePath) \in CM_L \\ CM, Root \vdash_{RPE} RelativePath \\ CM, id \vdash_{PRE} Predicates \\ CM \vdash LastId(RelativePath)\ GeneralComp\ Literal \end{array}}{CM, id \vdash_{PRE} [\ /\ RelativePath\ GeneralComp\ Literal]\ Predicates} \quad (\text{PR3})$$

$$\frac{\begin{array}{c} LastId(Path_1) \in CM_L \\ LastId(Path_2) \in CM_L \\ CM, id \vdash_{PE} Path_1 \\ CM, Root \vdash_{PE} Path_2 \\ CM, id \vdash_{PRE} Predicates \\ CM \vdash LastId(Path_1)\ GeneralComp\ LastId(Path_2) \end{array}}{CM, id \vdash_{PRE} [\ Path_1\ GeneralComp\ /\ Path_2]\ Predicates} \quad (\text{PR4})$$

$$\frac{\begin{array}{c} LastId(Path_1) \in CM_L \\ LastId(Path_2) \in CM_L \\ CM, id \vdash_{PE} Path_1 \\ CM, id \vdash_{PE} Path_2 \\ CM, id \vdash_{PRE} Predicates \\ CM \vdash LastId(Path_1)\ GeneralComp\ LastId(Path_2) \end{array}}{CM, id \vdash_{PRE} [\ Path_1\ GeneralComp\ Path_2]\ Predicates} \quad (\text{PR5})$$

$$\frac{\begin{array}{c} LastId(Path_1) \in CM_L \\ CM, id \vdash_{PE} Path_1 \\ CM, id \vdash_{PRE} Predicates \\ CM \vdash LastId(Path_1)\ GeneralComp\ Literal \end{array}}{CM, id \vdash_{PRE} [\ Path_1\ GeneralComp\ Literal]\ Predicates} \quad (\text{PR6})$$

$$\frac{}{CM \vdash_{PRE} \varepsilon} \quad (\text{PR7})$$

Estas regras utilizam a seguinte função auxiliar, chamada *LastId*, a qual simplesmente retorna o último nome de conceito antes do predicado de uma consulta *CXPath*:

$$\begin{array}{lcl} LastId(Relationship\ id\ Predicates) & = & id \\ LastId(Relationship\ id\ Predicates/\ Path) & = & LastId(Path) \end{array}$$

3.2.5 Comparações

As comparações são operadores que relacionam dois conceitos léxicos ou um conceito léxico e um valor literal. Uma comparação de dois conceitos léxicos é bem-formada em relação a um modelo conceitual somente se ambos termos envolvidos possuírem o mesmo tipo:

$$\frac{(id_1, t_1) \in CM_L \qquad (id_2, t_2) \in CM_L \qquad t_1 = t_2}{CM, \vdash id_1\ GeneralComp\ id_2} \quad (\text{COMP1})$$

Nas regras abaixo para validação entre um conceito léxico e um literal e entre dois literais, são usados s e n como meta-variáveis para indicar literais do tipo *string* e literais do tipo *integer* respectivamente.

$$\frac{(id, integer) \in CM_L}{CM \vdash id\ GeneralComp\ n} \quad (\text{COMP}2) \qquad \frac{(id, string) \in CM_L}{CM \vdash id\ GeneralComp\ s} \quad (\text{COMP}3)$$

$$\frac{}{CM \vdash n_1\ GeneralComp\ n_2} \quad (\text{COMP}4) \qquad \frac{}{CM \vdash s_1\ GeneralComp\ s_2} \quad (\text{COMP}5)$$

Para os demais tipos de dados as regras seguem a mesma organização. O que define se determinado token corresponde a um determinado tipo, para verificar se é enquadrado em cada uma das regras, é a implementação.

Aqui, considera-se o tratamento usual para esses tipos, ou seja, um tipo *string* é um conjunto de caracteres ou dígitos cercado para aspas duplas; um tipo *integer*, uma seqüência de dígitos. A aplicação pode definir regras diferenciadas para identificação desses tipos e dos demais que julgar necessários.

4 VALIDAÇÃO DA CONSULTA CONTRA UM ESQUEMA CONCEITUAL

Neste capítulo serão apresentados alguns exemplos de consultas *CXPath* e como as regras mostradas na seção 3.2 podem ser combinadas para validar as consultas contra um esquema conceitual.

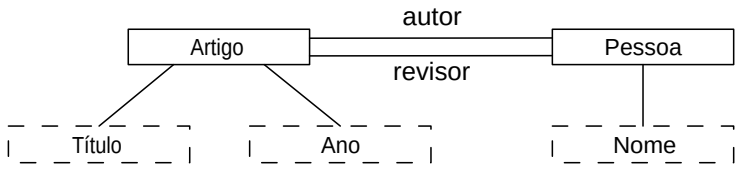

Figura 4.1: Modelo conceitual simplificado dos exemplos de validação

O modelo conceitual apresentado na Figura 4.1 é uma versão simplificada do modelo representado nos capítulos anteriores. Por simplicidade, esse modelo não possui a representação do conceito *Root* nem as informações de cardinalidade, estas inúteis para validação das consultas. Esse será o modelo utilizado para validação das consultas nos exemplos ilustrados a seguir.

4.1 Exemplo 1

A consulta *CXPath*

```
/Artigo
```

obtém todas instâncias do conceito `Artigo`. Essa consulta é validada usando as seguintes regras:

CXP1 A consulta "/ `Artigo`" inicia com uma "/", portanto o conjunto de regras para validar essa parte da consulta será o de expressões de caminho absolutas.

APE2 Para expressões de caminho absolutas existem duas possibilidades: a primeira quando se trata apenas de um /, a outra possibilidade é quando se trata de uma "/" seguido por uma expressão de caminho relativa. Este é o caso e portando a premissa da regra *APE2* deve ser satisfeita. Como se trata de uma expressão absoluta, ela deve ser avaliada utilizando o conjunto de regras para expressões relativas de caminho utilizando o contexto *Root*.

RPE1 Para que uma expressão de caminho relativa seja considerada bem-formada, é necessário que ela satisfaça as premissas da regra em que ela se enquadra.

No caso desse exemplo, a regra *RPE1* possui duas premissas que devem ser analisadas. A primeira premissa verifica se existe um relacionamento não-identificado entre o conceito que descreve o contexto e o primeiro conceito da expressão de caminho. A segunda premissa verifica se o predicado é válido passando o contexto em que esse deve ser avaliado.

REL3 Como existe um relacionamento não-identificado entre *Root* e o conceito `Artigo`, é necessário averigüar se as premissas dessa regra são satisfeitas. As duas primeiras premissas servem para verificar se existe um relacionamento entre os conceitos mais genéricos. Essa avaliação é feita com a regra *INH1*. A terceira e quarta premissas avaliam se existe um relacionamento não identificado entre o conceito *Root* e o conceito *Artigo*. Como dito anteriormente, sempre existe um relacionamento não-identificado entre o conceito *Root* e qualquer outro do modelo. Portanto essas premissas também são satisfeitas.

INH1 Para verificar se cada um dos conceitos possui relacionamentos é necessário verificar se os mais genéricos possuem relacionamentos. Mesmo não sendo o caso, as premissas são verdadeiras porque a operação de herança é reflexiva.

PR7 A última etapa da verificação é analisar se o predicado é válido. A regra não contém premissas e portanto é sempre verdadeira.

Como todas as regras foram validadas, a consulta obedece a todos critérios especificados e a consulta é considerada bem-formada em relação ao modelo conceitual.

4.2 Exemplo 2

A consulta *CXPath* seguinte retorna o título de todos artigos:

```
/Artigo/Título
```

Essa consulta é validada usando as seguintes regras:

CXP1 A consulta "`/ Artigo / Título`" inicia com uma "/", portanto o conjunto de regras para validar essa parte da consulta será o de expressões de caminho absolutas.

APE2 Assim como no exemplo anterior, para expressões de caminho absolutas existem duas possibilidades: a primeira quando se trata apenas de uma "/", a outra possibilidade é quando se trata de uma "/"seguida por uma expressão de caminho relativa. Este é o caso e portando a premissa da regra *APE2* deve ser satisfeita. Como se trata de uma expressão absoluta, ela deve ser avaliada utilizando o conjunto de regras para expressões relativas de caminho utilizando o contexto *Root*.

RPE2 O fato da consulta ser composta por duas parte (`Artigo` e `Título`) exige que seja utilizada a regra *RPE2*. A primeira premissa dessa regra verifica se existe um relacionamento não-identificado entre o conceito do contexto e o primeiro conceito da consulta. A segunda premissa verifica se o predicado do primeiro conceito da consulta é válido. A última premissa dessa regra verifica se a parte seguinte da consulta é válida utilizando-se como contexto o primeiro conceito da consulta (*Artigo*).

REL3 Como existe um relacionamento não-identificado entre *Root* e o conceito *Artigo*, é necessário averigüar se as premissas dessa regra são satisfeitas. As duas primeiras premissas servem para verificar se existe um relacionamento entre os conceitos mais genéricos. Elas são avaliadas pela regra *INH1*. A terceira e quarta premissas avaliam se existe um relacionamento não identificado entre o conceito *Root* e o conceito *Artigo*. Para esse caso, todas premissas são satisfeitas.

INH1 Para verificar se cada um dos conceitos possui relacionamentos é necessário verificar se os mais genéricos possuem relacionamentos. Mesmo não sendo o caso, as premissas são verdadeiras porque a operação de herança é reflexiva.

PR7 A verificação se o predicado é válido é trivial, pois o predicado é vazio e essa regra não contém premissas. Portanto a regra é verdadeira.

RPE1 Resta verificar se a segunda parte da consulta é válida. Para isso, analisa-se as premissas da regra. A primeira premissa verifica se o contexto, que agora é o conceito anterior (*Artigo*), é válido. Para isso, mais uma vez é necessário invocar a regra *REL3*. A segunda premissa verifica se o predicado é válido por meio da regra *PR7*.

REL3 Como existe um relacionamento não-identificado entre o conceito *Artigo* e o conceito *Título*, é necessário averigüar se as premissas dessa regra são satisfeitas. As duas primeiras premissas servem para verificar se existe um relacionamento entre os conceitos mais genéricos. Elas são avaliadas pela regra *INH1*. A terceira e quarta premissas avaliam se existe um relacionamento não-identificado entre o conceito *Artigo* e o conceito *Título*. Para esse caso, todas premissas são satisfeitas.

INH1 Para verificar se cada um dos conceitos possui relacionamentos é necessário verificar se os mais genéricos possuem relacionamentos. Mesmo não sendo o caso, as premissas são verdadeiras porque a operação de herança é reflexiva.

PR7 A verificação se o predicado é válido é trivial, pois o predicado é vazio e essa regra não contém premissas. Portanto a regra é verdadeira.

Como todas as regras foram validadas, a consulta obedece a todos critérios especificados e a consulta é considerada bem-formada em relação ao modelo conceitual.

4.3 Exemplo 3

Obter todas instâncias do conceito "*Título*" que se relacionam com as instâncias do conceito "*Artigo*" que são relacionadas com instâncias do conceito "*Ano*" com valor 2004.

```
/Artigo[Ano=2004]/Título
```

Essa consulta é validada seguindo a seguinte ordem de regras:

CXP1 A consulta "`/ Artigo[Ano="2004"]/ Título`" inicia com uma "/", portanto o conjunto de regras para validar essa parte da consulta será o de expressões de caminho absolutas.

APE2 Como não se trata de um /isolado, utiliza-se a regra *APE2* que possui como premissa a avaliação da expressão de caminho relativa que segue assumindo como contexto o conceito *Root*.

RPE2 O fato da consulta ser composta por duas parte (`Artigo` e `Título`) exige que seja utilizada a regra *RPE2*. A primeira premissa dessa regra verifica se existe um relacionamento não-identificado entre o conceito do contexto e o primeiro conceito da consulta. A segunda premissa verifica se o predicado do primeiro conceito da consulta é válido. A última premissa dessa regra verifica se a parte seguinte da consulta é válida utilizando-se como contexto o primeiro conceito da consulta (*Artigo*).

REL3 Como existe um relacionamento não-identificado entre *Root* e o conceito `Artigo`, é necessário averigüar se as premissas dessa regra são satisfeitas. As duas primeiras premissas servem para verificar se existe um relacionamento entre os conceitos mais genéricos. Elas são avaliadas pela regra *INH1*. A terceira e quarta premissas avaliam se existe um relacionamento não identificado entre o conceito *Root* e o conceito *Artigo*. Para esse caso, todas premissas são satisfeitas.

INH1 Para verificar se cada um dos conceitos possui relacionamentos é necessário verificar se os mais genéricos possuem relacionamentos. Mesmo não sendo o caso, as premissas são verdadeiras porque a operação de herança é reflexiva.

PR6 A verificação do predicado é dada pela regra *PR6*, pois a consulta faz uma comparação entre uma expressão de caminho relativa `Ano` e um literal inteiro. Para isso a função *LastId(Ano)* retorna o próprio *(*Ano), indicando que esse é o último conceito da expressão de caminho relativa. Como esse conceito pertence ao conjunto de conceitos léxicos, essa premissa é então satisfeita. A segunda premissa indica que é necessário avaliar a expressão relativa em relação ao contexto em que ela se encontra. Nesse caso, equivale a verificar se existe um relacionamento não-identificado entre *Artigo* e *Ano*. Para isso a regra *REL3* é novamente chamada para avaliá-la. A terceira premissa verifica o próximo predicado, para isso invoca a regra *PR7*. Por fim, a última premissa invoca a regra *COMP2* para avaliar se o conceito léxico `Ano` e o literal inteiro possuem o mesmo tipo.

REL3 Como existe um relacionamento não-identificado entre o conceito *Artigo* e o conceito *Ano*, é necessário averigüar se as premissas dessa regra são satisfeitas. As duas primeiras premissas servem para verificar se existe um relacionamento entre os conceitos mais genéricos. Elas são avaliadas pela regra *INH1*. A terceira e quarta premissas avaliam se existe um relacionamento não-identificado entre o conceito *Artigo* e o conceito *Ano*. Para esse caso, todas premissas são satisfeitas.

INH1 Para verificar se cada um dos conceitos possui relacionamentos é necessário verificar se os mais genéricos possuem relacionamentos. Mesmo não sendo o caso, as premissas são verdadeiras porque a operação de herança é reflexiva.

PR7 A verificação se o predicado é válido é trivial, pois o predicado é vazio e essa regra não contém premissas. Portanto a regra é verdadeira.

COMP2 O valor literal 2004 corresponde a um tipo inteiro. Por isso, é necessário avaliar se o mesmo tipo está indicado no modelo conceitual para o conceito *Ano*. Como no modelo esse é o tipo indicado a regra é validada.

RPE1 Resta verificar se a segunda parte da consulta (`Título`) é válida. Para isso, analisa-se as premissas da regra. A primeira premissa verifica se o contexto (*Artigo*) é válido. Para isso invoca-se a regra *REL3*. A segunda premissa verifica se o predicado é válido por meio da regra *PR7*.

REL3 Como existe um relacionamento não-identificado entre o conceito *Artigo* e o conceito *Ano*, é necessário averigüar se as premissas dessa regra são satisfeitas. As duas primeiras premissas servem para verificar se existe um relacionamento entre os conceitos mais genéricos. Elas são avaliadas pela regra *INH1*. A terceira e quarta premissas avaliam se existe um relacionamento não-identificado entre o conceito *Artigo* e o conceito *Ano*. Para esse caso, todas premissas são satisfeitas.

INH1 Para verificar se cada um dos conceitos possui relacionamentos é necessário verificar se os mais genéricos possuem relacionamentos. Mesmo não sendo o caso, as premissas são verdadeiras porque a operação de herança é reflexiva.

PR7 A verificação se o predicado é válido é trivial, pois o predicado é vazio e essa regra não contém premissas. Portanto a regra é verdadeira.

Como todas as regras foram validadas, a consulta obedece a todos critérios especificados e a consulta é considerada bem-formada em relação ao modelo conceitual.

5 EXTENSÃO DO MECANISMO DE MAPEAMENTO

O mecanismo de mapeamento de consultas apresentado em (CAMILLO; HEUSER; MELLO, 2003) mapeava diretamente uma expressão de consulta na linguagem CXPath para uma correspondente em XPath para cada fonte XML. Essa abordagem apresentava a desvantagem de necessitar que uma consulta, para navegar sobre relacionamentos que são baseados no conteúdo de elementos, tal como é feito em junções para bancos de dados relacionais, fosse escrita em CXPath montando o predicado de junção toda vez que fosse realizada.

Uma melhor solução é o mapeamento antecipado dos elementos que compõem esses tipos de relacionamentos para que seja possível a construção automática de tais predicados, permitindo uma navegação transparente para o usuário como se estivesse navegando por qualquer outro relacionamento.

A seção 5.1 apresenta o mecanismo de mapeamento original. A seção 5.2 mostra como a consulta é mapeada da linguagem *CXPath* para a linguagem *XPath*. A seção 5.3 mostra como esse trabalho foi estendido para o suporte a junções, ou seja, permitir a navegação para nodos não descendentes.

5.1 Mecanismo de Mapeamento

Para traduzir uma consulta *CXPath* para um consulta *XPath* equivalente, informações de mapeamento entre o esquema conceitual e o esquema XML são necessárias. Na literatura de integração de banco de dados, duas propostas são usualmente empregadas para definir as informações de mapeamento: um catálogo de mapeamento e visões (ELMARGAMID; RUSINKIEWCZ; SHETH, 1999).

Na proposta de um catálogo de mapeamento, as construções no esquema global são mapeadas em construções no esquema lógico. Na proposta de visões, uma consulta (*visão*) para cada conceito global e relacionamento é definida descrevendo qual o dado correspondente que deve ser obtido das fontes locais.

A proposta apresentada em (CAMILLO; HEUSER; MELLO, 2003) definiu que, para cada construção em nível conceitual, um consulta *XPath* que obtém o conjunto de instâncias na fonte XML é criada. A abordagem de visões foi usada, pois simplifica o processo de tradução, posto que cada referência para um conceito em uma consulta *CXPath* é diretamente substituída pela correspondente expressão *XPath*.

Expressões *XPath* são associadas a cada conceito e a cada navegação em um relacionamento no esquema conceitual. Além disso, para cada fonte XML, uma expressão *XPath* é associada a cada um dos conceitos representados no modelo.

```
<referencias>
  <artigo titulo='XML Overview'>
    <autor>
      <nome>João Souza</nome>
    </autor>
    <autor>
      <nome>Maria Santos</nome>
    </autor>
    <revisor>
      <nome>Pedro Silva</nome>
    </revisor>
    <revisor>
      <nome>Vera Oliveira</nome>
    </revisor>
  </artigo>
  <artigo titulo="XML Databases">
    ...
  </artigo>
</referencias>
```

(a)

```
<publicacoes>
  <autor>
    <nome>João Souza</nome>
    <artigo>
      <cabecalho>XML Overview</cabecalho>
      <ano>2003</ano>
    </artigo>
  </autor>
  <autor>
    <nome>Maria Santos</nome>
    <artigo>
      <cabecalho>XML Overview</cabecalho>
      <ano>2003</ano>
    </artigo>
    <artigo>
      <cabecalho>XML Databases</cabecalho>
      <ano>2002</ano>
    </artigo>
  </autor>
    ...
</publicacoes>
```

(b)

Figura 5.1: Exemplos de fontes XML

Para exemplificar o mapeamento entre *CXPath* e *XPath*, considere-se que o esquema conceitual apresentado na Figura 4.1 é uma abstração unificada dos fontes XML nas Figuras 5.1(a) e 5.1(b).

A Tabela 5.1 descreve as informações de mapeamento decorrentes dos conceitos e relacionamentos dos esquemas dos fontes XML.

A informação de mapeamento de um conceito para uma fonte XML é uma expressão *XPath* de caminho absoluta que define a hierarquia de elementos que devem ser buscadas para alcançar o correspondente elemento ou atributo. O mapeamento de *Artigo* e *Título* [1] para a fonte XML 1 são exemplos (veja a Tabela 5.1).

Quando um conceito possui mais de um elemento correspondente ou atributo em um fonte XML, a expressão de mapeamento deve se referir à união dos elementos. Um exemplo é o conceito *Pessoa* que é dado pela união de *autor* e *revisor* na fonte XML 1 [2]. Outro exemplo é o mapeamento do conceito *Nome* para a fonte XML 1.

A informação de mapeamento de um relacionamento $C_1 - C_2$ para uma fonte XML é uma expressão *XPath* relativa que denota como navegar do elemento correspondente ao conceito C_1 para o elemento correspondente no conceito C_2 (e vice-versa) nessa fonte.

Mapeamentos para ambas direções são definidos. O mapeamento entre o relacionamento *Pessoa-Nome* para a fonte XML 2 é um exemplo. Nessa fonte, o elemento

[1] Uma referência para um atributo é denotada em *XPath* com um @

[2] A função `local-name` retorna o nome do elemento

Tabela 5.1: Informações de mapeamento do esquema conceitual da Figura 4.1

Conceito/Relacionamento	Mapeamento para Fonte 1	Mapeamento para Fonte 2
Artigo	/referencias/artigo	/publicacoes/autor/artigo
Pessoa	/referencias/artigo/* [local-name(.)='autor' or local-name(.)='revisor']	/publicacoes/autor
Título	/referencias/artigo/@titulo	/publicacoes/autor/artigo/cabecalho
Ano	não se aplica	/publicacoes/autor/artigo/ano
Nome	/referencias/artigo/* [local-name(.)='autor' or local-name(.)='revisor']/name	/publicacoes/autor/name
Artigo{autor}→Pessoa	Autor	..
Artigo←{autor}Pessoa	[local-name(.)='autor']/..	artigo
Artigo{revisor}→Pessoa	Revisor	não se aplica
Artigo←{revisor}Pessoa	[local-name(.)='revisor']/..	não se aplica
Artigo→Título	@titulo	cabecalho
Artigo←Título	..	..
Artigo→Ano	não se aplica	Ano
Artigo←Ano	não se aplica	..
Artigo→Nome	nome	nome
Artigo←Nome	..	..

`nome` (correspondente ao conceito *Nome*) é um sub-elemento de `autor` (correspondente ao conceito *Pessoa*).

A direção da navegação Pessoa→Nome é mapeada para essa fonte XML pela expressão de caminho relativa `nome`. Essa expressão de caminho permite a navegação para os elementos `nome`, quando o contexto é um elemento `autor`. Analogamente, a direção da navegação Pessoa←Nome é mapeada para "..". Essa expressão de caminho permite a navegação na ordem inversa, ou seja, do elemento `autor`, quando o contexto é um elemento `nome`.

Um caso especial na definição de mapeamentos é um conceito C que possua mais de um elemento correspondente na fonte XML S, um para cada relacionamento R_i em que o elemento participa. Um mapeamento de um relacionamento $C \rightarrow^{\{R_i\}} C_x$ para S deve selecionar somente aqueles elementos em S que correspondam a C e façam parte do relacionamento R_i.

Um exemplo é feito pelo conceito *Pessoa* e o relacionamento *revisor* e *autor* quando a fonte 1 é considerada. Nessa fonte, o conceito *Pessoa* é mapeado em dois elementos, `autor` e `revisor`. Uma instância de *Pessoa* que mapeia para um elemento `autor` faz parte do relacionamento *autor*, ao passo que uma instância de *Pessoa* que mapeia para um elemento `revisor` faz parte do relacionamento *revisor*. Se uma navegação inicia com um elemento representando *Pessoa*, a expressão *XPath* deve assegurar que o elemento correto (no exemplo, um `autor` ou um `revisor`) será usado como origem na navegação.

Isso é obtido por meio da função `local-name()`, a qual resulta no nome do elemento. Então, o mapeamento de $Artigo^{\{revisor\}} \leftarrow Pessoa$ para a fonte XML 1, como mostrado na Tabela 5.1, corresponde à seguinte expressão *XPath*:

$$[\ \text{local-name(.)} = \text{“revisor”}\ ]\ /\ ..$$

Esse predicado garante que o elemento de origem da navegação é um elemento do tipo `revisor`, antes de navegar para o elemento `artigo`, que representa o conceito *Artigo*.

Em geral, predicados *XPath* devem ser definidos em uma informação de mapeamento toda vez que uma semântica específica deva ser avaliada para dar o correto mapeamento.

Para exemplificar outra situação, suponha que um conceito *Pessoa* tenha dois relacionamentos com um conceito *Endereço* chamados *Casa* e *Escritório* (endereços de casa e comercial de uma pessoa). Se um fonte XML define um elemento `pessoa` com dois sub-elementos `endereço`, onde o primeiro é o endereço de casa, o mapeamento do relacionamento $Pessoa^{\{casa\}} \rightarrow Endereço$ para essa fonte deveria ser `Endereço[position()=1]` [3].

5.2 Tradução *CXPath* para *XPath*

A tradução de *CXPath* para *XPath* utiliza a estratégia da reescrita, isto é, para cada referência a um conceito ou a um relacionamento na consulta *CXPath* é substituída pela correspondente informação de mapeamento considerada na fonte XML.

O processo de tradução de uma consulta *CXPath* de entrada para uma consulta *XPath* de saída para um documento fonte XML específico é feito da seguinte forma:

- a entrada é avaliada da esquerda para a direita;
- a cada *token* encontrado na entrada é escrito na saída um correspondente, com algumas exceções:
  - a primeira "/" de uma expressão de caminho absoluta não é escrita na saída;
  - cada *conceito* encontrado na entrada será substituído pela expressão *XPath* correspondente;
  - um operador de navegação qualificada na forma { *Relationship* } ou { *Relationship.Rolename* } é substituído na saída pelo operador "/".
- quando o primeiro conceito C_a de uma expressão *CXPath* de caminho absoluta é encontrado, a expressão *XPath* que mapeia o conceito C_a para a fonte é escrita na saída. Essa é sempre uma expressão de caminho absoluta.
- cada outro conceito C_r encontrado na consulta *CXPath* de entrada possui um contexto, ou seja, a expressão é relativa a outro conceito C_c. Quando tal conceito relativo C_r é encontrado na entrada, a expressão *XPath* que mapeia a navegação do relacionamento a partir do conceito C_c, que descreve o contexto, para o conceito relativo C_r ($C_c \rightarrow C_r$) é escrita na saída. Se o conceito C_r é precedido por uma expressão na forma { *Relationship* } ou { *Relationship.Rolename* }, essa expressão é usada para selecionar qual o mapeamento será usado.

5.3 Suporte a Relacionamentos Baseados no Conteúdo dos Elementos

Hoje, são comuns as ferramentas que exportam dados do formato relacional para XML (ORACLE CORPORATION, 2005; MICROSOFT CORPORATION,

[3] A função `position()` retorna a posição do sub-elemento Endereço

2005; IBM CORPORATION, 2005; MICROSOFT CORPORATION, 2001). Tais ferramentas, na maioria das vezes, exportam seus dados da mesma forma como eles estavam representados em suas estruturas de tabelas. Ou seja, apesar de os dados passarem para a estrutura de XML, eles ainda continuam com seus apontamentos dados por referências de chaves primárias e estrangeiras.

Portanto, ainda que os dados passem para a estrutura XML, muitas vezes eles permanecem com sua estrutura lógica intacta. Considerando o exemplo do modelo ER apresentado na figura 5.2, um exemplo de exportação para XML pode ser visto conforme o exemplo da figura 5.3

Figura 5.2: Modelo ER a ser exportado para a representação XML

```
<modelo>
  <autor>
    <cpf>123.456.789-01</cpf>
    <nome>Diego Feijó</nome>
  </autor>
  <artigo>
    <id>1</id>
    <titulo>Consultando XML por meio de Modelos Conceituais</titulo>
  </artigo>
  <autor_artigo>
    <cpf>123.456.789-01</cpf>
    <id>1</id>
  </autor_artigo>
</modelo>
```

Figura 5.3: Exemplo de documento gerado pelas ferramentas que exportam em formato XML

A representação conceitual do documento XML apresentado na figura 5.3 pode ser vista na figura 5.4. Nessa figura estão omitidos os conceitos léxicos que representam os atributos do modelo ER.

A forma de navegação por meio desses relacionamentos teria de ser escrita construindo o predicado de junção se fosse considerada o mecanismo de mapeamento original. Com a extensão proposta, as consultas podem ser escritas assumindo que os relacionamentos mostrados na figura 5.4 são relacionamentos associativos como os outros, por isso sua navegação torna-se mais adequada.

***Observação* 3** *Embora a abordagem XML não contemple o uso de chaves de junção ou restrições de integridade, o termo chave de junção será usado nesse texto para se referir à definição de critérios que permitem a navegação de um conceito para*

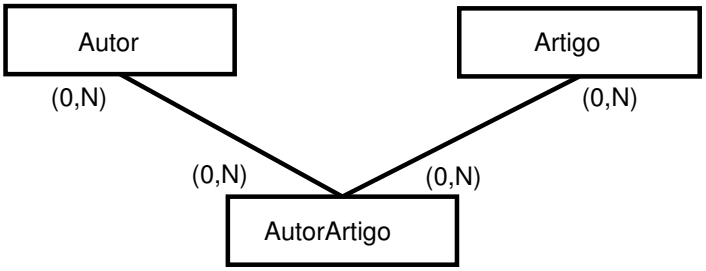

Figura 5.4: Modelo Conceitual sem a representação dos elementos léxicos (atributos)

```
-Para cada conceito da consulta CXPath
   -Procure na tabela de mapeamentos o token correspondente
      -Se existir, substitua e fique com o ultimo conceito
      -Se nao existir, procure pelo próximo conceito
      -Se o conceito anterior combinado com o atual estiver na tabela
      de juncoes há uma juncao
      -Para cada uma das chaves de junção
         -Adicione a chave de junção da tabela de junções
         -Guarde em uma lista de chaves de junçao
         -Monte a consulta usando a expressão designada na tabela com
         o predicado
      -Para cada chave de junção da tabela
         -Monte o elemento de junção com a chave de junção oposta e
         compare com o enésimo elemento da lista de chaves de junção
```

Figura 5.5: Extensão do algoritmo de mapeamento de consultas

o seu correspondente. Supõe-se, portanto, que existe um critério no fonte que possibilite estabelecer um vínculo entre conceitos que se referiram à mesma informação.

Com essa motivação, o suporte a junções foi implementado. A representação desses relacionamentos poderia ser feita do mesmo modo simplesmente por reescrita. No entanto, a representação terá de ser explícita. Esse não é o objetivo, porque a representação conceitual irá mapear diretamente o relacionamento, ainda que ele não exista dessa forma nas fontes.

5.4 Algoritmo de Mapeamento

O algoritmo de mapeamento de consulta utilizado em (CAMILLO; HEUSER; MELLO, 2003) realiza uma substituição direta de cada conceito para a expressão de caminho correspondente no documento fonte. Ou seja, o algoritmo varre a consulta e para cada conceito encontrado era buscada a expressão de caminho que o representa em cada um dos fontes.

Esse algoritmo foi estendido para permitir junções. Para isso, algumas modificações no algoritmo foram necessárias, visto que a estratégia de substituição direta adotada não funciona com essa extensão. O algoritmo estendido está descrito na Figura 5.5.

O algoritmo varre a consulta da esquerda para a direita. A cada conceito identificado, é procurado um elemento correspondente na lista de mapeamentos diretos, tal como era feito pela versão original do algoritmo. Caso não haja substituição direta, é verificado se existe uma junção. Para isso, é verificado se o conceito atual concatenado com o anterior estão representados na tabela de junções. Caso estejam, a junção foi identificada.

Portanto, a tabela de mapeamentos teve de ser estendida para contemplar as informações adicionais que ela conterá. Essas informações dizem respeito ao elemento que irá estabeler o elo do relacionamento, funcionando como uma chave de junção. Mesmo com mais de um elemento, o algoritmo consegue construir o predicado de forma correta.

A reescrita da consulta a partir de uma junção identificada é feita concatenando-se a consulta com a chave de junção. Esse processo é realizado para cada uma das n chaves de junções. A seguir, a consulta XPath é montada usando-se a consulta descrita na tabela, acrescentando-se como predicado de tal consulta os elementos que o constituem.

Os elementos que compõe a junção são formados usando-se as chaves opostas, que estão na tabela, comparando-os com as consultas que foram geradas anteriormente. Cada uma dos elementos é unido por um *AND* (E lógico), para formar o predicado que compõe a junção.

Para facilitar o entendimento do algoritmo, é apresentado, na seção seguinte, um exemplo que utiliza junção.

5.5 Exemplo de Junção

A estrutura do documento XML do exemplo é exemplificada a seguir na Figura 5.6. É possível notar as relações existentes entre os conceitos, onde claramente os códigos das disciplinas estabelecem um relacionamento.

No processo de integração de fontes como descrito em (MELLO; HEUSER, 2001b), essa estrutura seria mapeada para um modelo conceitual como o descrito na Figura 5.7. Esse modelo contém relações que não estão presentes na estrutura original. Nesse caso, uma consulta em *XPath* como a mostrada a seguir não poderia ser resolvida da forma mostrada anteriormente.

```
/Aluno[Nome="Fulano de Tal"]/Histórico[Nota="A"]/Disciplina/Nome
```

A semântica da consulta seria: "Obter o nome das disciplinas em que o aluno 'Fulano de Tal' obteve a nota A". Essa consulta deve ser resolvida através de uma junção. No entanto, essa junção surgiu em decorrência da construção do modelo conceitual. Portanto, é necessário determinar um modo que permita que as consultas expressas sobre o modelo conceitual global, sejam traduzidas para o esquema das fontes.

Na Figura 5.7 está destacada através de um risco mais espesso a relação que inexiste nos documentos. Essa é a relação representada através da junção mencionada anteriormente. Por isso, é necessário algum mecanismo para permitir realizar essa junção.

Embora o exemplo mostre uma situação que não é muito realista, por representar uma junção com uma "chave de junção" bastante aparente, em muitos casos, é possível que esse conceito não esteja presente. Nesses casos, é bastante provável

```
<cursos>
  <curso>
    <codigo>102-05</codigo>
    <nome>CIC</nome>
    <curriculo>
      <disciplina>
        <codigo>INF01145</codigo>
        <nome>Fund de BD</nome>
      </disciplina>
      <disciplina>
        <codigo>INF01007</codigo>
        <nome>Intr Arq Comp</nome>
      </disciplina>
      <disciplina>
        <codigo>INF01008</codigo>
        <nome>Arq Comp I</nome>
      </disciplina>
    </curriculo>
  </curso>
  <alunos>
    <aluno>
      <nome>Fulano de Tal</nome>
      <curso>102-05</curso>
      <historico>
        <disciplina>
          <codigo>INF01145</codigo>
          <conceito>A</conceito>
        </disciplina>
        <disciplina>
          <codigo>INF01112</codigo>
          <conceito>B</conceito>
        </disciplina>
        <disciplina>
          <codigo>INF01008</codigo>
          <conceito>A</conceito>
        </disciplina>
      </historico>
    </aluno>
  </alunos>
</cursos>
```

Figura 5.6: Informações complementares em um mesmo documento

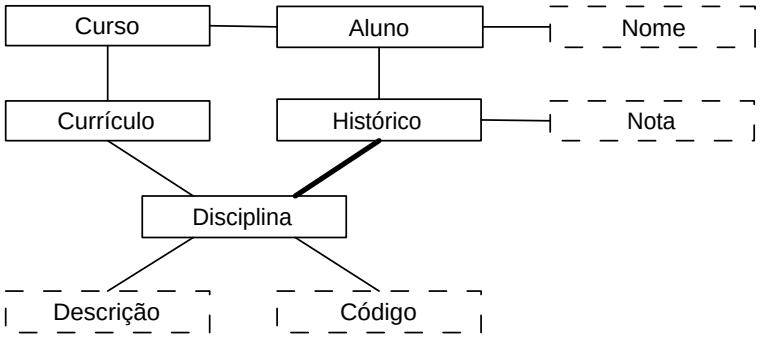

Figura 5.7: Modelo conceitual gerado a partir do documento fonte

que a consulta não possa ser resolvida pois não exista nenhum mecanismo capaz de determinar quais conceitos se relacionam e através de quais "chaves".

A proposta apresentada restringe-se, portanto, para as situações onde essas relações possam ser estabelecidas. Em outras palavras, somente nos casos onde houver um elo que permita realizar a operação de junção. Para os demais casos seria necessário estabelecer algum critério diferenciado. Nesse sentido, existem trabalhos que utilizam similaridade para resolver os casos onde não existem os elos aqui necessários. Contudo esse não é o foco desta seção.

Nesse método, é necessário definir elementos que atuem como chaves, definir as estruturas que irão armazenar essas informações além de estabelecer a metodologia para realizar a combinação entre a teoria de junções aplicada a consultas através de expressões de caminho.

As informações extras necessárias para possibilitar as junções são as chaves *primária* e *estrangeira* dos relacionamentos de junção. No entanto, essas chaves não estão presentes e diretamente relacionadas tal como é feita em uma abordagem relacional. Portanto, é necessário que o encarregado pela construção do modelo conceitual informe quais as informações relevantes para o estabelecimento de critérios para a realização de uma junção.

As informações para o estabelecimento de um vínculo entre conceitos que representam um mesmo conceito fazem parte do conjunto de informações de mapeamento que o especialista deve identificar durante a construção do modelo conceitual. Os conceitos serão identificados nos fontes XML como *elementos*, *atributos* ou mesmo *expressões de caminho*.

As chaves servem para estabelecer o critério de junção. Esse critério é mapeado para um predicado na seguinte forma:

```
[ <chave-primária> = <chave-estrangeira> ]
```

A chave estrangeira é a chave primária do relacionamento inverso. Portanto, essa informação poderia até ser omitida. No entanto, essa informação teria sempre de ser buscada. Portanto, mesmo causando certa redudância de informação, essa informação estará presente no exemplo que segue.

A realização do mapeamento de consultas globais (consultas sobre o modelo conceitual global) para consultas locais (consultas sobre as fontes XML) será feita por meio de dois processos. O primeiro realiza o mapeamento dos termos que podem ser mapeados diretamente; o segundo, das junções que devem ser executadas.

O primeiro processo consiste em identificar cada termo e mapeá-lo para seu correspondente. Para isso, é necessária a criação de uma tabela contendo cada termo e sua tradução para a fonte correspondente. Cada fonte possuirá uma tabela distinta que identificará, para cada termo da expressão de caminho, o seu mapeamento.

O segundo processo consiste em identificar relacionamentos que inexistem nas fontes, mas podem ser resolvidos através de junções sobre elementos das fontes. Para cada termo, é preciso verificar qual será o elemento que será considerado na junção e adaptá-lo. A consulta modificada tornar-se-á parte da junção. A outra parte é fornecida pelo mapeamento indicando como será feita a junção.

Seguindo essa idéia, as informações de mapeamento necessárias são o nome do elemento de junção e o caminho correspondente para formação da junção. Para ilustrar esse funcionamento, é apresentado a seguir um exemplo de um documento XML e o modelo conceitual para o qual foi mapeado.

O documento fonte XML ilustrado na Figura 5.8 tem o seu correspondente modelo conceitual mostrado na Figura 5.9. Há alunos que possuem um histórico que contém a relação das turmas que já foram cursadas.

Pode-se perceber que o relacionamento entre o conceito *Aluno* e o conceito *Turma* é feito por meio dos conceitos *Semestre* e *Código*. Portanto, para navegar de *Aluno* para *Turma*, ou o contrário, seria necessário a escolha de um dos caminhos. Nessa situação, percebe-se que a modelagem não reflete a semântica do documento, pois o ideal seria um relacionamento direto entre *Aluno* e *Turma*.

A Figura 5.10 exibe como o modelo foi adaptado para refletir a semântica do documento e permitir a navegação direta entre *Aluno* e *Turma*. Embora esse modelo represente o ideal, ele contém caminhos que inexistem no documento. Por isso, para permitir tal relacionamento é necessário que o modelo permita a navegação por meio de junções com mais de uma chave.

O exemplo ilustrado pelo passo-a-passo da Tabela 5.3 demonstra como é feito o mapeamento para chaves compostas. Deseja-se obter o nome das disciplinas já cursadas pelo aluno chamado "Joao da Silva". Essa consulta poderia ser expressa, na linguagem *CXPath*, da forma como mostrado na Figura 5.11.

Para mapear a consulta global da Figura 5.11, será preciso resolver duas junções. A primeira junção é composta por duas chaves. A segunda junção é composta por um única chave e é resolvida como mostrado no exemplo anterior.

A primeira junção envolve os conceitos *Aluno* e *Turma*. Seguindo os dados da Tabela 5.2 para o relacionamento *Turma*←*Aluno*, a primeira chave primária é `historico/turma/sem`; a segunda, `historico/turma/cod-cad`. A consulta local correspondente é montada utilizando-se a informação de mapeamento `/universidade/turmas/turma`, adicionando-se um predicado com o seguinte formato:

[¡chave-primária$_1$¿ = ¡consulta-anterior¿ / ¡chave-estrangeira$_1$¿ *and*
¡chave-primária$_2$¿ = ¡consulta-anterior¿ / ¡chave-estrangeira$_2$¿ *and*
¡chave-primária$_n$¿ = ¡consulta-anterior¿ / ¡chave-estrangeira$_n$¿]

A Tabela 5.3 ilustra as etapas resumidas envolvidas no processo de mapeamento.

A Figura 5.12 ilustra a consulta local resultante do mapeamento da consulta global mostrada na Figura 5.11. A consulta resultante demonstra a complexidade de expressar junções na linguagem *XPath* e como o mecanismo de mapeamento evita que o usuário tenha de preocupar-se com detalhes específicos da fonte em

```
<universidade>
  <turmas>
    <turma>
      <semestre>2004/2</semestre>
      <cod-disc>INF01001</cod-disc>
    </turma>
  </turmas>
  <disciplinas>
    <disciplina>
      <codigo>INF01001</codigo>
      <denominacao>Disciplina Exemplo</denominacao>
    </disciplina>
  </disciplinas>
  <alunos>
    <aluno>
      <matricula>0001/04-0</matricula>
      <nome>Fulano da Silva</nome>
      <historico>
        <cadeira>
         <sem>2004/2</sem>
         <cod-cad>INF01001</cod-cad>
        </cadeira>
      </historico>
    </aluno>
  </alunos>
</universidade>
```

Figura 5.8: Fonte XML com informações sobre alunos e seu histórico

Tabela 5.2: Informações de mapeamento do esquema conceitual da Figura 5.10

Conc/Rel	Mapeamento	Chave Prim 1	Chave Prim 2
Código	/universidade/disciplinas /disciplina/codigo	-	-
Disciplina	/universidade/disciplinas /disciplina	-	-
Semestre	/universidade/turmas /turma/semestre	-	-
Aluno	/universidade/alunos /aluno	-	-
Turma	/universidade/turmas /turma	-	-
Denominação	/universidade/disciplinas /disciplina/denominacao	-	-
Nome	/universidade/alunos /aluno/nome	-	-
Código→Disciplina	..	-	-
Código←Disciplina	codigo	-	-
Disciplina→Denominação	denominacao	-	-
Disciplina←Denominação	..	-	-
Disciplina→Turma	/universidade/turmas /turma	codigo	-
Disciplina←Turma	/universidade/disciplinas /disciplina	cod-disc	-
Turma→Semestre	semestre	-	-
Turma←Semestre	..	-	-
Turma→Aluno	/universidade/alunos /aluno	semestre	cod-disc
Turma←Aluno	/universidade/turmas /turma	historico/cadeira/sem	historico/cadeira/cod-cad
Matrícula→Aluno	..	-	-
Matrícula←Aluno	matricula	-	-
Aluno→Nome	nome	-	-
Aluno←Nome	..	-	-

Tabela 5.3: Etapas da Operação de Junção do Exemplo

Etapa: 0	-
Consulta Global:	/Aluno[Nome="Joao da Silva"]/Turma/Disciplina/Denominacao
Consulta Local:	-
Etapa: 1	/Aluno
Consulta Global:	**/Aluno**[Nome="Joao da Silva"]/Turma/Disciplina/Denominacao
Consulta Local:	**/universidade/alunos/aluno**
Etapa: 2	Aluno→Nome
Consulta Global:	/Aluno**[Nome="Joao da Silva"]**/Turma/Disciplina/Denominacao
Consulta Local:	/universidade/alunos/aluno**[nome="Joao da Silva"]**
Etapa: 3	Turma←Aluno
Consulta Global:	/Aluno[Nome="Joao da Silva"]**/Turma**/Disciplina/Denominacao
Consulta Local:	**/universidade/turmas/turma[semestre=** /universidade/alunos/aluno[Nome="Joao da Silva"] **/historico/turma/sem *and*** cod-disc=/universidade/alunos/aluno[Nome="Joao da Silva"] **/historico/turma/cod-cad]**
Etapa: 4	Disciplina←Turma
Consulta Global:	/Aluno[Nome="Joao da Silva"]/Turma**/Disciplina**/Denominacao
Consulta Local:	**/universidade/disciplinas/disciplina[codigo=** /universidade/turmas/turma[semestre= /universidade/alunos/aluno[Nome="Joao da Silva"] /historico/turma/sem *and* cod-disc=/universidade/alunos/aluno[Nome="Joao da Silva"] /historico/turma/cod-cad]**/cod-disc]**
Etapa: 5	Disciplina→Denominação
Consulta Global:	/Aluno[Nome="Joao da Silva"]/Turma/Disciplina**/Denominacao**
Consulta Local:	/universidade/disciplinas/disciplina[codigo= /universidade/turmas/turma[semestre= /universidade/alunos/aluno[Nome="Joao da Silva"] /historico/turma/sem *and* cod-disc=/universidade/alunos/aluno[Nome="Joao da Silva"] /historico/turma/cod-cad]/cod-disc]**/denominacao**

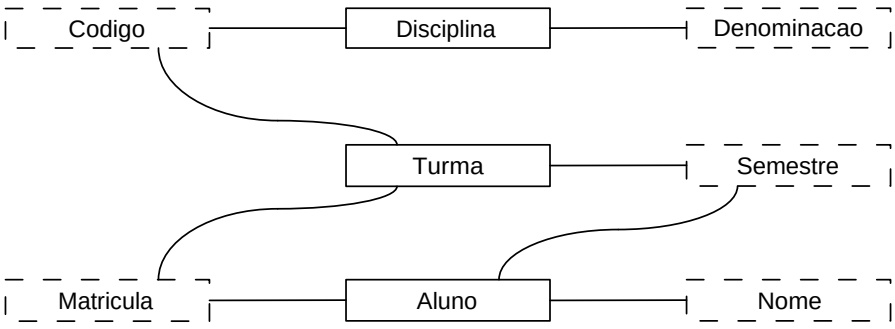

Figura 5.9: Modelo conceitual com relacionamentos físicos gerado a partir do documento fonte

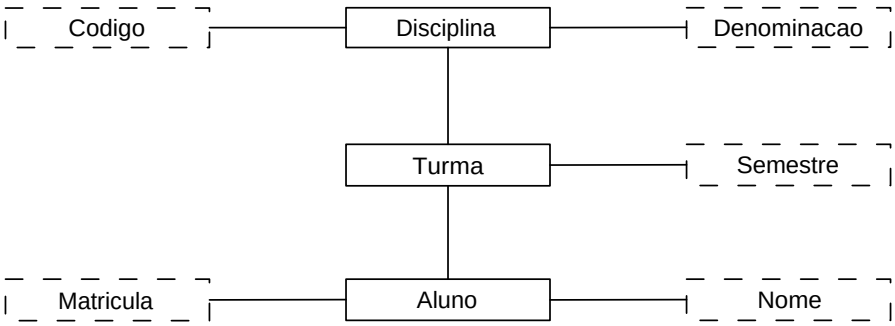

Figura 5.10: Modelo conceitual com relacionamentos semânticos gerado a partir do documento fonte

que a consulta está sendo executada. Resta ainda lembrar que, sem o mecanismo de mapeamento, a consulta teria de ser reescrita pelo usuário para cada um dos documentos fontes.

```
/Aluno[Nome="Joao da Silva"]/Turma/Disciplina/Denominacao
```

Figura 5.11: Consulta global sobre a base de alunos

```
/universidade/disciplinas/disciplina
  [
    codigo=
      /universidade/turmas/turma
        [
          semestre=
            /universidade/alunos/aluno
              [
                nome="Joao da Silva"
              ]
              /historico/cadeira/sem and
          cod-disc=
            /universidade/alunos/aluno
              [
                nome="Joao da Silva"
              ]
              /historico/cadeira/cod-cad
        ]
        /cod-disc
  ]
  /denominacao
```

Figura 5.12: Consulta local resultante do processo de mapeamento do Exemplo

6 CONCLUSÕES

A construção de um modelo conceitual conceitual global é essencial para reduzir a complexidade de consultar fontes de dados XML. Cada fonte possui uma estrutura diferente o que torna o trabalho de consultar uma tarefa demasiado complicada.

Nesse contexto, uma linguagem de consulta é peça chave para permitir que o modelo conceitual seja consultado no nível conceitual. A linguagem *CXPath* permite que as consultas sejam elaboradas diretamente contra o modelo conceitual permitindo uma navegação intuitiva tendo uma sintaxe semelhante à já conhecida da linguagem *XPath*.

A principal contribuição desse trabalho foi a especificação formal de um critério para validar consultas *CXPath* contra modelos conceituais. A formalização, na forma de um conjunto de regras de inferência similares àquelas usadas para especificação da semântica de linguagens de programação (PIERCE, 2002), fornece uma descrição concisa e precisa das propriedades estáticas da linguagem *CXPath*. Tal descrição serve como referência para implementadores da linguagem, planejadores de consultas e os próprios usuários.

A formalização proposta é etapa essencial para estabelecer resultados fundamentais sobre a tarefa de consultar fontes XML, dentre os quais é possível mencionar o seguinte: é preciso provar formalmente que o resultado da avaliação de consultas *CXPath* sobre um modelo conceitual global conduz aos mesmos resultados da avaliação das consultas *XPath* resultantes do processo de mapeamento contra os documentos que compõem o modelo. A prova desse fato depende da garantia de que as consultas *CXPath* são válidas se confrontadas com o modelo conceitual considerado.

Outro possível trabalho futuro ainda quanto a formalização de linguagem seria na área de otimização de consultas: com o objetivo de provar a correção de um processo de otimização para consultas *CXPath*, é preciso provar que a consulta otimizada também é válida se confrontada contra o esquema conceitual do modelo. Essa é uma etapa fundamental para provar que, a consulta original e a otimizada, produzem o mesmo resultado.

Resta ainda definir uma variação da linguagem *XQuery* para consultar modelos conceituais. Como a linguagem *XQuery* contém a linguagem *XPath*, o uso de *XQuery+CXPath* como uma linguagem de consulta no nível conceitual, e a formalização de seu critério de validação ainda devem ser investigados.

O trabalho de extensão do mecanismo de mapeamento, para suportar relacionamentos que inexistem diretamente, mas são adequados semanticamente por meio do uso de junções, possibilita a integração de documentos XML resultantes do uso de ferramentas que exportam informações contidas em bases relacionais. A navegação utilizando o modelo conceitual é mais intuitiva e evita a necessidade de conhecer os

detalhes específicos sobre como cada fonte está organizada. Esse estilo de navegação dificulta a construção de junções e, portanto, torna inadequado o seu uso a menos que as junções sejam feitas de modo automático.

É necessário também estender o mecanismo de mapeamento e o processo de tradução para permitir decomposição de consultas. A decomposição serviria para definir quais os fontes XML, supondo um modelo conceitual construído a partir de muitos documentos, possuíram dados de interesse da consulta. Essa situação é fundamental para permitir a construção de modelos conceituais complexos, que envolveriam grandes quantidades de documentos, mas que ainda possuíram um tempo para avaliação das aceitável por evitar que fontes que não possuam dados relevantes à consulta fossem consultadas.

Apesar de a extensão do mecanismo de mapeamento já suportar junções, ainda há outra necessidade que resta para ser suprida que é a definição de um mecanismo de mapeamento para suportar herança. As regras que definem como uma consulta com herança é avaliada contra um esquema conceitual já integram parte deste trabalho, mas resta ainda definir como o mecanismo de mapeamento traduz essas consultas para a linguagem *XPath* que não suporta tal característica.

REFERÊNCIAS

BIERMAN, G. M. Formal Semantics and analysis of object queries. In: ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, 2003, San Diego, CA. **Proceedings...** [S.l.: s.n.], 2003. p.407–418.

BRADLEY, N. **XML Companion**. 2nd.ed. Boston: Addison-Wesley, 2000.

CAMILLO, S. D.; HEUSER, C. A.; MELLO, R. D. S. Querying Heterogeneous XML Sources through a Conceptual Schema. In: INTERNATIONAL CONFERENCE ON CONCEPTUAL MODELING, 22., 2003, Chigago, IL, 2003. **Proceedings...** Springer, 2003. p.186–199. (Lecture Notes in Computer Science, v.2813).

CHAMBERLIN, D.; ROBIE, J.; FLORESCU, D. Quilt: an xml query language for heterogeneous data sources. In: INTERNATIONAL WORKSHOP WEBDB 2000 ON THE WEB AND DATABASES, WebDB, 3., 2001. **Proceedings...** London: Springer-Verlag, 2001. p.1–25. (Lecture Notes in Computer Science, v.1997).

DATE, C. J. Some principles of good language design. **ACM SIGMOD Record**, New York, v.14, n.3, p.1–7, 1984.

ELMARGAMID, A.; RUSINKIEWCZ, M.; SHETH, A. **A Management of Heterogeneous and Autonomous Database Systems**. San Francisco: Morgan Kauffmann, 1999.

FREUND, S.; MITCHELL, J. A Type System for the Java Bytecode Language and Verifier. **J. Autom. Reasoning**, [S.l.], v.30, n.3-4, p.271–321, 2003.

HOSOYA, H.; PIERCE, B. C. XDuce: a typed XML processing language. In: INTERNATIONAL WORKSHOP ON THE WEB AND DATABASES, WebDB, 2000, Dallas, TX. **Proceedings...** [S.l.: s.n.], 2000.

HOSOYA, H.; PIERCE, B. C. Regular expression pattern matching for XML. In: ACM SYMPOSIUM ON PRINCIPLES OF PROGRAMMING LANGUAGES, 28., 2001. **Proceedings...** London: ACM Press, 2001. p.67–80.

HOSOYA, H.; VOUILLON, J.; PIERCE, B. C. Regular expression types for XML. **ACM SIGPLAN Notices**, [S.l.], v.35, n.9, p.11–22, 2000.

IBM CORPORATION. **IBM DB2 XML Extender**. Disponível em: <http://www-306.ibm.com/software/data/db2/extenders/xmlext/>. Acesso em: 10 jun. 2005.

MCKINNA, J.; POLLACK, R. Pure Type Systems Formalized. In: INTERNATIONAL CONFERENCE ON TYPED LAMBDA CALCULI AND APPLICATIONS, TLCA, 1., 1993, Utrecht, The Netherlands. **Proceedings...** Berlin: Springer-Verlag, 1993. p.289–305. (Lecture Notes in Computer Science, v.664).

MELLO, R. D. S.; HEUSER, C. A. A Bottom-Up Approach for Integration of XML Sources. In: WORKSHOP ON INFORMATION INTEGRATION ON THE WEB, 2001, Rio de Janeiro. **Proceedings...** [S.l.: s.n.], 2001. p.118–124.

MELLO, R. D. S.; HEUSER, C. A. A rule-based conversion of a DTD to a conceptual schema. In: INTERNATIONAL CONFERENCE ON CONCEPTUAL MODELING, 20., 2001, Yokohama, Japan. **Proceedings...** Berlin: Springer-Verlang, 2001. p.133–148. (Lecture Notes in Computer Science, v.2224).

MICROSOFT CORPORATION. **XML Perspective**. 2001. Disponível em: <http://msdn.microsoft.com/library/periodic/period01/xmlExplicit.htm>. Acesso em: 10 jun. 2005.

MICROSOFT CORPORATION. **Writing XML Using OPENXML**. Disponível em: <http://msdn.microsoft.com/library/psdk/sql/ac_openxml_94mk.htm>. Acesso em: 10 jun. 2005.

MILO, T.; SUCIU, D.; VIANU, V. Typechecking for XML transformers. **J. Comput. Syst. Sci.**, [S.l.], v.66, n.1, p.66–97, 2003.

ORACLE CORPORATION. **Oracle XML-SQL Utility**. Disponível em: <http://otn.oracle.com/tech/xml/oracle_xsu/>. Acesso em: 10 jun. 2005.

PERST, T.; SEIDL, H. A type-safe macro system for XML. In: EXTREME MARKUP LANGUAGES, 2., 2002. **Proceedings...** [S.l.: s.n.], 2002.

PIERCE, B. **Types and Programming Languages**. Cambridge, MA: MIT Press, 2002.

PIERCE, B. C. **Advanced Topics in Types and Programming Languages**. [S.l.]: Cambridge, MA: MIT Press, 2005.

PRATT, V. R. Semantical Considerations on Floyd-Hoare Logic. In: IEEE SYMPOSIUM ON FOUNDATIONS OF COMPUTER SCIENCE, 1976. **Proceedings...** [S.l.: s.n.], 1976. p.109–121.

SIMÉON, J.; WADLER, P. The essence of XML. In: ACM SYMPOSIUM ON PRINCIPLES OF PROGRAMMING LANGUAGES, 30., 2003, New Orleans, Louisiana. **Proceedings...** [S.l.: s.n.], 2003. p.1–13.

WORLD WIDE WEB CONSORTIUM. **XQuery 1.0 and XPath 2.0 Formal Semantics**. Disponível em: <http://www.w3.org/TR/2003/WD-xquery-semantics-20030822/> Acesso em: 10 jun. 2005.

WORLD WIDE WEB CONSORTIUM. **Extensible Markup Language-XML**. Disponível em: <http://www.w3c.org/XML>. Acesso em: 10 jun. 2005.

WORLD WIDE WEB CONSORTIUM. **XQuery 1.0 and XPath 2.0 Data Model**. Disponível em: <http://www.w3c.org/TR/query-datamodel>. Acesso em: 10 jun. 2005.

WORLD WIDE WEB CONSORTIUM. **XML Schema**. Disponível em: <http://www.w3c.org/TR/Schema>. Acesso em: 10 jun. 2005.

WORLD WIDE WEB CONSORTIUM. **XML Query Language- XQuery**. Disponível em: <http://www.w3c.org/TR/xquery>. Acesso em: 10 jun. 2005.